References

Abdullah, H. M., & Zeki, A. M. (2014). Frontend and Backend Web Technologies in Social Networking Sites: Facebook as an Example. 2014 3rd International Conference on Advanced Computer Science Applications and Technologies. https://doi.org/10.1109/acsat.2014.22

Ahmad, P., Qamar, S., & Qasim Afser Rizvi, S. (2015). Techniques of Data Mining In Healthcare: A Review. International Journal of Computer Applications, 120(15), 38–50. https://doi.org/10.5120/21307-4126

Anaconda Software Distribution. (2018a). Retrieved from Anaconda website: https://www.anaconda.com/

Baştanlar, Y., & Özuysal, M. (2013). Introduction to Machine Learning. MiRNomics: MicroRNA Biology and Computational Analysis, 1107, 105–128. https://doi.org/10.1007/978-1-62703-748-8_7

Beretta, L., & Santaniello, A. (2016). Nearest neighbor imputation algorithms: a critical evaluation. BMC Medical Informatics and Decision Making, 16(S3). https://doi.org/10.1186/s12911-016-0318-z

Bertsimas, D., Bjarnadóttir, M. V., Kane, M. A., Kryder, J. C., Pandey, R., Vempala, S., & Wang, G. (2008). Algorithmic Prediction of Health-Care Costs. Operations Research, 56(6), 1382–1392. https://doi.org/10.1287/opre.1080.0619

Bodenreider, O., Cornet, R., & Vreeman, D. (2018a). Recent Developments in Clinical Terminologies — SNOMED CT, LOINC, and RxNorm. Yearbook of Medical Informatics, 27(01), 129–139. https://doi.org/10.1055/s-0038-1667077

Bowd, C., Weinreb, R. N., Balasubramanian, M., Lee, I., Jang, G., Yousefi, S., … Goldbaum, M. H. (2014). Glaucomatous Patterns in Frequency Doubling Technology (FDT) Perimetry Data Identified by Unsupervised Machine Learning Classifiers. PLoS ONE, 9(1), e85941. https://doi.org/10.1371/journal.pone.0085941

C, A. D., T, L. W., A, S. C., J, R. A., Chelluri, L., Newbold, R C, 3rd, … R, P. M. (1996). The effect of managed care on ICU length of stay: implications for medicare. JAMA, 276(13), 1075–1082. Retrieved from https://pubmed.ncbi.nlm.nih.gov/8847771

Centers for Disease Control and Prevention. (2019). ICD - ICD-10-CM - International Classification of Diseases, Tenth Revision, Clinical Modification. Retrieved from National Center for Health Statistics website: https://www.cdc.gov/nchs/icd/icd10cm.htm

Chang, C.-L., & Chen, C.-H. (2009). Applying decision tree and neural network to increase quality of dermatologic diagnosis. Expert Systems with Applications, 36(2), 4035–4041. https://doi.org/10.1016/j.eswa.2008.03.007

Chekroud, A. M., Zotti, R. J., Shehzad, Z., Gueorguieva, R., Johnson, M. K., Trivedi, M. H., … Corlett, P. R. (2016). Cross-trial prediction of treatment outcome in depression: a machine learning approach. The Lancet Psychiatry, 3(3), 243–250. https://doi.org/10.1016/s2215-0366(15)00471-x

Chen, R., Kumar, V., Fitch, N., Jagadish, J., Zhang, L., Dunn, W., & Chau, D. H. (2015). explICU: A web-based visualization and predictive modeling toolkit for mortality in intensive care patients. 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 6830–6833. https://doi.org/10.1109/embc.2015.7319962

Chen, T., & Guestrin, C. (2016). XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (pp. 785-794). https://doi.org/10.1145/2939672.2939785

CITI - Collaborative Institutional Training Initiative. (n.d.). Retrieved from CITI Program website: https://www.citiprogram.org/index.cfm?pageID=14

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement, 20(1), 37–46. https://doi.org/10.1177/001316446002000104

Cowie, M. R., Blomster, J. I., Curtis, L. H., Duclaux, S., Ford, I., Fritz, F., … Zalewski, A. (2016). Electronic health records to facilitate clinical research. Clinical Research in Cardiology, 106(1), 1–9. https://doi.org/10.1007/s00392-016-1025-6

Cuperlovic-Culf, M. (2018). Machine Learning Methods for Analysis of Metabolic Data and Metabolic Pathway Modeling. Metabolites, 8(1), 4. https://doi.org/10.3390/metabo8010004

Dahl, D., Wojtal, G. G., Breslow, M. J., Huguez, D., Stone, D., & Korpi, G. (2012). The High Cost of Low-Acuity ICU Outliers. Journal of Healthcare Management, 57(6), 421–433. https://doi.org/10.1097/00115514-201211000-00009

Dasgupta, Abhijit, Sun, Y. V., König, I. R., Bailey-Wilson, J. E., & Malley, J. D. (2011). Brief review of regression-based and machine learning methods in genetic epidemiology: the Genetic Analysis Workshop 17 experience. Genetic Epidemiology, 35(S1), S5–S11. https://doi.org/10.1002/gepi.20642

Dasgupta, Avijit, & Singh, S. (2017). A fully convolutional neural network based structured prediction approach towards the retinal vessel segmentation. 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017). https://doi.org/10.1109/isbi.2017.7950512

Databricks Inc. (2019). Apache Spark. Retrieved from Databricks website: https://databricks.com/spark/about

De Fauw, J., Ledsam, J. R., Romera-Paredes, B., Nikolov, S., Tomasev, N., Blackwell, S., … Ronneberger, O. (2018). Clinically applicable deep learning for diagnosis and referral in retinal disease. Nature Medicine, 24(9), 1342–1350. https://doi.org/10.1038/s41591-018-0107-6

Deloitte. (2019). Predictive Analytics in Healthcare. In E. Leo, S. Allen, & R. Chat (Eds.), Deloitte. Retrieved from Deloitte website: https://www2.deloitte.com/content/dam/Deloitte/ec/Documents/life-sciences-health-care/DI_Predictive-analytics-in-health-care%20(2).pdf

Erickson, B. J., Korfiatis, P., Akkus, Z., & Kline, T. L. (2017). Machine Learning for Medical Imaging. RadioGraphics, 37(2), 505–515. https://doi.org/10.1148/rg.2017160130

Facebook Open Source. (2019). React – A JavaScript library for building user interfaces. Retrieved from Reactjs.org website: https://reactjs.org/

Frazier, P., rossi mori, Angelo, Dolin, R., Alschuler, L., & Huff, S. (2001a). The creation of an ontology of clinical document names. Studies in Health Technology and Informatics, 84, 94–98.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics, 29(5), 1189–1232. Retrieved from http://www.jstor.org/stable/2699986

GitHub. (2018). Retrieved from GitHub website: https://github.com/

Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

Goldstein, B. A., Navar, A. M., Pencina, M. J., & Ioannidis, J. P. A. (2016). Opportunities and challenges in developing risk prediction models with electronic health records data: a systematic review. Journal of the American Medical Informatics Association, 24(1), 198–208. https://doi.org/10.1093/jamia/ocw042

Grinberg, M. (2018). Flask Web Development. (2nd ed.). O'reilly Media, Incorporated.

Han, J., & Kamber, M. (2012). Data Mining: Concepts and Techniques (3rd ed.). Haryana, India ; Burlington, Ma: Elsevier.

Hand, D. J. (2007). Principles of Data Mining. Drug Safety, 30(7), 621–622. https://doi.org/10.2165/00002018-200730070-00010

Harispe, S., Sánchez, D., Ranwez, S., Janaqi, S., & Montmain, J. (2014a). A framework for unifying ontology-based semantic similarity measures: A study in the biomedical domain. Journal of Biomedical Informatics, 48, 38–53. https://doi.org/10.1016/j.jbi.2013.11.006

Harutyunyan, H., Kale, D., Khachatrian, H., Natny, Badger, The Gitter, Saqib, & Romanov, A. (2018). YerevaNN/mimic3-benchmarks: MIMIC3-Benchmarks v1.0.0-alpha. https://doi.org/10.5281/ZENODO.1306527

Harutyunyan, H., Khachatrian, H., Kale, D. C., Steeg, G. V., & Galstyan, A. (2019). Multitask learning and benchmarking with clinical time series data. Scientific Data, 6(1), 96. https://doi.org/10.1038/s41597-019-0103-9

Hootsuite, & We Are Social. (2020). Digital 2020 : Global Digital Overview. In DataReportal. Retrieved from https://datareportal.com/reports/digital-2020-global-digital-overview

ICD-9-CM Diagnosis Codes 754.* : Certain congenital musculoskeletal deformities. (2015). Retrieved September 21, 2020, from www.icd9data.com website: http://www.icd9data.com/2015/Volume1/740-759/754/default.htm

Jeni, L. A., Cohn, J. F., & De La Torre, F. (2013). Facing Imbalanced Data--Recommendations for the Use of Performance Metrics. 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction. https://doi.org/10.1109/acii.2013.47

JetBrains. (2019). PyCharm. Retrieved from JetBrains website: https://www.jetbrains.com/pycharm/

Jie Cai, Jiawei Luo, Shulin Wang, & Yang, S. (2018). Feature selection in machine learning: A new perspective. Neurocomputing, 300, 70–79. https://doi.org/https://doi.org/10.1016/j.neucom.2017.11.077

Johnson, A. E. W., Stone, D. J., Celi, L. A., & Pollard, T. J. (2017). The MIMIC Code Repository: enabling reproducibility in critical care research. Journal of the American Medical Informatics Association, 25(1), 32–39. https://doi.org/10.1093/jamia/ocx084

Johnson, A., Pollard, T., Shen, L., Lehman, L.-W., Feng, M., Ghassemi, M., … Mark, R. (2016). MIMIC-III, a freely accessible critical care database. Scientific Data, 3(160035 (2016)). https://doi.org/10.1038/sdata.2016.35

Jothi, N., Rashid, N. A., & Husain, W. (2015). Data Mining in Healthcare – A Review. Procedia Computer Science, 72, 306–313. https://doi.org/10.1016/j.procs.2015.12.145

Khair, U., Fahmi, H., Hakim, S. A., & Rahim, R. (2017). Forecasting Error Calculation with Mean Absolute Deviation and Mean Absolute Percentage Error. Journal of Physics: Conference Series, 930, 012002. https://doi.org/10.1088/1742-6596/930/1/012002

Khairat, S., Coleman, C., Ottmar, P., Jayachander, D. I., Bice, T., & Carson, S. S. (2020). Association of Electronic Health Record Use With Physician Fatigue and Efficiency. JAMA Network Open, 3(6), e207385. https://doi.org/10.1001/jamanetworkopen.2020.7385

Khan, M. U., Choi, J. P., Shin, H., & Kim, M. (2008). Predicting breast cancer survivability using fuzzy decision trees for personalized healthcare. 2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. https://doi.org/10.1109/iembs.2008.4650373

Kirlidog, M., & Asuk, C. (2012). A Fraud Detection Approach with Data Mining in Health Insurance. Procedia - Social and Behavioral Sciences, 62, 989–994. https://doi.org/10.1016/j.sbspro.2012.09.168

Lapalu, J., Bouchard, K., Bouzouane, A., Bouchard, B., & Giroux, S. (2013). Unsupervised Mining of Activities for Smart Home Prediction. Procedia Computer Science, 19, 503–510. https://doi.org/10.1016/j.procs.2013.06.067

Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D. D., & Chen, M. (2014). Medical image classification with convolutional neural network. 2014 13th International Conference on Control Automation Robotics & Vision (ICARCV). https://doi.org/10.1109/icarcv.2014.7064414

Liu, B., Xiao, Y., Cao, L., Hao, Z., & Deng, F. (2012). SVDD-based outlier detection on uncertain data. Knowledge and Information Systems, 34(3), 597–618. https://doi.org/10.1007/s10115-012-0484-y

McDonald, C. J., Huff, S. M., Suico, J. G., Hill, G., Leavelle, D., Aller, R., … Maloney, P. (2003a). LOINC, a Universal Standard for Identifying Laboratory Observations: A 5-Year Update. Clinical Chemistry, 49(4), 624–633. https://doi.org/10.1373/49.4.624

McHugh, M. L. (2012). Interrater reliability: the kappa statistic. Biochemia Medica, 22(3), 276–282. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/

Mehta, N., & Pandit, A. (2018). Concurrence of big data analytics and healthcare: A systematic review. International Journal of Medical Informatics, 114, 57–65. https://doi.org/10.1016/j.ijmedinf.2018.03.013

Muchlinski, D., Siroky, D., He, J., & Kocher, M. (2016). Comparing Random Forest with Logistic Regression for Predicting Class-Imbalanced Civil War Onset Data. Political Analysis, 24(1), 87–103. https://doi.org/10.1093/pan/mpv024

Mukhopadhyay, A., Tai, B. C., See, K. C., Ng, W. Y., Lim, T. K., Onsiong, S., … Phua, J. (2014). Risk Factors for Hospital and Long-Term Mortality of Critically Ill Elderly Patients Admitted to an Intensive Care Unit. BioMed Research International, 2014, 1–10. https://doi.org/10.1155/2014/960575

Nonerocandela, J., & Roweis, S. (2003). Data Imputation and Robust Training with Gaussian Processes.

P, W. D., & A, D. E. (1984). Acute physiology and chronic health evaluation (APACHE II) and Medicare reimbursement. Health Care Financing Review, Suppl(Suppl), 91–105. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4195105/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

Purushotham, S., Meng, C., Che, Z., & Liu, Y. (2018). Benchmarking deep learning models on large healthcare datasets. Journal of Biomedical Informatics, 83, 112–134. https://doi.org/10.1016/j.jbi.2018.04.007

Putra, W. D. K. (2018). Supervised Learning. Retrieved from GitHub website: https://github.com/WiraDKP/supervised_learning

Rajkomar, A., Dean, J., & Kohane, I. (2019). Machine Learning in Medicine. New England Journal of Medicine, 380(14), 1347–1358. https://doi.org/10.1056/nejmra1814259

Reddy, C. K., & Aggarwal, C. C. (2015). Healthcare Data Analytics (p. 22). Boca Raton: CRC Press.

Ribeiro de Paula, D., Ziegler, E., Abeyasinghe, P. M., Das, T. K., Cavaliere, C., Aiello, M., … Soddu, A. (2017). A method for independent component graph analysis of resting-state fMRI. Brain and Behavior, 7(3), e00626. https://doi.org/10.1002/brb3.626

Rocheteau, E., Liò, P., & Hyland, S. (2020). Temporal Pointwise Convolutional Networks for Length of Stay Prediction in the Intensive Care Unit.

Saabith, A. L. S., Fareez, MMM., & Vinothraj, T. (2019). PYTHON CURRENT TREND APPLICATIONS- AN OVERVIEW. International Journal of Advance Engineering and Research Development, 6(10).

Schrider, D. R., & Kern, A. D. (2018). Supervised Machine Learning for Population Genetics: A New Paradigm. Trends in Genetics, 34(4), 301–312. https://doi.org/10.1016/j.tig.2017.12.005

Schuh, G., Reinhart, G., Prote, J.-P., Sauermann, F., Horsthofer, J., Oppolzer, F., & Knoll, D. (2019). Data Mining Definitions and Applications for the Management of Production Complexity. Procedia CIRP, 81, 874–879. https://doi.org/10.1016/j.procir.2019.03.217

Shahzad, F. (2017). Modern and Responsive Mobile-enabled Web Applications. Procedia Computer Science, 110, 410–415. https://doi.org/10.1016/j.procs.2017.06.105

Shammamah Hossain. (2019a). Visualization of Bioinformatics Data with Dash Bio. In C. Calloway, D. Lippa, D. Niederhut, & D. Shupe (Eds.), Proceedings of the 18th Python in Science Conference (SciPy 2019) (pp. 126–133). https://doi.org/10.25080/Majora-7ddc1dd1-012

Shammamah Hossain. (2019b). Visualization of Bioinformatics Data with Dash Bio. In C. Calloway, D. Lippa, D. Niederhut, & D. Shupe (Eds.), Proceedings of the 18th Python in Science Conference (SciPy 2019) (pp. 126–133). https://doi.org/10.25080/Majora-7ddc1dd1-012

Shan, T., & Hua, W. (2006). Taxonomy of Java Web Application Frameworks. 2006 IEEE International Conference on E-Business Engineering (ICEBE'06). https://doi.org/10.1109/icebe.2006.98

Silva, I., Moody, G., Scott, D. J., Celi, L. A., & Mark, R. G. (2012a). Predicting InHospital Mortality of ICU Patients: The PhysioNet/Computing in Cardiology Challenge 2012. Computing in Cardiology, 39, 245–248. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3965265/

Silva, I., Moody, G., Scott, D. J., Celi, L. A., & Mark, R. G. (2012b). Predicting InHospital Mortality of ICU Patients: The PhysioNet/Computing in Cardiology Challenge 2012. Computing in Cardiology, 39, 245–248. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3965265/

Snowflake. (2018). The Data Warehouse Built for the Cloud | Snowflake. Retrieved from Snowflake website: https://www.snowflake.com/

Spackman, K.A., Campbell, K. E., & Côté, R. A. (1997a). SNOMED RT: a reference terminology for health care. Proceedings : A Conference of the American Medical Informatics Association. AMIA Fall Symposium, 640–644.

Spackman, K.A., Campbell, K. E., & Côté, R. A. (1997b). SNOMED RT: a reference terminology for health care. Proceedings : A Conference of the American Medical Informatics Association. AMIA Fall Symposium, 640–644.

Spackman, Kent A., Dionne, R., Mays, E., & Weis, J. (2002a). Role grouping as an extension to the description logic of Ontylog, motivated by concept modeling in SNOMED. Proceedings. AMIA Symposium, 712–716.

Spackman, Kent A., Dionne, R., Mays, E., & Weis, J. (2002b). Role grouping as an extension to the description logic of Ontylog, motivated by concept modeling in SNOMED. Proceedings. AMIA Symposium, 712–716.

Stack Overflow 2020 Developer Survey. (2019). Programming, Scripting, and Markup Languages. Retrieved from Stack Overflow website: https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-all-respondents

Thompson, P. A. (1990). An MSE statistic for comparing forecast accuracy across series. International Journal of Forecasting, 6(2), 219–227. https://doi.org/10.1016/0169-2070(90)90007-X

Topaz, M., Shafran-Topaz, L., & Bowles, K. H. (2013). ICD-9 to ICD-10: Evolution, revolution, and current debates in the united states. Perspectives in Health Information Management, 10(Spring), 1d.

Ukil, A., Bandyoapdhyay, S., Puri, C., & Pal, A. (2016). IoT Healthcare Analytics: The Importance of Anomaly Detection. 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). https://doi.org/10.1109/aina.2016.158

Veloso, R., Portela, F., Santos, M. F., Silva, Á., Rua, F., Abelha, A., & Machado, J. (2014). A Clustering Approach for Predicting Readmissions in Intensive Medicine. Procedia Technology, 16, 1307–1316. https://doi.org/10.1016/j.protcy.2014.10.147

Vreeman, D. J., & McDonald, C. J. (2005a). Automated mapping of local radiology terms to LOINC. AMIA ... Annual Symposium Proceedings. AMIA Symposium, 2005, 769–773. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1560555/

Vreeman, D. J., & McDonald, C. J. (2005b). Automated mapping of local radiology terms to LOINC. AMIA ... Annual Symposium Proceedings. AMIA Symposium, 2005, 769–773. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1560555/

Vreeman, D. J., McDonald, C. J., & Huff, S. M. (2010a). LOINC®: a universal catalogue of individual clinical observations and uniform representation of enumerated collections. International Journal of Functional Informatics and Personalised Medicine, 3(4), 273. https://doi.org/10.1504/ijfipm.2010.040211

Vreeman, D. J., McDonald, C. J., & Huff, S. M. (2010b). LOINC®: a universal catalogue of individual clinical observations and uniform representation of enumerated collections. International Journal of Functional Informatics and Personalised Medicine, 3(4), 273. https://doi.org/10.1504/ijfipm.2010.040211

Wei, W.-Q., Bastarache, L. A., Carroll, R. J., Marlo, J. E., Osterman, T. J., Gamazon, E. R., … Denny, J. C. (2017). Evaluating phecodes, clinical classification software, and ICD-9-CM codes for phenome-wide association studies in the electronic health record. PLOS ONE, 12(7), e0175508. https://doi.org/10.1371/journal.pone.0175508

Wong, A., Young, A. T., Liang, A. S., Gonzales, R., Douglas, V. C., & Hadley, D. (2018). Development and Validation of an Electronic Health Record–Based Machine Learning Model to Estimate Delirium Risk in Newly Hospitalized Patients Without Known Cognitive Impairment. JAMA Network Open, 1(4), e181018. https://doi.org/10.1001/jamanetworkopen.2018.1018

XGBoost. (2019, October 25). XGBoost eXtreme Gradient Boosting. Retrieved from GitHub website: https://github.com/dmlc/xgboost

Xie, J., Douglas, P. K., Wu, Y. N., Brody, A. L., & Anderson, A. E. (2017). Decoding the encoding of functional brain networks: An fMRI classification comparison of non-negative matrix factorization (NMF), independent component analysis (ICA), and sparse coding algorithms. Journal of Neuroscience Methods, 282, 81–94. https://doi.org/10.1016/j.jneumeth.2017.03.008

Yoo, I., Alafaireet, P., Marinov, M., Pena-Hernandez, K., Gopidi, R., Chang, J.-F., & Hua, L. (2011). Data Mining in Healthcare and Biomedicine: A Survey of the Literature. Journal of Medical Systems, 36(4), 2431–2448. https://doi.org/10.1007/s10916-011-9710-5

Appendices

Protocols to run the data preprocessing and preparation workflow

Below are the procedures to run the script shown on Figure 6. The current protocol is adapted from the data preprocessing pipeline conducted by Harutyunyan et al. (2019). The directory for which the script is run might differ between each local computer or hardware and thus must be adjusted accordingly. Certain scripts might require different modules or packages scattered through the directories, thus it is advisable to run the application on the same directory to prevent errors. Furthermore, the required python package is shown on the top of each script and imported accordingly.

1. Iterates through all the MIMIC-III csv dataset files and extracts redundant data points and removes the pediatric patient category. A directory containing each ICU stays will be generated based on each SUBJECT_ID. The output file is a CSV file of 'stays.csv', 'diagnoses.csv', and 'events.csv'. [**extract_subjects.py**]
2. Fix issues with missing ICUSTAYS_ID stays and HADM_ID [**validate_events.py**]
3. Breaks up per-subject data into separate episodes (pertaining to ICU stays) [**extract_episodes_from_subjects.py**]
4. Splits the whole dataset into training and testing sets [**split_train_and_test.py**]
5. Generate task-specific datasets, which can later be used in models [**create_in_hospital_mortality.py** and **create_length_of_stay.py**]
6. Extract validation set from the training set [**split_train_val.py**]

➔ **extract_subjects.py**

```
from __future__ import absolute_import
from __future__ import print_function

import argparse
import yaml
import os

from mimic3benchmark.mimic3csv import *
from mimic3benchmark.preprocessing import
add_hcup_ccs_2015_groups, make_phenotype_label_matrix
from mimic3benchmark.util import *

parser = argparse.ArgumentParser(description='Extract per-
subject data from MIMIC-III CSV files.')
parser.add_argument('mimic3_path', type=str,
help='Directory containing MIMIC-III CSV files.')
parser.add_argument('output_path', type=str,
help='Directory where per-subject data should be written.')
parser.add_argument('--event_tables', '-e', type=str,
nargs='+', help='Tables from which to read events.',
```

```
                    default=['CHARTEVENTS', 'LABEVENTS',
'OUTPUTEVENTS'])
parser.add_argument('--phenotype_definitions', '-p',
type=str,

default=os.path.join(os.path.dirname(__file__),
'../resources/hcup_ccs_2015_definitions.yaml'),
                    help='YAML file with phenotype
definitions.')
parser.add_argument('--itemids_file', '-i', type=str,
help='CSV containing list of ITEMIDs to keep.')
parser.add_argument('--verbose', '-v', type=int,
help='Level of verbosity in output.', default=1)
parser.add_argument('--test', action='store_true',
help='TEST MODE: process only 1000 subjects, 1000000
events.')
args, _ = parser.parse_known_args()

try:
    os.makedirs(args.output_path)
except:
    pass

patients = read_patients_table(args.mimic3_path)
admits = read_admissions_table(args.mimic3_path)
stays = read_icustays_table(args.mimic3_path)
if args.verbose:
    print('START:', stays.icustay_id.unique().shape[0],
stays.hadm_id.unique().shape[0],
          stays.subject_id.unique().shape[0])

stays = remove_icustays_with_transfers(stays)
if args.verbose:
    print('REMOVE ICU TRANSFERS:',
stays.icustay_id.unique().shape[0],
stays.hadm_id.unique().shape[0],
          stays.subject_id.unique().shape[0])

stays = merge_on_subject_admission(stays, admits)
stays = merge_on_subject(stays, patients)
stays = filter_admissions_on_nb_icustays(stays)
if args.verbose:
    print('REMOVE MULTIPLE STAYS PER ADMIT:',
stays.icustay_id.unique().shape[0],
stays.hadm_id.unique().shape[0],
          stays.subject_id.unique().shape[0])

stays = add_age_to_icustays(stays)
stays = add_inunit_mortality_to_icustays(stays)
stays = add_inhospital_mortality_to_icustays(stays)
stays = filter_icustays_on_age(stays)
```

```
    if args.verbose:
        print('REMOVE PATIENTS AGE < 18:',
    stays.icustay_id.unique().shape[0],
    stays.hadm_id.unique().shape[0],
              stays.subject_id.unique().shape[0])

    stays.to_csv(os.path.join(args.output_path,
    'all_stays.csv'), index=False)
    diagnoses = read_icd_diagnoses_table(args.mimic3_path)
    diagnoses = filter_diagnoses_on_stays(diagnoses, stays)
    diagnoses.to_csv(os.path.join(args.output_path,
    'all_diagnoses.csv'), index=False)
    count_icd_codes(diagnoses,
    output_path=os.path.join(args.output_path,
    'diagnosis_counts.csv'))

    phenotypes = add_hcup_ccs_2015_groups(diagnoses,
    yaml.load(open(args.phenotype_definitions, 'r')))
    make_phenotype_label_matrix(phenotypes,
    stays).to_csv(os.path.join(args.output_path,
    'phenotype_labels.csv'),

    index=False, quoting=csv.QUOTE_NONNUMERIC)

    if args.test:
        pat_idx = np.random.choice(patients.shape[0],
    size=1000)
        patients = patients.iloc[pat_idx]
        stays = stays.merge(patients[['subject_id']],
    left_on='subject_id', right_on='subject_id')
        args.event_tables = [args.event_tables[0]]
        print('Using only', stays.shape[0], 'stays and only',
    args.event_tables[0], 'table')

    subjects = stays.subject_id.unique()
    break_up_stays_by_subject(stays, args.output_path,
    subjects=subjects, verbose=args.verbose)
    break_up_diagnoses_by_subject(phenotypes, args.output_path,
    subjects=subjects, verbose=args.verbose)
    items_to_keep = set(
        [int(itemid) for itemid in
    dataframe_from_csv(args.itemids_file)['itemid'].unique()])
    if args.itemids_file else None
    for table in args.event_tables:

    read_events_table_and_break_up_by_subject(args.mimic3_path,
    table, args.output_path, items_to_keep=items_to_keep,

    subjects_to_keep=subjects, verbose=args.verbose)
```

➔ **validate_events.py**

```
from __future__ import absolute_import
from __future__ import print_function

import os
import argparse
import pandas as pd


def is_subject_folder(x):
    return str.isdigit(x)


def main():

    n_events = 0                    # total number of events
    empty_hadm = 0                  # HADM_ID is empty in
events.csv. We exclude such events.
    no_hadm_in_stay = 0             # HADM_ID does not
appear in stays.csv. We exclude such events.
    no_icustay = 0                  # ICUSTAY_ID is empty in
events.csv. We try to fix such events.
    recovered = 0                   # empty ICUSTAY_IDs are
recovered according to stays.csv files (given HADM_ID)
    could_not_recover = 0           # empty ICUSTAY_IDs that
are not recovered. This should be zero.
    icustay_missing_in_stays = 0    # ICUSTAY_ID does not
appear in stays.csv. We exclude such events.

    parser = argparse.ArgumentParser()
    parser.add_argument('subjects_root_path', type=str,
                        help='Directory containing subject
subdirectories.')
    args = parser.parse_args()
    print(args)

    subdirectories = os.listdir(args.subjects_root_path)
    subjects = list(filter(is_subject_folder,
subdirectories))

    for (index, subject) in enumerate(subjects):
        if index % 100 == 0:
            print("processed {} / {} {}\r".format(index+1,
len(subjects), ' '*10))

        stays_df =
pd.read_csv(os.path.join(args.subjects_root_path, subject,
'stays.csv'), index_col=False,
                                dtype={'HADM_ID': str,
"ICUSTAY_ID": str})
        stays_df.columns = stays_df.columns.str.upper()
```

```
        # assert that there is no row with empty ICUSTAY_ID
or HADM_ID
        assert(not stays_df['ICUSTAY_ID'].isnull().any())
        assert(not stays_df['HADM_ID'].isnull().any())

        # assert there are no repetitions of ICUSTAY_ID or
HADM_ID
        # since admissions with multiple ICU stays were
excluded
        assert(len(stays_df['ICUSTAY_ID'].unique()) ==
len(stays_df['ICUSTAY_ID']))
        assert(len(stays_df['HADM_ID'].unique()) ==
len(stays_df['HADM_ID']))

        events_df =
pd.read_csv(os.path.join(args.subjects_root_path, subject,
'events.csv'), index_col=False,
                                    dtype={'HADM_ID': str,
"ICUSTAY_ID": str})
        events_df.columns = events_df.columns.str.upper()
        n_events += events_df.shape[0]

        # we drop all events for them HADM_ID is empty
        # TODO: maybe we can recover HADM_ID by looking at
ICUSTAY_ID
        empty_hadm += events_df['HADM_ID'].isnull().sum()
        events_df = events_df.dropna(subset=['HADM_ID'])

        merged_df = events_df.merge(stays_df,
left_on=['HADM_ID'], right_on=['HADM_ID'],
                                    how='left',
suffixes=['', '_r'], indicator=True)

        # we drop all events for which HADM_ID is not
listed in stays.csv
        # since there is no way to know the targets of that
stay (for example mortality)
        no_hadm_in_stay += (merged_df['_merge'] ==
'left_only').sum()
        merged_df = merged_df[merged_df['_merge'] ==
'both']

        # if ICUSTAY_ID is empty in stays.csv, we try to
recover it
        # we exclude all events for which we could not
recover ICUSTAY_ID
        cur_no_icustay =
merged_df['ICUSTAY_ID'].isnull().sum()
        no_icustay += cur_no_icustay
        merged_df.loc[:, 'ICUSTAY_ID'] =
merged_df['ICUSTAY_ID'].fillna(merged_df['ICUSTAY_ID_r'])
```

```
        recovered += cur_no_icustay -
merged_df['ICUSTAY_ID'].isnull().sum()
        could_not_recover +=
merged_df['ICUSTAY_ID'].isnull().sum()
        merged_df = merged_df.dropna(subset=['ICUSTAY_ID'])

        # now we take a look at the case when ICUSTAY_ID is
present in events.csv, but not in stays.csv
        # this mean that ICUSTAY_ID in events.csv is not
the same as that of stays.csv for the same HADM_ID
        # we drop all such events
        icustay_missing_in_stays +=
(merged_df['ICUSTAY_ID'] !=
merged_df['ICUSTAY_ID_r']).sum()
        merged_df = merged_df[(merged_df['ICUSTAY_ID'] ==
merged_df['ICUSTAY_ID_r'])]

        to_write = merged_df[['SUBJECT_ID', 'HADM_ID',
'ICUSTAY_ID', 'CHARTTIME', 'ITEMID', 'VALUE', 'VALUEUOM']]

    to_write.to_csv(os.path.join(args.subjects_root_path,
    subject, 'events.csv'), index=False)

        assert(could_not_recover == 0)
        print('n_events: {}'.format(n_events))
        print('empty_hadm: {}'.format(empty_hadm))
        print('no_hadm_in_stay: {}'.format(no_hadm_in_stay))
        print('no_icustay: {}'.format(no_icustay))
        print('recovered: {}'.format(recovered))
        print('could_not_recover:
    {}'.format(could_not_recover))
        print('icustay_missing_in_stays:
    {}'.format(icustay_missing_in_stays))


    if __name__ == "__main__":
        main()
```

➔ **extract_episodes_from_subjects.py**

```
    from __future__ import absolute_import
    from __future__ import print_function

    import argparse

    import os
    import sys

    from mimic3benchmark.subject import read_stays,
    read_diagnoses, read_events, get_events_for_stay,
    add_hours_elpased_to_events
```

```
from mimic3benchmark.subject import
convert_events_to_timeseries,
get_first_valid_from_timeseries
from mimic3benchmark.preprocessing import
read_itemid_to_variable_map, map_itemids_to_variables,
read_variable_ranges, clean_events
from mimic3benchmark.preprocessing import transform_gender,
transform_ethnicity, assemble_episodic_data


parser = argparse.ArgumentParser(description='Extract
episodes from per-subject data.')
parser.add_argument('subjects_root_path', type=str,
help='Directory containing subject sub-directories.')
parser.add_argument('--variable_map_file', type=str,

default=os.path.join(os.path.dirname(__file__),
'../resources/itemid_to_variable_map.csv'),
                    help='CSV containing ITEMID-to-VARIABLE
map.')
parser.add_argument('--reference_range_file', type=str,

default=os.path.join(os.path.dirname(__file__),
'../resources/variable_ranges.csv'),
                    help='CSV containing reference ranges
for VARIABLEs.')
parser.add_argument('--verbose', '-v', type=int,
help='Level of verbosity in output.', default=1)
args, _ = parser.parse_known_args()

var_map =
read_itemid_to_variable_map(args.variable_map_file)
variables = var_map.VARIABLE.unique()

for subject_dir in os.listdir(args.subjects_root_path):
    dn = os.path.join(args.subjects_root_path, subject_dir)
    try:
        subject_id = int(subject_dir)
        if not os.path.isdir(dn):
            raise Exception
    except:
        continue
    sys.stdout.write('Subject {}: '.format(subject_id))
    sys.stdout.flush()

    try:
        sys.stdout.write('reading...')
        sys.stdout.flush()
        stays =
read_stays(os.path.join(args.subjects_root_path,
subject_dir))
```

```
        diagnoses =
read_diagnoses(os.path.join(args.subjects_root_path,
subject_dir))
        events =
read_events(os.path.join(args.subjects_root_path,
subject_dir))
    except:
        sys.stdout.write('error reading from disk!\n')
        continue
    else:
        sys.stdout.write('got {0} stays, {1} diagnoses, {2}
events...'.format(stays.shape[0], diagnoses.shape[0],
events.shape[0]))
        sys.stdout.flush()

    episodic_data = assemble_episodic_data(stays,
diagnoses)

    sys.stdout.write('cleaning and converting to time
series...')
    sys.stdout.flush()
    events = map_itemids_to_variables(events, var_map)
    events = clean_events(events)
    if events.shape[0] == 0:
        sys.stdout.write('no valid events!\n')
        continue
    timeseries = convert_events_to_timeseries(events,
variables=variables)

    sys.stdout.write('extracting separate episodes...')
    sys.stdout.flush()

    for i in range(stays.shape[0]):
        stay_id = stays.ICUSTAY_ID.iloc[i]
        sys.stdout.write(' {}'.format(stay_id))
        sys.stdout.flush()
        intime = stays.INTIME.iloc[i]
        outtime = stays.OUTTIME.iloc[i]

        episode = get_events_for_stay(timeseries, stay_id,
intime, outtime)
        if episode.shape[0] == 0:
            sys.stdout.write(' (no data!)')
            sys.stdout.flush()
            continue

        episode = add_hours_elpased_to_events(episode,
intime).set_index('HOURS').sort_index(axis=0)
        episodic_data.Weight.ix[stay_id] =
get_first_valid_from_timeseries(episode, 'Weight')
```

```
        episodic_data.Height.ix[stay_id] =
get_first_valid_from_timeseries(episode, 'Height')

episodic_data.ix[episodic_data.index==stay_id].to_csv(os.pa
th.join(args.subjects_root_path, subject_dir,
'episode{}.csv'.format(i+1)), index_label='Icustay')
        columns = list(episode.columns)
        columns_sorted = sorted(columns, key=(lambda x: ""
if x == "Hours" else x))
        episode = episode[columns_sorted]

episode.to_csv(os.path.join(args.subjects_root_path,
subject_dir, 'episode{}_timeseries.csv'.format(i+1)),
index_label='Hours')
    sys.stdout.write(' DONE!\n')
```

➔ **split_train_and_test.py**

```
from __future__ import absolute_import
from __future__ import print_function

import os
import shutil
import argparse


def move_to_partition(args, patients, partition):
    if not
os.path.exists(os.path.join(args.subjects_root_path,
partition)):
        os.mkdir(os.path.join(args.subjects_root_path,
partition))
    for patient in patients:
        src = os.path.join(args.subjects_root_path,
patient)
        dest = os.path.join(args.subjects_root_path,
partition, patient)
        shutil.move(src, dest)


def main():
    parser = argparse.ArgumentParser(description='Split
data into train and test sets.')
    parser.add_argument('subjects_root_path', type=str,
help='Directory containing subject sub-directories.')
    args, _ = parser.parse_known_args()

    test_set = set()
    with open(os.path.join(os.path.dirname(__file__),
'../resources/testset.csv'), "r") as test_set_file:
        for line in test_set_file:
```

```
            x, y = line.split(',')
            if int(y) == 1:
                test_set.add(x)

    folders = os.listdir(args.subjects_root_path)
    folders = list((filter(str.isdigit, folders)))
    train_patients = [x for x in folders if x not in
test_set]
    test_patients = [x for x in folders if x in test_set]

    assert len(set(train_patients) & set(test_patients)) ==
0

    move_to_partition(args, train_patients, "train")
    move_to_partition(args, test_patients, "test")


if __name__ == '__main__':
    main()
```

➔ **create_in_hospital_mortality.py**

```
from __future__ import absolute_import
from __future__ import print_function

import os
import argparse
import pandas as pd
import random
random.seed(49297)


def process_partition(args, partition, eps=1e-6,
n_hours=48):
    output_dir = os.path.join(args.output_path, partition)
    if not os.path.exists(output_dir):
        os.mkdir(output_dir)

    xy_pairs = []
    patients = list(filter(str.isdigit,
os.listdir(os.path.join(args.root_path, partition))))
    for (patient_index, patient) in enumerate(patients):
        patient_folder = os.path.join(args.root_path,
partition, patient)
        patient_ts_files = list(filter(lambda x:
x.find("timeseries") != -1, os.listdir(patient_folder)))

        for ts_filename in patient_ts_files:
            with open(os.path.join(patient_folder,
ts_filename)) as tsfile:
```

```
                lb_filename =
ts_filename.replace("_timeseries", "")
                label_df =
pd.read_csv(os.path.join(patient_folder, lb_filename))

                # empty label file
                if label_df.shape[0] == 0:
                    continue

                mortality =
int(label_df.iloc[0]["Mortality"])
                los = 24.0 * label_df.iloc[0]['Length of
Stay']  # in hours
                if pd.isnull(los):
                    print("\n\t(length of stay is
missing)", patient, ts_filename)
                    continue

                if los < n_hours - eps:
                    continue

                ts_lines = tsfile.readlines()
                header = ts_lines[0]
                ts_lines = ts_lines[1:]
                event_times = [float(line.split(',')[0])
for line in ts_lines]

                ts_lines = [line for (line, t) in
zip(ts_lines, event_times)
                            if -eps < t < n_hours + eps]

                # no measurements in ICU
                if len(ts_lines) == 0:
                    print("\n\t(no events in ICU) ",
patient, ts_filename)
                    continue

                output_ts_filename = patient + "_" +
ts_filename
                with open(os.path.join(output_dir,
output_ts_filename), "w") as outfile:
                    outfile.write(header)
                    for line in ts_lines:
                        outfile.write(line)

                xy_pairs.append((output_ts_filename,
mortality))

        if (patient_index + 1) % 100 == 0:
```

```
            print("processed {} / {}
patients".format(patient_index + 1, len(patients)),
end='\r')

    print("\n", len(xy_pairs))
    if partition == "train":
        random.shuffle(xy_pairs)
    if partition == "test":
        xy_pairs = sorted(xy_pairs)

    with open(os.path.join(output_dir, "listfile.csv"),
"w") as listfile:
        listfile.write('stay,y_true\n')
        for (x, y) in xy_pairs:
            listfile.write('{},{:d}\n'.format(x, y))


def main():
    parser = argparse.ArgumentParser(description="Create
data for in-hospital mortality prediction task.")
    parser.add_argument('root_path', type=str, help="Path
to root folder containing train and test sets.")
    parser.add_argument('output_path', type=str,
help="Directory where the created data should be stored.")
    args, _ = parser.parse_known_args()

    if not os.path.exists(args.output_path):
        os.makedirs(args.output_path)

    process_partition(args, "test")
    process_partition(args, "train")


if __name__ == '__main__':
    main()
```

➔ **create_length_of_stay.py**

```
from __future__ import absolute_import
from __future__ import print_function

import os
import argparse
import numpy as np
import pandas as pd
import random
random.seed(49297)


def process_partition(args, partition, sample_rate=1.0,
shortest_length=4.0, eps=1e-6):
    output_dir = os.path.join(args.output_path, partition)
```

```
    if not os.path.exists(output_dir):
        os.mkdir(output_dir)

    xty_triples = []
    patients = list(filter(str.isdigit,
os.listdir(os.path.join(args.root_path, partition))))
    for (patient_index, patient) in enumerate(patients):
        patient_folder = os.path.join(args.root_path,
partition, patient)
        patient_ts_files = list(filter(lambda x:
x.find("timeseries") != -1, os.listdir(patient_folder)))

        for ts_filename in patient_ts_files:
            with open(os.path.join(patient_folder,
ts_filename)) as tsfile:
                lb_filename =
ts_filename.replace("_timeseries", "")
                label_df =
pd.read_csv(os.path.join(patient_folder, lb_filename))

                # empty label file
                if label_df.shape[0] == 0:
                    print("\n\t(empty label file)",
patient, ts_filename)
                    continue

                los = 24.0 * label_df.iloc[0]['Length of
Stay']  # in hours
                if pd.isnull(los):
                    print("\n\t(length of stay is
missing)", patient, ts_filename)
                    continue

                ts_lines = tsfile.readlines()
                header = ts_lines[0]
                ts_lines = ts_lines[1:]
                event_times = [float(line.split(',')[0])
for line in ts_lines]

                ts_lines = [line for (line, t) in
zip(ts_lines, event_times)
                            if -eps < t < los + eps]
                event_times = [t for t in event_times
                               if -eps < t < los + eps]

                # no measurements in ICU
                if len(ts_lines) == 0:
                    print("\n\t(no events in ICU) ",
patient, ts_filename)
                    continue
```

```
                sample_times = np.arange(0.0, los + eps,
sample_rate)

                sample_times = list(filter(lambda x: x >
shortest_length, sample_times))

                # At least one measurement
                sample_times = list(filter(lambda x: x >
event_times[0], sample_times))

                output_ts_filename = patient + "_" +
ts_filename
                with open(os.path.join(output_dir,
output_ts_filename), "w") as outfile:
                    outfile.write(header)
                    for line in ts_lines:
                        outfile.write(line)

                for t in sample_times:
                    xty_triples.append((output_ts_filename,
t, los - t))

        if (patient_index + 1) % 100 == 0:
            print("processed {} / {}
patients".format(patient_index + 1, len(patients)),
end='\r')

    print(len(xty_triples))
    if partition == "train":
        random.shuffle(xty_triples)
    if partition == "test":
        xty_triples = sorted(xty_triples)

    with open(os.path.join(output_dir, "listfile.csv"),
"w") as listfile:
        listfile.write('stay,period_length,y_true\n')
        for (x, t, y) in xty_triples:
            listfile.write('{},{:.6f},{:.6f}\n'.format(x,
t, y))


def main():
    parser = argparse.ArgumentParser(description="Create
data for length of stay prediction task.")
    parser.add_argument('root_path', type=str, help="Path
to root folder containing train and test sets.")
    parser.add_argument('output_path', type=str,
help="Directory where the created data should be stored.")
    args, _ = parser.parse_known_args()

    if not os.path.exists(args.output_path):
```

```
        os.makedirs(args.output_path)

    process_partition(args, "test")
    process_partition(args, "train")


if __name__ == '__main__':
    main()
```

➜ **split_train_val.py**

```
from __future__ import absolute_import
from __future__ import print_function

import shutil
import argparse
import os


def main():
    parser = argparse.ArgumentParser(description="Split
train data into train and validation sets.")
    parser.add_argument('dataset_dir', type=str, help='Path
to the directory which contains the dataset')
    args, _ = parser.parse_known_args()

    val_patients = set()
    with open(os.path.join(os.path.dirname(__file__),
'resources/valset.csv'), 'r') as valset_file:
        for line in valset_file:
            x, y = line.split(',')
            if int(y) == 1:
                val_patients.add(x)

    with open(os.path.join(args.dataset_dir,
'train/listfile.csv')) as listfile:
        lines = listfile.readlines()
        header = lines[0]
        lines = lines[1:]

    train_lines = [x for x in lines if x[:x.find("_")] not
in val_patients]
    val_lines = [x for x in lines if x[:x.find("_")] in
val_patients]
    assert len(train_lines) + len(val_lines) == len(lines)

    with open(os.path.join(args.dataset_dir,
'train_listfile.csv'), 'w') as train_listfile:
        train_listfile.write(header)
        for line in train_lines:
            train_listfile.write(line)
```

```
    with open(os.path.join(args.dataset_dir,
'val_listfile.csv'), 'w') as val_listfile:
        val_listfile.write(header)
        for line in val_lines:
            val_listfile.write(line)

    shutil.copy(os.path.join(args.dataset_dir,
'test/listfile.csv'),
                os.path.join(args.dataset_dir,
'test_listfile.csv'))


if __name__ == '__main__':
    main()
```

Protocols to run the machine learning models

- **In-Hospital Mortality Prediction**

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from jcopml.pipeline import num_pipe, cat_pipe
from jcopml.utils import save_model, load_model
from jcopml.plot import plot_missing_value
from jcopml.feature_importance import mean_score_decrease

# ------------------------------------------Import Data
X_train = np.load("data/train_X.npy")
y_train = np.load("data/train_y.npy")

X_val = np.load("data/val_X.npy")
y_val = np.load("data/val_y.npy")

X_test = np.load("data/test_X.npy")
y_test = np.load("data/test_y.npy")

# ------------------------------------------Training
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from jcopml.tuning import random_search_params as rsp

preprocessor = ColumnTransformer([
    ('numeric', num_pipe(), list(range(X_train.shape[1])))
])
```

```
pipeline = Pipeline([
    ('prep', preprocessor),
    ('algo', XGBClassifier(n_jobs=-1, random_state=42))
])

model = RandomizedSearchCV(pipeline, rsp.xgb_params, cv=4,
scoring='f1', n_iter=50, n_jobs=-1, verbose=1,
random_state=42)
model.fit(X_train, y_train)

print(model.best_params_)
print(model.score(X_train, y_train), model.score(X_val,
y_val), model.score(X_test, y_test))

# ----------------------------------------Evaluation
Report
from jcopml.plot import plot_classification_report

plot_classification_report(X_train, y_train, X_test,
y_test, model, report=True)

# ----------------------------------------Save Models
save_model(model.best_estimator_, "xgboost.pkl")
```

- **Length of Stay**

```
from __future__ import absolute_import
from __future__ import print_function

from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from mimic3benchmark.readers import LengthOfStayReader
from mimic3models import common_utils
from mimic3models.metrics import print_metrics_regression
from mimic3models.length_of_stay.utils import save_results

import xgboost as xgb
import matplotlib as plt

import os
import numpy as np
import argparse
import json
import pandas as pd
import csv


def read_and_extract_features(reader, count, period,
features):
```

```
    read_chunk_size = 1000
    Xs = []
    ys = []
    names = []
    ts = []
    for i in range(0, count, read_chunk_size):
        j = min(count, i + read_chunk_size)
        ret = common_utils.read_chunk(reader, j - i)
        X =
common_utils.extract_features_from_rawdata(ret['X'],
ret['header'], period, features)
        Xs.append(X)
        ys += ret['y']
        names += ret['name']
        ts += ret['t']
    Xs = np.concatenate(Xs, axis=0)
    return (Xs, ys, names, ts)


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--period', type=str,
default='all', help='specifies which period extract
features from',
                        choices=['first4days',
'first8days', 'last12hours', 'first25percent',
'first50percent', 'all'])
    parser.add_argument('--features', type=str,
default='all', help='specifies what features to extract',
                        choices=['all', 'len',
'all_but_len'])
    parser.add_argument('--data', type=str, help='Path to
the data of length-of-stay task',

default=os.path.join(os.path.dirname(__file__),
'../../../data/length-of-stay/'))
    parser.add_argument('--output_dir', type=str,
help='Directory relative which all output files are
stored',
                        default='.')
    args = parser.parse_args()
    print(args)

    train_reader =
LengthOfStayReader(dataset_dir=os.path.join(args.data,
'train'),

listfile=os.path.join(args.data, 'train_listfile.csv'))
```

```
    val_reader =
LengthOfStayReader(dataset_dir=os.path.join(args.data,
'train'),

listfile=os.path.join(args.data, 'val_listfile.csv'))

    test_reader =
LengthOfStayReader(dataset_dir=os.path.join(args.data,
'test'),

listfile=os.path.join(args.data, 'test_listfile.csv'))

    print('Reading data and extracting features ...')
    n_train = min(100000,
train_reader.get_number_of_examples())
    n_val = min(100000,
val_reader.get_number_of_examples())

    (train_X, train_y, train_names, train_ts) =
read_and_extract_features(
        train_reader, n_train, args.period, args.features)

    (val_X, val_y, val_names, val_ts) =
read_and_extract_features(
        val_reader, n_val, args.period, args.features)

    (test_X, test_y, test_names, test_ts) =
read_and_extract_features(
        test_reader, test_reader.get_number_of_examples(),
args.period, args.features)

    print('Imputing missing values ...')
    imputer = SimpleImputer(missing_values=np.nan,
strategy='mean')
    imputer.fit(train_X)
    train_X =
pd.DataFrame(np.array(imputer.transform(train_X),
dtype=np.float32))
    val_X = pd.DataFrame(np.array(imputer.transform(val_X),
dtype=np.float32))
    test_X =
pd.DataFrame(np.array(imputer.transform(test_X),
dtype=np.float32))

    print('Normalizing the data to have zero mean and unit
variance ...')
    scaler = StandardScaler()
    scaler.fit(train_X)
    train_X = scaler.transform(train_X)
    val_X = scaler.transform(val_X)
    test_X = scaler.transform(test_X)
```

```
    file_name = "{}.{}".format(args.period, args.features)

    model = xgb.XGBRegressor(learning_rate=0.01,
random_state=42)
    kfold = StratifiedKFold(n_splits=4, random_state=7)

    result_dir = os.path.join(args.output_dir, 'results')
    common_utils.create_directory(result_dir)

    with open(os.path.join(result_dir,
'train_{}.json'.format(file_name)), "w") as res_file:
        ret = print_metrics_regression(train_y,
xgb_model.predict(train_X))
        ret = {k: float(v) for k, v in ret.items()}
        json.dump(ret, res_file)

    with open(os.path.join(result_dir,
'val_{}.json'.format(file_name)), 'w') as res_file:
        ret = print_metrics_regression(val_y,
xgb_model.predict(val_X))
        ret = {k: float(v) for k, v in ret.items()}
        json.dump(ret, res_file)

    prediction = xgb_model.predict(test_X)

    with open(os.path.join(result_dir,
'test_{}.json'.format(file_name)), 'w') as res_file:
        ret = print_metrics_regression(test_y, prediction)
        ret = {k: float(v) for k, v in ret.items()}
        json.dump(ret, res_file)

    save_results(test_names, test_ts, prediction, test_y,
                 os.path.join(args.output_dir,
'predictions', file_name + '.csv'))


if __name__ == '__main__':
    main()
```

Script to run the dashboard web application platform

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
import plotly.graph_objs as go
import pandas as pd
from dash.dependencies import Input, Output
```

```
import dash_bootstrap_components as dbc
import dash_table as dt
import plotly.figure_factory as ff
# import base64

# image_filename =
'/Users/anggi/PycharmProjects/Final_Dash/logo-plotly.svg'
# encoded_image = base64.b64encode(open(image_filename,
'rb').read())

# Styling and Dash Components ---------------------------------
-------------------------------------------------------
# external_stylesheets = [
'https://codepen.io/chriddyp/pen/bWLwgP.css',
# 'https://codepen.io/chriddyp/pen/brPBPO.css'
,[dbc.themes.BOOTSTRAP]]
BS =
"https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstra
p.min.css"
app = dash.Dash(external_stylesheets=[BS])
app.config.suppress_callback_exceptions = True
app.title = 'MIMIC-III Dashboard Exploration'

# All Diagnosis table on Tab 1 - Files
# ---------------------------------------------------------------
--------------------------
tab1_stays = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=",",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)

# Nav Bar 2 - Hospital Mortality Files
# ---------------------------------------------------------------
-------------------------
tab2_stays = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=",",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)
tab2_in_cm_test = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=";",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)
tab2_in_cm_test_report = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=";",
```

```
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)
tab2_in_cm_train_report = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=";",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)
tab2_in_cm_train = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=";",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)

# Nav Bar 3 - Length of Stay ---------------------------------
-----------------------------------------------------
tab3 = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=",",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)

tab3_in_cm = pd.read_csv(
    "{PATH_TO_CSV_DATA}",
    sep=",",
    skiprows=0,
    error_bad_lines=False,
    low_memory=False)

# Filter Column used in Filtering Method and Buttons
# Styling Filter Button
# ---------------------------------------------------------------
-----------------------------------
features = ['ETHNICITY', 'DIAGNOSIS', 'GENDER', 'AGE',
'MORTALITY_INUNIT', 'MORTALITY_INHOSPITAL']
opts = [{'label': i, 'value': i} for i in features]

white_button_style = {'background-color': 'white', 'color':
'Blue', }
blue_button_style = {'background-color': 'blue', 'color':
'White'}

# Variables to Call Plotly for Diagrams in Landing Page
# ---------------------------------------------------------------
---------------------------------------
diag_histo = tab2_stays["DIAGNOSIS"]
age_histo = tab2_stays["AGE"]
sex_histo = tab2_stays["GENDER"]
```

```
# Figure for Diagnosis Distributions
fig = go.Figure(
    data=[go.Histogram(x=diag_histo, marker={'color': '
#009dff'})],
    layout={
        'title': {'text': 'Disease Diagnosis Occurrence',
                  'font': {'size': 40}},
        'font': {'family': 'Barlow Semi Condensed'},
        'yaxis': {'title': 'Number of Occurrences'},
    }
)

# Figure for Age and Sex Distribution
# ------------------------------------------------------------
------------------------------------------------
histo = px.histogram(tab2_stays,
                     x=age_histo,
                     color=sex_histo,
                     histnorm='percent',
                     marginal='box',
                     color_discrete_sequence=["#009dff",
"#a7fad5"],
                     title='Distribution of Sex and Age')

# Figure for confusion matrix in length of stay
# ------------------------------------------------------------
----------------------------------------------------
z = [[0, 101746, 12331, 9608, 7015, 4255, 2435, 1358, 817, 85],
     [0, 52694, 9073, 10507, 8387, 4798, 2958, 1328, 688, 44],
     [0, 25084, 5746, 6692, 7549, 5902, 3145, 1387, 755, 33],
     [0, 13295, 3788, 4583, 6161, 5727, 3224, 1390, 619, 8],
     [0, 7353, 2652, 3167, 5188, 5204, 2845, 1590, 508, 35],
     [0, 4577, 1928, 2268, 4195, 4659, 2602, 1547, 440, 10],
     [0, 2875, 1581, 1772, 3567, 3851, 2573, 1333, 519, 6],
     [0, 2221, 1132, 1352, 3008, 3267, 2366, 1315, 460, 24],
     [0, 5095, 3292, 5307, 10971, 13066, 9995, 6328, 1875, 68],
     [0, 3374, 3192, 4012, 9208, 12648, 12994, 11404, 3819, 59]]

x = ['<1', '1-2', '2-3', '3-4', '4-5', '5-6', '6-7', '7-8', '8-
14', '14>']
y = ['<1', '1-2', '2-3', '3-4', '4-5', '5-6', '6-7', '7-8', '8-
14', '14>']

z_text = [[str(y) for y in x] for x in z]

los_cm = ff.create_annotated_heatmap(z, x=x, y=y,
annotation_text=z_text, colorscale='Viridis')

los_cm.update_layout(title_text='<i><b>Confusion
matrix</b></i>',
```

```
                    xaxis=dict(title=''),
                    yaxis=dict(title='y'))

# add custom xaxis title
los_cm.add_annotation(dict(font=dict(color="black", size=14),
                           x=0.5,
                           y=-0.15,
                           showarrow=False,
                           text="Predicted value",
                           xref="paper",
                           yref="paper"))

# add custom yaxis title
los_cm.add_annotation(dict(font=dict(color="black", size=14),
                           x=-0.35,
                           y=0.5,
                           showarrow=False,
                           text="Real value",
                           textangle=-90,
                           xref="paper",
                           yref="paper"))

# adjust margins to make room for yaxis title
los_cm.update_layout(margin=dict(t=50, l=200))

los_cm['data'][0]['showscale'] = True

# --------------------------------------------------------------
------------------------------------------------------

def generate_table(dataframe, max_rows=10):
    return dbc.Table([
        html.Thead(
            html.Tr([html.Th(col) for col in dataframe.columns],
style={'color': '#0000A0'})
        ),
        html.Tbody([
            html.Tr([
                html.Td(dataframe.iloc[i][col]) for col in
dataframe.columns
            ]) for i in range(min(len(dataframe), max_rows))
        ], style={'bordered': True, 'hover': True, 'striped':
True, 'dark': True}
        ),
    ])


# Variables to for Table Figure in Nav 2 - Hospital Mortality
# --------------------------------------------------------------
-------------------------------------
```

```
tab2_in_cm_train_report["Precision"],
tab2_in_cm_train_report["Recall"], tab2_in_cm_train_report["f1-
score"], \
tab2_in_cm_train_report["Support"] =
tab2_in_cm_train_report["measures,precision,recall,f1-
score,support"].str.split(",", 3).str
tab2_in_cm_train_report =
tab2_in_cm_train_report.drop(['measures,precision,recall,f1-
score,support'], axis=1)

tab2_in_cm_test_report["Precision"],
tab2_in_cm_test_report["Recall"], tab2_in_cm_test_report["f1-
score"], \
tab2_in_cm_test_report["Support"] =
tab2_in_cm_test_report["measures,precision,recall,f1-
score,support"].str.split(",", 3).str
tab2_in_cm_test_report =
tab2_in_cm_test_report.drop(['measures,precision,recall,f1-
score,support'], axis=1)

# Calling Dash Layout for Web Page
# ----------------------------------------------------------------
---------------------------------------
app.layout = html.Div(children=[
    dbc.Row(html.Div(children=[
        dbc.Col(html.Div(
            html.H1(children='MIMIC-III Exploration Dashboard',
                    style={
                        'color': '#1c3763',
                        'display': 'inline-block',
                        'float': 'left',
                        'marginLeft': 30,
                        'marginTop': '1%',
                        'marginBottom': '1%',
                        'marginRight': 50,
                        'backgroundColor': 'white',
                        'fontFamily': 'Barlow Semi Condensed',
                        'fontWeight': 610}
                    )
        )
        )
    ],
        style={
            'width': '100%',
            'backgroundColor': 'white',
            'max-width': '1440px'})),

    html.Div(children=[
        dbc.ButtonGroup(children=[  # Use this to edit all
buttons at once
```

```
            dbc.Button('MIMIC Diagnosis Data', id='btn-nclicks-
1', n_clicks=0, color="primary", size='lg'),
            dbc.Button('In-Hospital Mortality', id='btn-nclicks-
2', n_clicks=0, color="primary", size='lg'),
            dbc.Button('Length-of-Stay', id='btn-nclicks-3',
n_clicks=0, color="primary", size='lg'),
        ],
            id='filter-buttons',
            style={
                'width': '85%',
                'maxWidth': '1410px',
                'marginLeft': '240px',
                'fontFamily': 'Barlow Semi Condensed',
                'fontWeight': 610
            },
            className='row')
    ],
        id='filter-button-container'),
    dbc.Row(html.Div(children=[
        dbc.Col(html.Div(children=[
            html.H1('Filters',
                    style={
                        'color': '#1c3763',
                        'marginLeft': '30px',
                        'marginTop': '20px',
                        'marginBottom': '20px',
                        'fontFamily': 'Barlow Semi Condensed',
                        'fontWeight': 610}),
            dbc.Col(html.Div(children=[
                dbc.DropdownMenu(
                    [
                        dbc.DropdownMenuItem('ETHNICITY',
id='ETHNICITY'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem("DIAGNOSIS",
id="DIAGNOSIS"),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('GENDER',
id='GENDER'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('AGE', id='AGE'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('MORTALITY_INUNIT',
id='MORTALITY_INUNIT'),
                        dbc.DropdownMenuItem(divider=True),

dbc.DropdownMenuItem('MORTALITY_INHOSPITAL',
id='MORTALITY_INHOSPITAL'),
                    ],
                    id='opt',
                    label="Filter 1",
```

```
                    bs_size="lg",
                    color="primary"
                )],
                style={
                    'float': 'left',
                    'marginLeft': 12}
            )),
            dbc.Col(html.Div(children=[
                dbc.DropdownMenu(
                    [
                        dbc.DropdownMenuItem('ETHNICITY',
id='ETHNICITY1'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem("DIAGNOSIS",
id="DIAGNOSIS1"),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('GENDER',
id='GENDER1'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('AGE', id='AGE1'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('MORTALITY_INUNIT',
id='MORTALITY_INUNIT1'),
                        dbc.DropdownMenuItem(divider=True),

dbc.DropdownMenuItem('MORTALITY_INHOSPITAL',
id='MORTALITY_INHOSPITAL1'),
                    ],
                    id='opt1',
                    label="Filter 2",
                    bs_size="lg",
                    color="primary",
                )],
                style={
                    'float': 'left',
                    'marginLeft': 12,
                    'paddingTop': 20}
            )),
            dbc.Col(html.Div(children=[
                dbc.DropdownMenu(
                    [
                        dbc.DropdownMenuItem('ETHNICITY',
id='ETHNICITY2'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem("DIAGNOSIS",
id="DIAGNOSIS2"),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('GENDER',
id='GENDER2'),
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('AGE', id='AGE2'),
```

```
                        dbc.DropdownMenuItem(divider=True),
                        dbc.DropdownMenuItem('MORTALITY_INUNIT',
id='MORTALITY_INUNIT2'),
                        dbc.DropdownMenuItem(divider=True),

dbc.DropdownMenuItem('MORTALITY_INHOSPITAL',
id='MORTALITY_INHOSPITAL2'),
                    ],
                    id='opt2',
                    label="Filter 3",
                    bs_size="lg",
                    color="primary",
                )],
                style={
                    'float': 'left',
                    'marginLeft': 12,
                    'paddingTop': 20})),
        ],
            style={
                'width': '15%',
                'marginRight': 0,
                'marginLeft': '4px',
                'height': '100%',
                'display': 'inline-block',
                'backgroundColor': 'white',
                'float': 'left'
            },
            className="one column")),  # Is the block for
"Filter" and dropdown setting panel

        dbc.Col(html.Div(children=[
            html.Div(children=[
                html.Div(
                    id='container-button-timestamp'
                ),
            ])
        ])),
    ],
        style={
            'width': '100%',
            'maxWidth': '1410px',
            'backgroundColor': 'white'
        },
        className='container')),
],
    id='full-layout')


@app.callback(Output('container-button-timestamp', 'children'),
              [Input('btn-nclicks-1', 'n_clicks'),
               Input('btn-nclicks-2', 'n_clicks'),
```

```
                Input('btn-nclicks-3', 'n_clicks'),
                Input('ETHNICITY', "n_clicks"),
                Input('DIAGNOSIS', "n_clicks"),
                Input('GENDER', "n_clicks"),
                Input('AGE', "n_clicks"),
                Input('MORTALITY_INUNIT', "n_clicks"),
                Input('MORTALITY_INHOSPITAL', "n_clicks"),
                Input('ETHNICITY1', "n_clicks"),
                Input('DIAGNOSIS1', "n_clicks"),
                Input('GENDER1', "n_clicks"),
                Input('AGE1', "n_clicks"),
                Input('MORTALITY_INUNIT1', "n_clicks"),
                Input('MORTALITY_INHOSPITAL1', "n_clicks"),
                Input('ETHNICITY2', "n_clicks"),
                Input('DIAGNOSIS2', "n_clicks"),
                Input('GENDER2', "n_clicks"),
                Input('AGE2', "n_clicks"),
                Input('MORTALITY_INUNIT2', "n_clicks"),
                Input('MORTALITY_INHOSPITAL2', "n_clicks")])
# Define Functions for Button Clicks on Filter Page
# ------------------------------------------------------------------
-------------------------------------
def displayClick(btn1, btn2, btn3, x1, x2, x3, x4, x5, x6, y1,
y2, y3, y4, y5, y6, z1, z2, z3, z4, z5, z6):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    ctx = dash.callback_context
    # Define Nav bar 1 for "MIMIC Datasets"
    # ------------------------------------------------------------
----------------------------------------
    if 'btn-nclicks-1' in changed_id:
        data = tab1_stays.to_dict('rows')
        columns = [{"name": i, "id": i} for i in
tab1_stays.columns]
        return (html.Div(
            children=[
                dt.DataTable(
                    data=data,
                    columns=columns,
                    page_size=20,
                    style_table={'overflowX': 'auto'},
                    style_cell={
                        'height': 'auto',
                        'maxWidth': '160px',
                        'overflow': 'hidden',
                        'textOverflow': 'ellipsis',
                        'textAlign': 'left',
                        'padding': '10px',
                        'marginLeft': '50xp',
                        'fontFamily': 'Barlow Semi Condensed',
                        'fontSize': '15px'
```

```
                    },
                    style_cell_conditional=[{
                        'if': {'column_id': 'SUBJECT_ID'},
                        'textAlign': 'right'
                    }],
                    style_as_list_view=True,
                    filter_action='native',
                    style_header={
                        'backgroundColor': 'aliceblue',
                        'font_family': 'Barlow Semi Condensed',
                        'fontSize': '15px',
                        'color': '#1c3763',
                        'fontWeight': 'bold'
                    },
                    style_data_conditional=[{
                        'if': {'row_index': 'odd'},
                        'backgroundColor': 'rgb(248, 248, 248)'
                    }],
                )],
            id='table1',
            style={
                'width': '100%',
                'maxWidth': '1209px',
                'height': '30%',
                'marginLeft': 0,
                'float': 'left',
                'display': 'inline-block',
                'marginRight': 0,
                'position': 'absolute'}))
    # Define Nav bar 2 for "In-Hospital Mortality"
    # ------------------------------------------------------------
----------------------------------------
    elif 'btn-nclicks-2' in changed_id:
        data0 = tab2_stays.to_dict('rows')
        data1 = tab2_in_cm_test_report.to_dict('rows')
        data2 = tab2_in_cm_train_report.to_dict('rows')
        columns0 = [{"name": i, "id": i, } for i in
tab2_stays.columns]
        columns1 = [{"name": i, "id": i, } for i in
tab2_in_cm_test_report.columns]
        columns2 = [{"name": i, "id": i, } for i in
tab2_in_cm_train_report.columns]
        # arr = tab2_stays.to_numpy()
        return (
            dbc.Row(
                html.Div(
                    children=[
                        dt.DataTable
                        (data=data0,
                         columns=columns0,
                         page_size=7,
```

```
                        style_table={'overflowX': 'auto'},
                        style_cell={
                            'height': 'auto',
                            'maxWidth': '160px',
                            'overflow': 'hidden',
                            'textOverflow': 'ellipsis',
                            'textAlign': 'left',
                            'padding': '10px',
                            'marginLeft': '50xp',
                            'fontFamily': 'Barlow Semi
Condensed',
                            'fontSize': '15px'
                        },
                        style_cell_conditional=[{
                            'if': {'column_id': 'SUBJECT_ID'},
                            'textAlign': 'right'}],
                        style_as_list_view=True,
                        filter_action='native',
                        style_header={'backgroundColor':
'aliceblue',
                                      'font_family': 'Barlow
Semi Condensed',
                                      'font_size': '15px',
                                      'color': '#1c3763',
                                      'fontWeight': 'bold'
                                      },
                        style_data_conditional=[{
                            'if': {'row_index': 'odd'},
                            'backgroundColor': 'rgb(248, 248,
248)'}],
                        )],
                    style={'width': '100%',
                           'maxWidth': '1410px',  # Styling for
tab 2 page numbers
                           'marginLeft': 0,
                           'float': 'left',
                           'display': 'inline-block',
                           'marginRight': 0})
            ),
            # Model Result Block
            # ------------------------------------------------
------------------------------------------------
            dbc.Row(
                html.Div(
                    children=[
                        dbc.Button("MODEL RESULT",
                                   color="primary",
                                   block=True,
                                   disabled=True,
                                   style={'color': '#ffffff',
```

```
                                            'font_family': 'Barlow
Semi Condensed',
                                            'fontWeight':
'bold'}),
                    ],
                    style={'width': '100%'})
            ),
            # Test Score Heatmap
            # ------------------------------------------------
----------------------------------------------
            dbc.Row(
                html.Div(
                    children=[
                        html.Div(children=[
                            dcc.Graph(
                                figure={
                                    'data': [
                                        go.Heatmap(
                                            x=['Predicted 0',
'Predicted 1'],
                                            y=['Actual 1',
'Actual 0'],
                                            z=[[.1, .4],
                                               [1.0, .07]],
                                            colorscale='bugn',
                                            text=[['241',
'133'],
                                                  ['2772',
'90']]
                                        )],
                                    'layout': go.Layout(
                                        # hovermode="x unified",
                                        title={
                                            'text': 'Test Score:
0.446',
                                            'font_family':
'Barlow Semi Condensed',
                                            'font_size': 20,
                                            'x': 0.5,
                                            'xanchor': 'center',
                                            'yanchor': 'top'},

                                    )})
                            ],
                            style={'float': 'left',
                                   'display': 'inline-block',
                                   'width': '42%',
                                   'marginLeft': '10px'},
                            className='three columns'),

                        html.Div(children=[
```

```
                            dcc.Graph(
                                figure={
                                    'data': [
                                        go.Heatmap(
                                            x=['Predicted 0',
'Predicted 1'],
                                            y=['Actual 1',
'Actual 0'],
                                            z=[[.1, .4],
                                               [1.0, .07]],
                                            colorscale=[[0,
'rgb(224, 243, 248)'],
                                                        [1,
'rgb(12, 51, 131)']],
                                            text=[['759',
'1228'],
                                                  ['12572',
'122']]

                                        )],
                                    'layout': go.Layout(
                                        # hovermode="x unified",
                                        title={'text': 'Train
Score: 0.736', 'font_family': 'Barlow Semi Condensed',
                                               'font_size': 20,
                                               'x': 0.5,
                                               'xanchor':
'center',
                                               'yanchor':
'top'},
                                        xaxis={'ticksuffix':
''},
                                        yaxis={'title': ''},
                                        # annotation=[['241',
'133'],
                                        #            ['2772',
'90']]

                                    )})
                        ], style={'float': 'right',
                                  'display': 'inline-block',
                                  'width': '42%',
                                  'marginLeft': '10px'},
                            className='three columns'),
                    ],
                    id='heatmaps',
                    style={'width': '90%',
                           'max-width': '1410px'},
                    className='container')),

            # Test and Train Label
```

```
            dbc.Row(
                html.Div(
                    children=[
                        html.Div(children=[
                            html.H4('Test',
                                    style={'color': ' #0000A0',
                                           'font_family':
'Barlow Semi Condensed'}),
                            dt.DataTable(
                                data=data1,
                                columns=columns1,
                                page_size=20,
                                style_cell={'textAlign': 'left',
                                            'padding': '10px',
                                            'marginLeft':
'80xp',
                                            'font_family':
'Barlow Semi Condensed',
                                            'font_size':
'15px'},
                                style_as_list_view=True,
                                filter_action='native',
                                style_header={
                                    'backgroundColor':
'aliceblue',
                                    'font_family': 'Barlow Semi
Condensed',
                                    'font_size': '20px',
                                    'color': '#0000A0',
                                    'fontWeight': 'bold'},
                                style_data_conditional=[
                                    {
                                        'if': {'row_index':
'odd'},
                                        'backgroundColor':
'rgb(248, 248, 248)'
                                    }
                                ])
                        ],
                            style={'float': 'left',
                                   'display': 'inline-block',
                                   'width': '42%',
                                   'marginRight': 100},
                            className='three columns'),

                        html.Div(children=[
                            html.H4('Train',
                                    style={'color': ' #0000A0',
                                           'font_family':
                                               'Barlow Semi
Condensed'}),
```

```
                            dt.DataTable(
                                data=data2,
                                columns=columns2,
                                page_size=20,
                                style_cell={'textAlign': 'left',
'padding': '10px', 'marginLeft': '80xp',
                                            'font_family':
'Barlow Semi Condensed',
                                            'font_size':
'15px'},
                                style_as_list_view=True,
                                filter_action='native',
                                style_header={
                                    'backgroundColor':
'aliceblue',
                                    'font_family': 'Barlow Semi
Condensed',
                                    'font_size': '20px',
                                    'color': '#1c3763',
                                    'fontWeight': 'bold'},
                                style_data_conditional=[
                                    {
                                        'if': {'row_index':
'odd'},
                                        'backgroundColor':
'rgb(248, 248, 248)'
                                    }
                                ]),
                        ],
                            style={'float': 'right',
                                   'display': 'inline-block',
                                   'width': '42%', 'marginLeft':
'10px'},
                            className='three columns'),
                    ],
                    id='test-train-label',
                    style={'width': '90%',
                           'max-width': '1410px'},
                    className='container')),
        )
    # Navigation page for tab 3
    # ---------------------------------------------------------
-----------------------------------------
    elif 'btn-nclicks-3' in changed_id:
        data0 = tab2_stays.to_dict('rows')
        columns0 = [{"name": i, "id": i, } for i in
tab2_stays.columns]
        data3 = tab3_in_cm.to_dict('rows')
        columns3 = [{"name": i, "id": i, } for i in
tab3_in_cm.columns]
```

```
        return (
            dbc.Row(
                html.Div(
                    children=[
                        dt.DataTable(
                            data=data0,
                            columns=columns0,
                            page_size=10,
                            style_table={'overflowX': 'auto'},
                            style_cell={'textAlign': 'left',
                                        'padding': '10px',
                                        'marginLeft': '50xp',
                                        'font_family': 'Barlow
Semi Condensed',
                                        'font_size': '15px'
                                        },
                            style_as_list_view=True,
                            filter_action='native',
                            style_header={
                                'backgroundColor': 'aliceblue',
                                'font_family': 'Barlow Semi
Condensed',
                                'font_size': '15px',
                                'color': '#1c3763',
                                'fontWeight': 'bold'},
                            style_cell_conditional=[{
                                'if': {'column_id':
'SUBJECT_ID'},
                                'textAlign': 'right'}],
                            style_data_conditional=[
                                {
                                    'if': {'row_index': 'odd'},
                                    'backgroundColor': 'rgb(248,
248, 248)'
                                }
                            ], )],
                    id='tab3_table',
                    style={'width': '100%',
                           'maxWidth': '1410px',  # Styling for
tab 3 page numbers
                           'marginLeft': 0,
                           'float': 'left',
                           'display': 'inline-block',
                           'marginRight': 0})),

            # Model ResultPane
            # ------------------------------------------------
------------------------------------------
            dbc.Row(
                html.Div(
                    children=[
```

```
                        dbc.Button("MODEL RESULT",
                                   color="primary",
                                   block=True,
                                   disabled=True,
                                   style={'color': '#ffffff',
                                          'fontFamily': 'Barlow
Semi Condensed',
                                          'fontWeight':
'bold'}),
                    ],
                    style={
                        'width': '100%'
                    }
                )),

            # Model For Table 3
            # ------------------------------------------------
------------------------------------------------
            dbc.Row(
                html.Div(
                    children=[
                        html.Div(children=[
                            html.H4('Test Result',
                                    style={'color': ' #0000A0',
                                           'font_family':
'Barlow Semi Condensed'}),
                            dt.DataTable(
                                data=data3,
                                columns=columns3,
                                page_size=20,
                                style_cell={'textAlign': 'left',
                                            'padding': '10px',
                                            'marginLeft':
'80xp',
                                            'minWidth': '100px',
                                            'width': '100px',
                                            'whiteSpace':
'normal',
                                            'height': 'auto',
                                            'maxWidth': '100px',
                                            'font_family':
'Barlow Semi Condensed',
                                            'font_size':
'15px'},
                                style_as_list_view=True,
                                filter_action='native',
                                style_header={
                                    'height': 'auto',
                                    'backgroundColor':
'aliceblue',
```

```
                                        'font_family': 'Barlow Semi
Condensed',
                                        'font_size': '15px',
                                        'color': '#0000A0',
                                        'fontWeight': 'bold'},
                                    style_data_conditional=[
                                        {
                                            'if': {'row_index':
'odd'},
                                            'backgroundColor':
'rgb(248, 248, 248)'
                                        }
                                    ])
                            ],
                                style={'float': 'left',
                                       'display': 'inline-block',
                                       'width': '42%',
                                       'paddingTop': '30px',
                                       'marginRight': 30},
                                className='three columns'),

                            html.Div(children=[
                                dcc.Graph(
                                    id='los_cm',
                                    figure=los_cm
                                ),
                            ], style={'float': 'right',
                                      'display': 'inline-block',
                                      'width': '55%',
                                      'paddingTop': '30px',
                                      'marginLeft': '10px'},
                                className='three columns'),
                        ],
                        id='los',
                        style={'width': '100%',
                               'max-width': '1410px'},
                        className='container'))
            )

        # This triggers the whole button filters in the Side Panel
        # ------------------------------------------------------------
----------------------------------------
        elif ctx.triggered:
            button_id = ctx.triggered[0]["prop_id"].split(".")[0]
            # First Filter Button for Ethnicity
            if button_id in ['ETHNICITY']:
                df = tab2_stays[['ETHNICITY']]
                data = tab2_stays.to_dict('rows')
                columns = [{"name": i, "id": i, } for i in (df)]
```

```
            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                        ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                              'color':
'#1c3763',

'fontWeight': 'bold'},

style_data_conditional=[
                                                              {
                                                                  'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                              }
                                                          ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['DIAGNOSIS']:
            df = tab2_stays[['DIAGNOSIS']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',
```

```
'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                        ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['GENDER']:
            df = tab2_stays[['GENDER']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',
```

```
'font_size': '15px'}
                                                        ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                            'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                            {
                                                                'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                            }
                                                        ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['AGE']:
            df = tab2_stays[['AGE']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                        ,
style_as_list_view=True,
```

```
filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['MORTALITY_INUNIT']:
            df = tab2_stays[['MORTALITY_INUNIT']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,

style_as_list_view=True,

filter_action='native',
```

```
style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                                'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                                {
                                                                    'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                                }
                                                            ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['MORTALITY_INHOSPITAL']:
            df = tab2_stays[['MORTALITY_INHOSPITAL']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                            ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',
```

```
'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                            style={'width': '20%',
                                   'height': '30%',
                                   'marginLeft': 0,
                                   'float': 'left',
                                   'display': 'inline-block',
                                   'marginRight': 0}),

        elif button_id in ['ETHNICITY1']:
            df = tab2_stays[['ETHNICITY']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
```

```
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                            style={'width': '20%',
                                   'height': '30%',
                                   'marginLeft': 0,
                                   'float': 'left',
                                   'display': 'inline-block',
                                   'marginRight': 0})

        elif button_id in ['DIAGNOSIS1']:
            df = tab2_stays[['DIAGNOSIS']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,

style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',
```

```
'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['GENDER1']:
            df = tab2_stays[['GENDER']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                   ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
```

```
                                                                {
                                                                    'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                                }
                                                            ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        elif button_id in ['AGE1']:
            df = tab2_stays[['AGE']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                   ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                                'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                                {
                                                                    'if':
{'row_index': 'odd'},
```

```
'backgroundColor': 'rgb(248, 248, 248)'
                                                    }
                                                ], )],
                            style={'width': '20%', 'height':
'30%', 'marginLeft': 0, 'float': 'left',
                                    'display': 'inline-block',
'marginRight': 0})

        elif button_id in ['MORTALITY_INUNIT1']:
            df = tab2_stays[['MORTALITY_INUNIT']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                ,

style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                    'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                    {
                                                        'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                    }
                                                ], )],
                            style={'width': '20%',
```

```
                                    'height': '30%',
                                    'marginLeft': 0,
                                    'float': 'left',
                                    'display': 'inline-block',
                                    'marginRight': 0})

        elif button_id in ['MORTALITY_INHOSPITAL1']:
            df = tab2_stays[['MORTALITY_INHOSPITAL']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                   ,
style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                   ], )],
                                style={'width': '20%',
                                    'height': '30%',
                                    'marginLeft': 0,
                                    'float': 'left',
                                    'display': 'inline-block',
```

```
                                    'marginRight': 0})

        # Button Filter for Ethnicity
        elif button_id in ['ETHNICITY2']:
            df = tab2_stays[['ETHNICITY']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,

style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})
```

```
        # Button Filter for Diagnosis
        elif button_id in ['DIAGNOSIS2']:
            df = tab2_stays[['DIAGNOSIS']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,

style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        # Button Filter for Genders
        elif button_id in ['GENDER2']:
```

```
            df = tab2_stays[['GENDER']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,
style_as_list_view=True,

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        # Button Filter for Age
        elif button_id in ['AGE2']:
            df = tab2_stays[['AGE']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]
```

```
            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,

style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                                style={'width': '20%',
                                       'height': '30%',
                                       'marginLeft': 0,
                                       'float': 'left',
                                       'display': 'inline-block',
                                       'marginRight': 0})

        # Button Filter for Mortality in Unit
        elif button_id in ['MORTALITY_INUNIT2']:
            df = tab2_stays[['MORTALITY_INUNIT']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,
```

```
style_cell={'textAlign': 'left', 'padding': '10px',

'marginLeft': '50xp',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px'}
                                                    ,

style_as_list_view=True,

filter_action='native',

style_header={

'backgroundColor': 'aliceblue',

'font_family': 'Barlow Semi Condensed',

'font_size': '15px',
                                                        'color':
'#0000A0',

'fontWeight': 'bold'},

style_data_conditional=[
                                                        {
                                                            'if':
{'row_index': 'odd'},

'backgroundColor': 'rgb(248, 248, 248)'
                                                        }
                                                    ], )],
                              style={'width': '20%',
                                     'height': '30%',
                                     'marginLeft': 0,
                                     'float': 'left',
                                     'display': 'inline-block',
                                     'marginRight': 0})

        # Button Filter for Mortality in Hospital
        elif button_id in ['MORTALITY_INHOSPITAL2']:
            df = tab2_stays[['MORTALITY_INHOSPITAL']]
            data = tab2_stays.to_dict('rows')
            columns = [{"name": i, "id": i, } for i in (df)]

            return html.Div(children=[dt.DataTable(data=data,
columns=columns, page_size=20,

style_cell={'textAlign': 'left', 'padding': '10px',
```

```
'marginLeft': '50xp',
'font_family': 'Barlow Semi Condensed',
'font_size': '15px'}
                                                    ,
style_as_list_view=True,
filter_action='native',
style_header={
'backgroundColor': 'aliceblue',
'font_family': 'Barlow Semi Condensed',
'font_size': '15px',
                                                    'color':
'#0000A0',
'fontWeight': 'bold'},
style_data_conditional=[
                                                    {
                                                        'if':
{'row_index': 'odd'},
'backgroundColor': 'rgb(248, 248, 248)'
                                                    }
                                                ], )],
                            style={'width': '20%',
                                   'height': '30%',
                                   'marginLeft': 0,
                                   'float': 'left',
                                   'display': 'inline-block',
                                   'marginRight': 0})
    # Shows landing page
    # ------------------------------------------------------------
----------------------------------------
    else:
        data = tab2_stays.to_dict('rows')
        columns = [{"name": i, "id": i, } for i in
(tab2_stays.columns)]
        return (
            dbc.Row(
                html.Div(
                    children=[
                        dt.DataTable(
                            data=data,
                            columns=columns,
```

```
                            page_size=10,
                            style_table={'overflowX': 'auto'},
                            style_cell={'textAlign': 'left',
                                        'padding': '10px',
                                        'marginLeft': '50xp',
                                        'font_family': 'Barlow
Semi Condensed',
                                        'font_size': '15px'
                                        },
                            style_as_list_view=True,
                            style_cell_conditional=[{
                                'if': {'column_id':
'SUBJECT_ID'},
                                'textAlign': 'right'}],
                            filter_action='native',
                            style_header={
                                'backgroundColor': 'aliceblue',
                                'font_family': 'Barlow Semi
Condensed',
                                'font_size': '15px',
                                'color': '#0000A0',
                                'fontWeight': 'bold'},
                            style_data_conditional=[
                                {
                                    'if': {'row_index': 'odd'},
                                    'backgroundColor': 'rgb(248,
248, 248)'
                                }
                            ], )],
                    style={'width': '100%',
                           'maxWidth': '1410px',
                           'marginLeft': 0,
                           'float': 'left',
                           'display': 'inline-block',
                           'marginRight': 0})
            ),
            # Diagrams Name
            dbc.Row(
                html.Div(
                    children=[
                        dbc.Button("MODEL RESULT",
                                   color="primary",
                                   block=True,
                                   disabled=True,
                                   style={'color': '#ffffff',
                                          'font_family': 'Barlow
Semi Condensed',
                                          'fontWeight':
'bold'}),
                    ],
                    style={'width': '100%',
```

```
                            'maxWidth': '1410px'})
            ),
            # Graph for Diagnosis
            # ------------------------------------------------------
------------------------------------------------
            dbc.Row(
                html.Div(
                    dcc.Graph(
                        id='diag',
                        figure=fig
                    ),
                    style={
                        'width': '100%',
                        'maxWidth': '1440px'
                    }
                ),
            ),
            dbc.Row(
                html.Div(
                    dcc.Graph(
                        id='age_sex',
                        figure=histo
                    ),
                    style={
                        'width': '100%',
                        'maxWidth': '1440px'
                    }
                )
            )
        )


# App Callbacks to make the tabs Interactive
@app.callback(Output('btn-nclicks-1', 'style'),
              [Input('btn-nclicks-1', 'n_clicks'),
               Input('btn-nclicks-2', 'n_clicks'),
               Input('btn-nclicks-3', 'n_clicks')])
def change_button_style(n_clicks, n_clicks1, n_clicks2):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    if 'btn-nclicks-1' in changed_id:
        return white_button_style


@app.callback(Output('btn-nclicks-2', 'style'),
              [Input('btn-nclicks-1', 'n_clicks'),
               Input('btn-nclicks-2', 'n_clicks'),
               Input('btn-nclicks-3', 'n_clicks')])
def change_button_style(n_clicks, n_clicks1, n_clicks2):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
```

```
    if 'btn-nclicks-2' in changed_id:
        return white_button_style


@app.callback(Output('btn-nclicks-3', 'style'),
              [Input('btn-nclicks-1', 'n_clicks'),
               Input('btn-nclicks-2', 'n_clicks'),
               Input('btn-nclicks-3', 'n_clicks')])
def change_button_style(n_clicks, n_clicks1, n_clicks2):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    if 'btn-nclicks-3' in changed_id:
        return white_button_style


if __name__ == '__main__':
    app.run_server(debug=True)
```