Introduction

We are attempting to leverage statistical and Machine Learning methods to predict patient survival. The idea is to create a model that helps the provider in better risk assessment. Which in turn can provide better intensive care facilities, provide personalized treatment, allocate resources for patients efficiently like getting the right doctors at the right time, timely change of medication or treatment, movement of patient to another facility etc.

Considering our team's work experience, past projects and interest in healthcare, we are participating in the datathon which focuses on patient survival prediction.

Related Work

Apache System

The APACHE system was first published long ago and provided predictions for patient mortality based upon data collected in the ICU. While the initial system was based on expert rules, later updates used data driven methods. In the kaggle competition we are provided all the fields that the apache system utilizes and therefore the models built are expected to perform better than the apache system.

Past models have been limited by technical and practical considerations, often using summary data from an entire day of a patient's ICU stay which was manually documented by trained personnel [1]. There have also been models that leverage Multivariable logistic regression modelling to evaluate information from history, physical examination (for example, blood pressure), drug use, and quality of life questionnaires that independently contributed to the prediction of death [2].

It is worth noting that the models mostly used purely physiology, laboratory measurements, and minimal demographic variables available on admission, whereas the APACHE system utilized over 100 diagnostic categories, comorbid burden, and treatment. These additional features can often require manual collection which can complicate automatic application of the model.

Other Methods

Apart from the apache system other methods that have gained momentum include models based on Gradient Boosting , Lasso. There have also been models for specific conditions which instead of being generally applicable, find their use in certain specific situations like for patients with acute renal failure [3] or for traumatically injured patients [4]

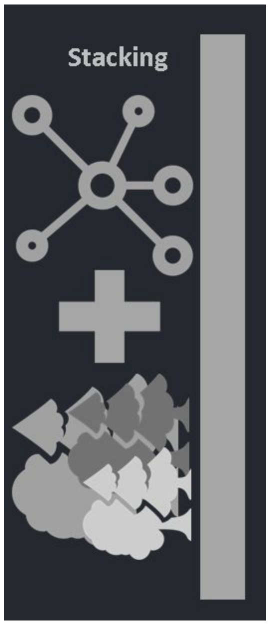
Project Topic and Proposed Solution

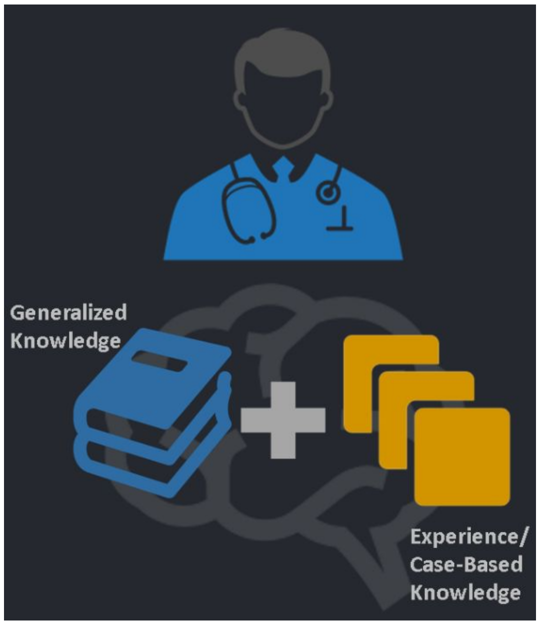
Project Topic : Patient Survival Prediction using Machine Learning and Statistical Methods

Dataset: Primary Dataset to be used is intensive care unit data from hospitals in multiple geographic regions, hosted on Kaggle.

Proposed Solution : Based on the initial research we are planning to use an ensemble method by combining Machine learning and traditional statistical methods. The choice of methods and algorithms would be driven by the EDA and Feature engineering results, as well as our choice of handling missing data.

Our initial intuition suggests **a stacking approach.** Trying to model after how a doctor or a nurse would approach the problem. Considering that a doctor not only has a vast knowledge that was gained from medical school but an even greater knowledge based on experience which allows them to begin investigation from the generalized knowledge but then being able to map out and fit the current patient to any existing case that they had previously handled gives the doctors a great advantage. What this means for us is that we should **not just try generalizing the patterns in medical data but try and use neighborhood based approach** to be able to use the most similar case that we know about as a reference in predicting patient mortality.





Research Question

- **What features impact the survival of the patients?**
- **How does age affect patient survival? Do older patients have a higher chance of death?**
- **Which disease results in the most number of deaths?**
- **Patients with which disease have a high number of admissions and readmissions?**

Data And Domain Description

The data has been taken from Kaggle Competition where the challenge is to create a model that uses data from the first 24 hours of intensive care to predict patient survival. This dataset has data for more than 130,000 hospital Intensive Care Unit (ICU) visits from patients spanning one year time frame.

Dataset Link: https://www.kaggle.com/c/widsdatathon2020/datas

Github Link: https://github.com/abhijeetdtu/Datathon

EDA

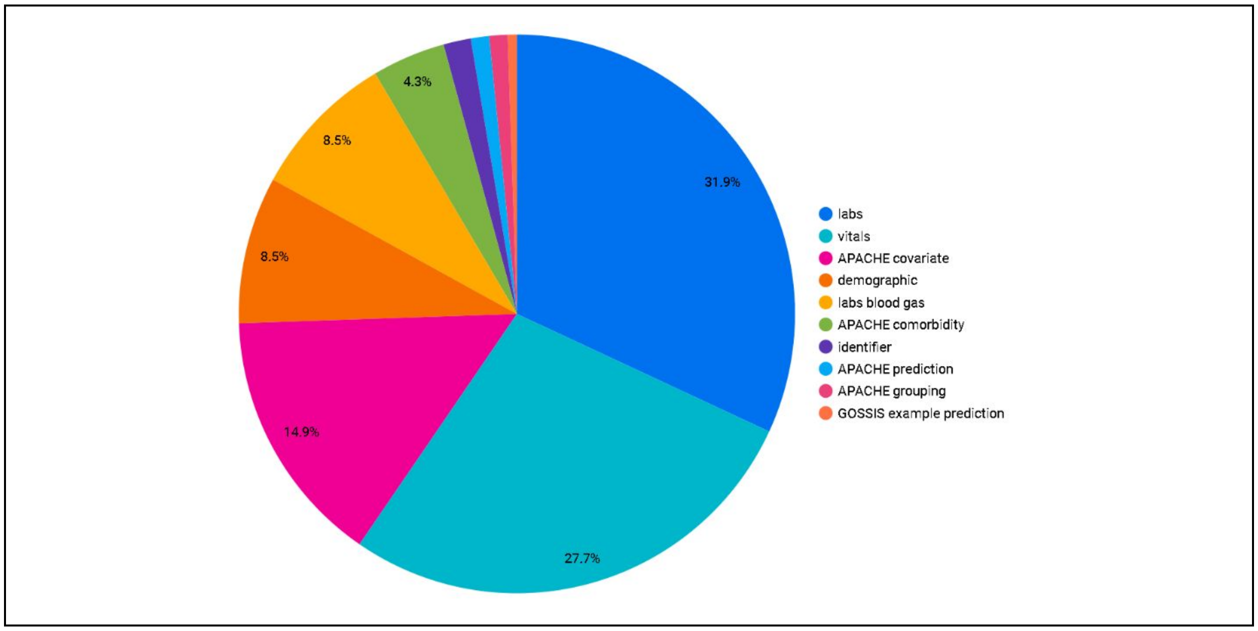
The dataset has a total 186 variables including the dependent variable. The dependent variable is **hospital_death**.

We performed initial exploratory data analysis of the dataset in order to understand the data better by visualizing on Google Cloud Studio.

Following are some of the insights from the visualization.

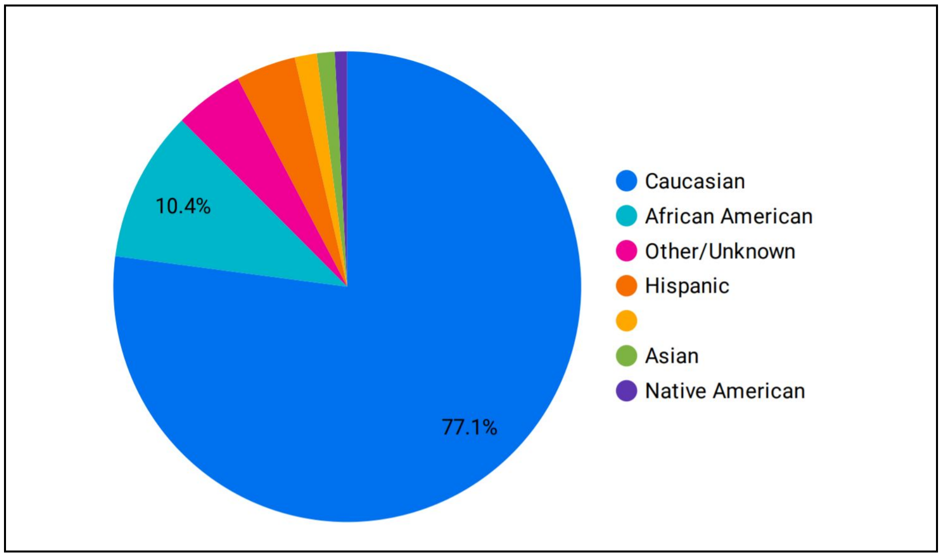
We tried to follow a line of thought as to what type of data we have at hand, who are the patients, what are the demographics , what are the main conditions because of which patients are being admitted etc.

What Data/Features do we have?

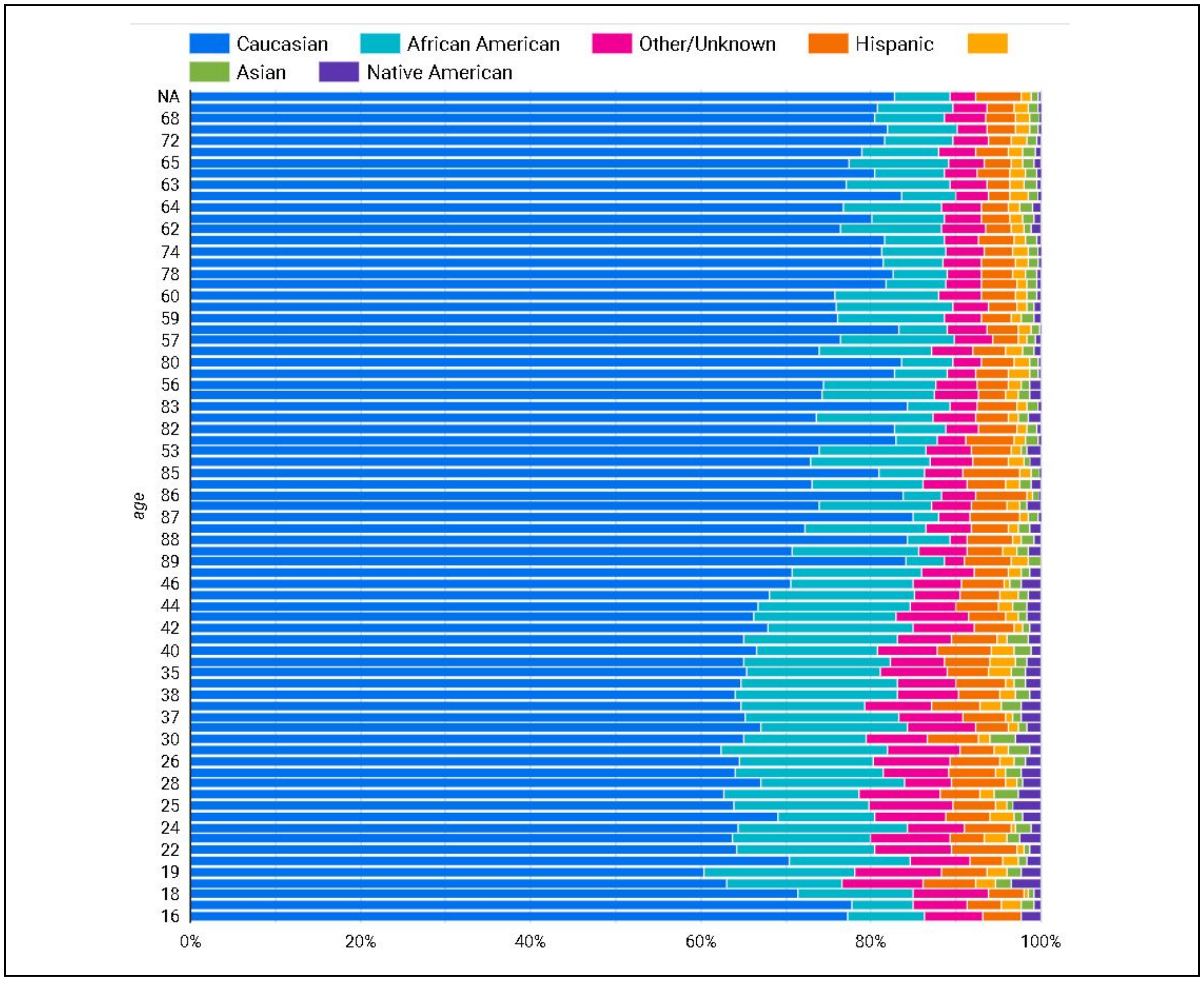


We began by examining what kind of data we have. For this we looked into the data dictionary. We see that **labs and vital readings** account for more than 50% of the data. i.e most of the columns in the dataset belong to these categories. Add to that **APACHE covariates** and we can account for almost 75% of the data. Apache Grouping and Prediction categories have very few columns compared to others.

Who is getting admitted?

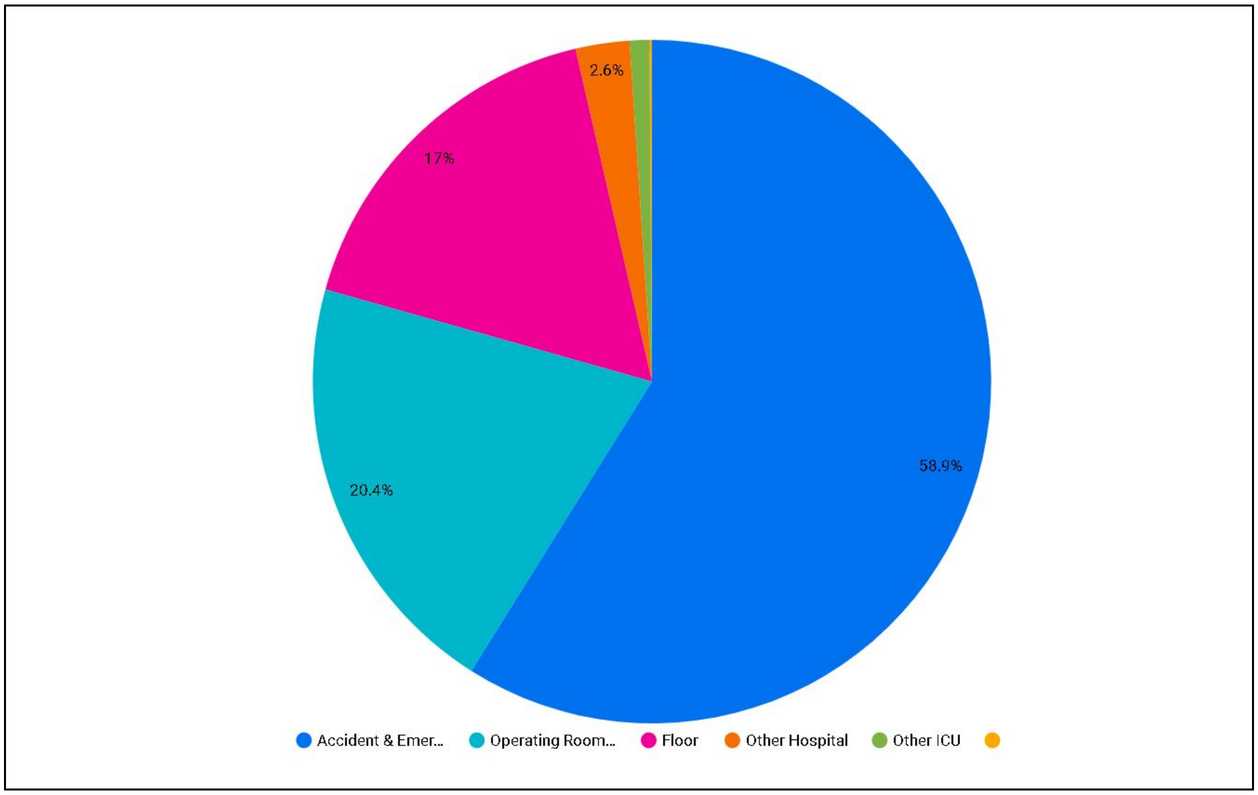


To understand whose data we actually have, we try to drill down along demographic lines. We use **Ethnicity and Age** to drill into the data. From the Ethnicity Pie Chart, we see that most of the individuals in our dataset are **Caucasians** followed by African Americans.



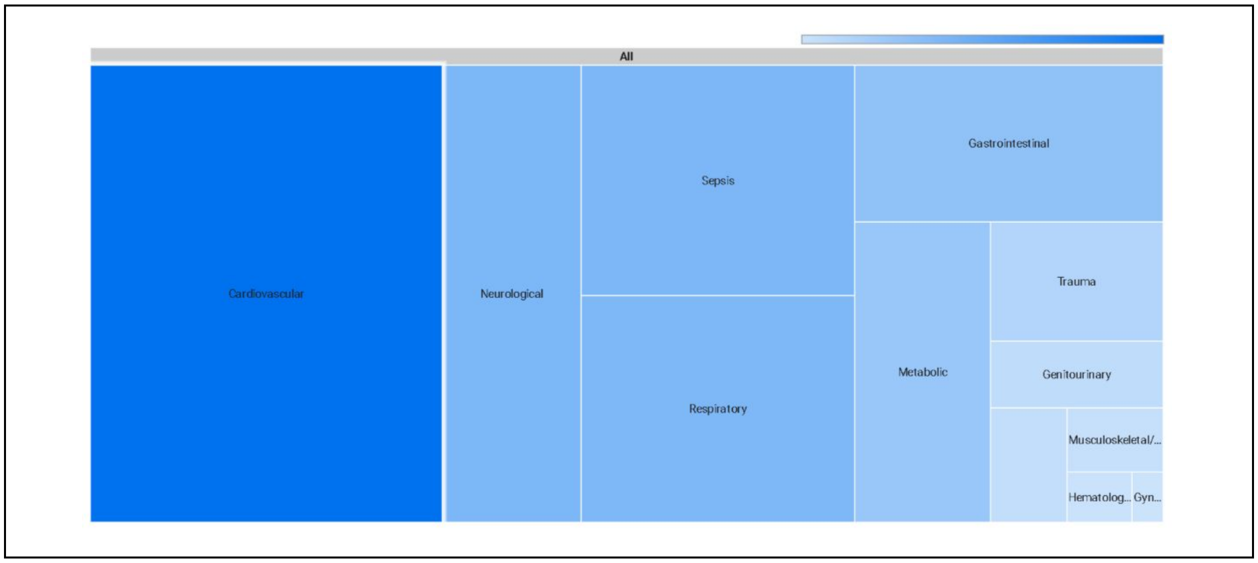
If We add another dimension - Age to the mix - we see an interesting pattern in the bar chart. We see that for African Americans in particular, the **proportion of patients declines with age.** We see that there are a number of young african american patients accounting for around 25-30% of total young patients but as we go upward in the chart to the older population we can see that the proportion drops to an average of around 10%.

Where did they come from?



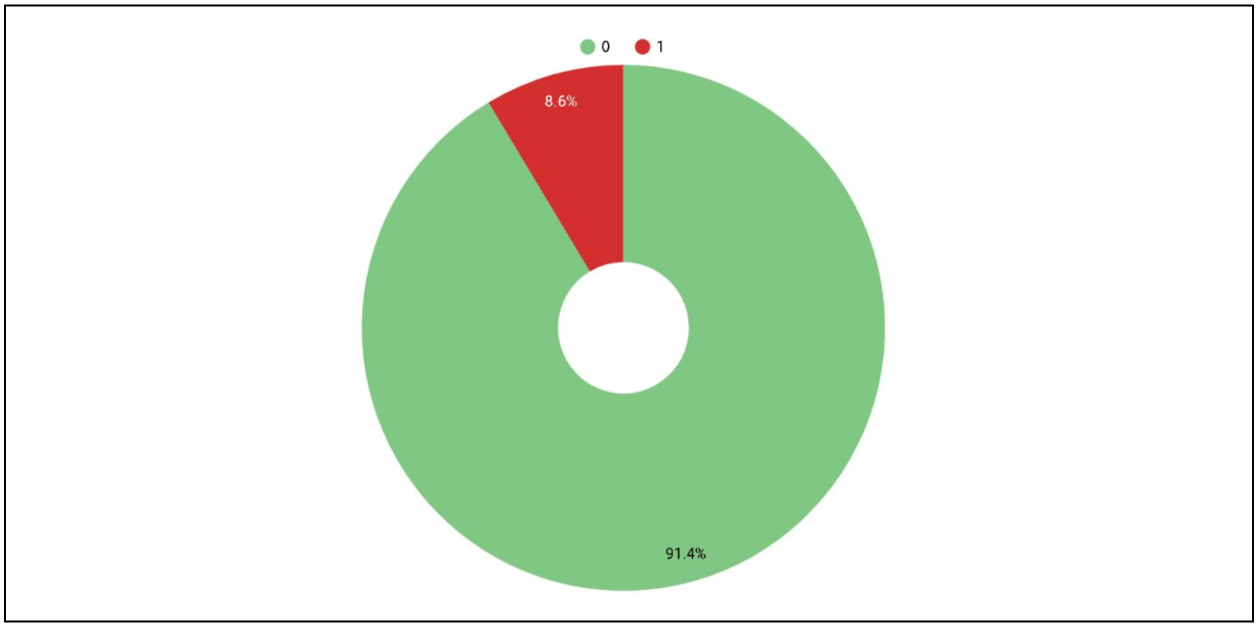
Next step is to explore how people ended up in the ICU. We can see that **Accident & Emergency** account for most of the ICU admissions which is 58.9% of the total admissions. Followed by Movement from the Operating room and then from the hospital wards.

Why are patients getting admitted?



Having seen that most of the people end up in ICU because of an accident or emergency. We would like to see what kind of cases are usually seen in the ICU. For that we make a tree map of the diagnosis column. As it turns out most of the cases are Cardiovascular - heart attacks. We see that neurological cases , which might be from car crashes or other accidents , Respiratory and Sepsis cases also account for a large chunk of cases

How Bad is it? : Hospital Deaths

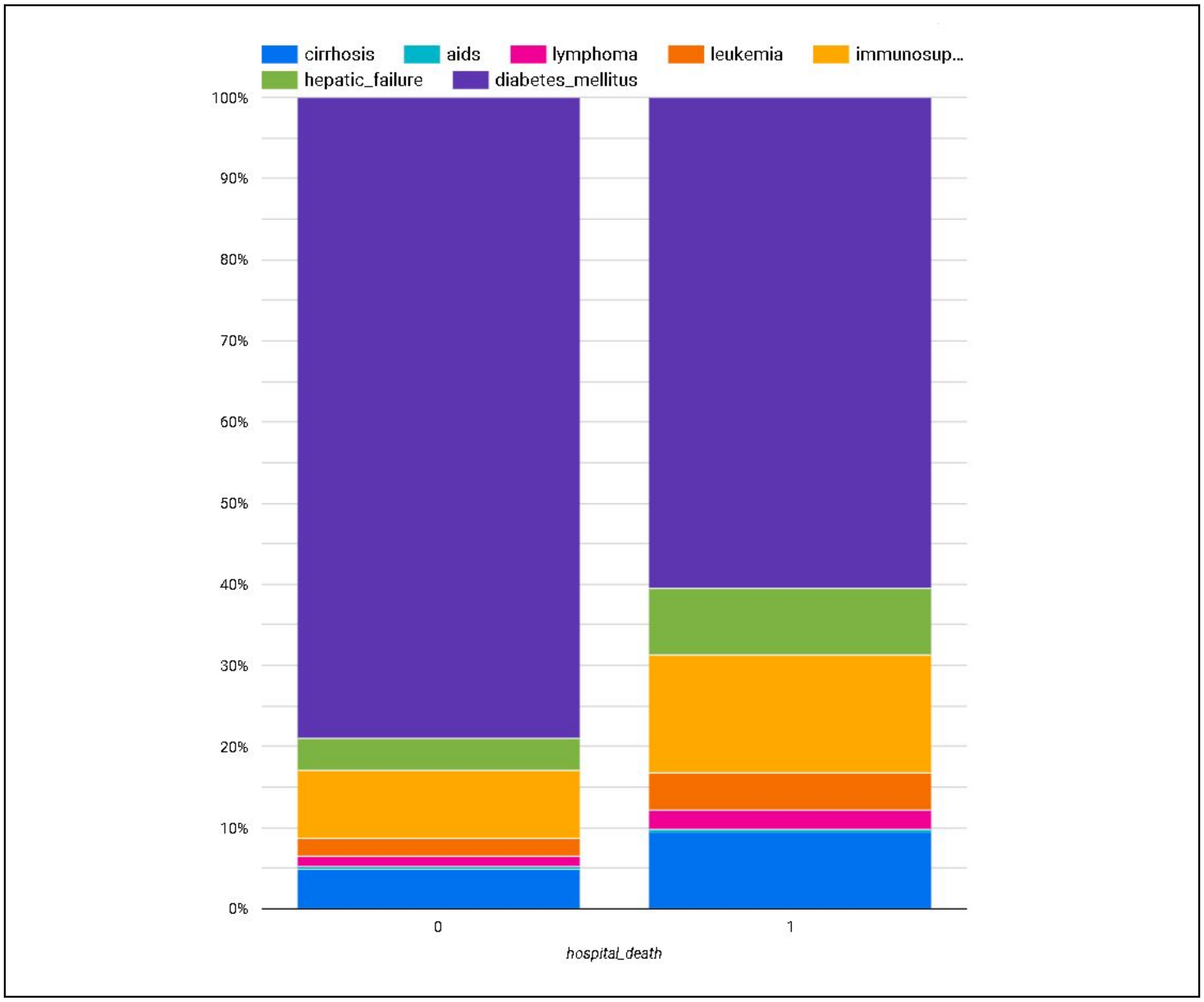


Here we wanted to explore how often these ICU admissions are fatal. The pie Chart shows that the picture is not that grim with only 8.6% cases being fatal.

Value count of number of hospital deaths

0	83798
1	7915

There are 7915 not survived in the training set and 83798 patients survived from the analysis.

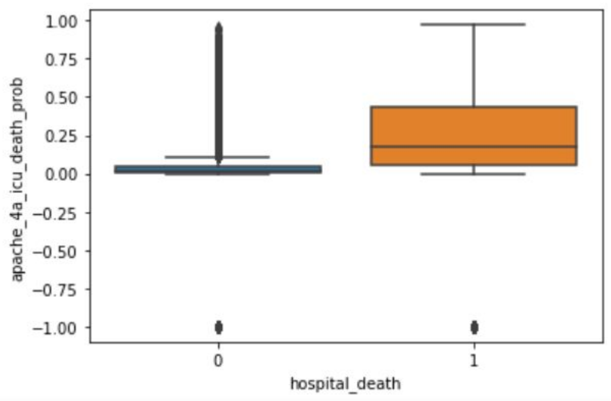


If we further split the data based on different conditions, we see that most of the patients had diabetes. Comparing fatal cases to non fatal cases, we see an interesting pattern - there are different proportions for diseases in the two cases. Even though diabetes accounts for almost 80% of the non-fatal cases, it comprises only 60% for the fatal ones. Proportion for Hepatic Failures almost doubles from 5% to around 10% and for immunosuppression they go from around 10% to 15%.

Box Plots

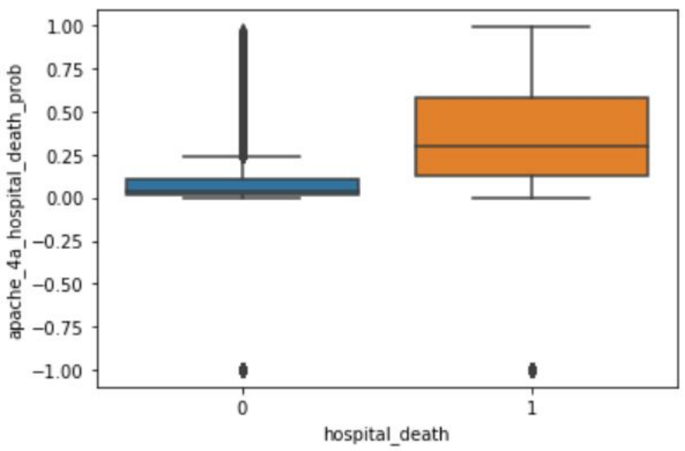
Using the graph, we can compare the range and distribution of various attributes for Survived patients(Hospital_death=0) and Not survived patients(Hospital_death=1).

apache_4a_icu_death_prob



From the above box plot, we can see that the median of apache_4a_icu_death_prob for Survived and Not Survived patients differ which means that hospital death may be dependent on apache_4a_icu_death_prob

Apache_4a_hospital_death_prob

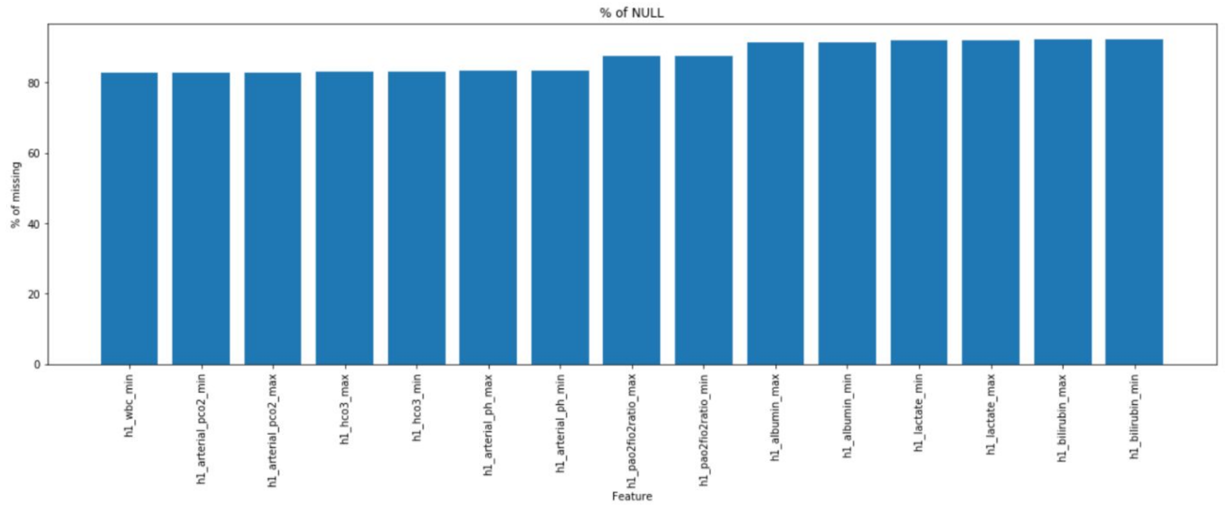


From the above box plot, we can see that the median of apache_4a_hospital_death_prob for Survived and Not Survived patients differ which means that hospital death may be dependent on apache_4a_hospital_death_prob

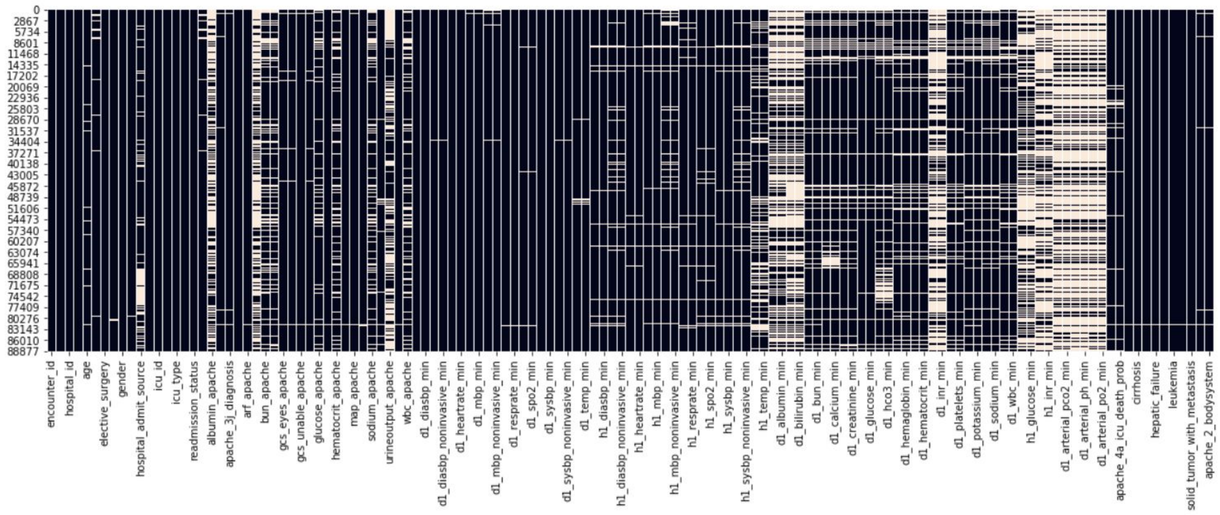
Missing values:

Around 160 variables have missing values out of which 74 variables have more than 50% of missing values.

Some of the variables with highest number of missing values:



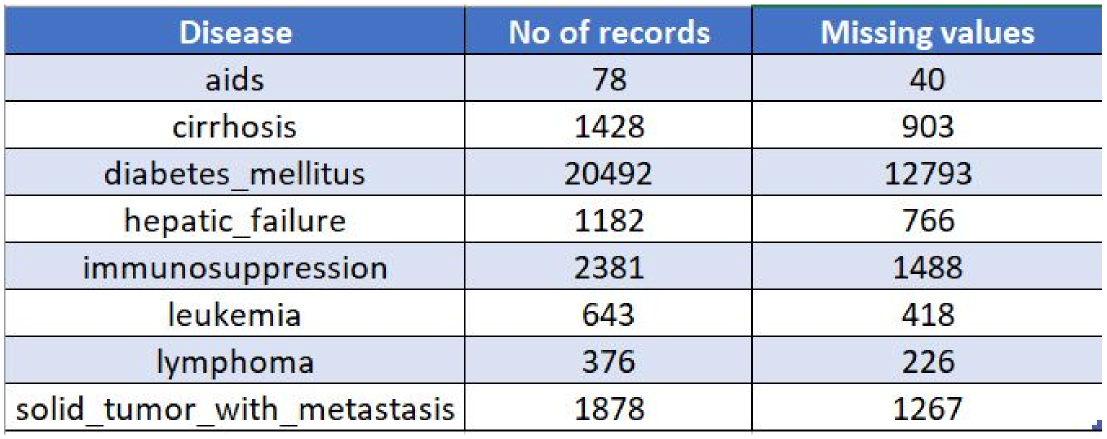
We would need to check if these values are missing at random or if there is a pattern associated with it.



It was observed that most of the values which are missing are min and max of the same variable. Most of the missing values are across vitals, lab gas and lab reports. On further analysis it is seen that most of the missing values taken under the initial hour of admission (h1). Most of the values are available for d1 (24 hours).

Number of missing values more than 50% in the case of initial hour(h1) is 44 while for 24 hrs(d1) is 21.

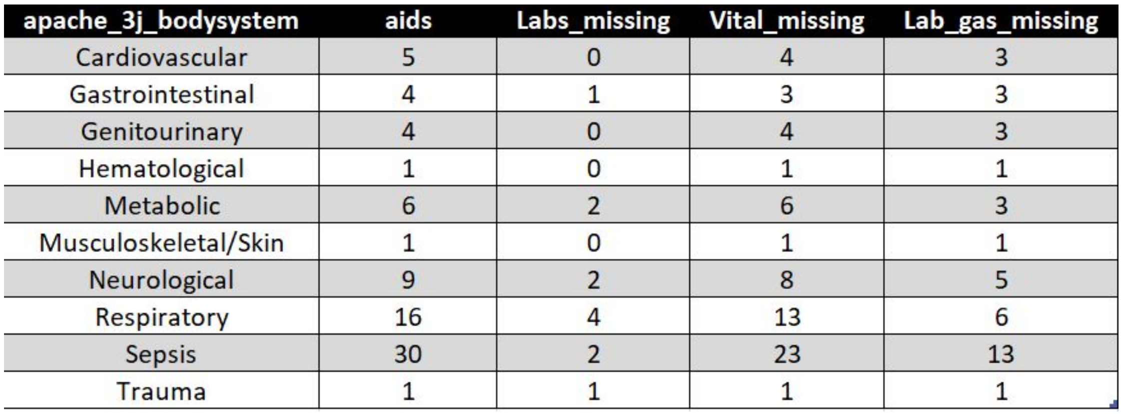
To check if the missing values are pertaining to some particular disease:



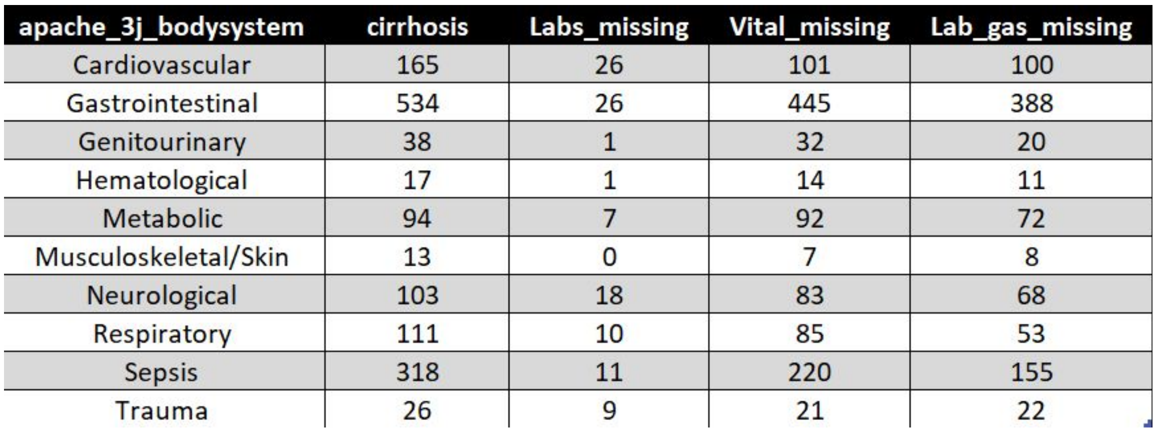
Disease	No of records	Missing values
aids	78	40
cirrhosis	1428	903
diabetes_mellitus	20492	12793
hepatic_failure	1182	766
immunosuppression	2381	1488
leukemia	643	418
lymphoma	376	226
solid_tumor_with_metastasis	1878	1267

The missing values are spread across all the diseases and not biased over a disease

To check if there is any missing pattern across diseases, the variables were grouped based on the category as described in the data dictionary. It is observed that for all the diseases, the vitals and lab records were missing at random.



apache_3j_bodysystem	aids	Labs_missing	Vital_missing	Lab_gas_missing
Cardiovascular	5	0	4	3
Gastrointestinal	4	1	3	3
Genitourinary	4	0	4	3
Hematological	1	0	1	1
Metabolic	6	2	6	3
Musculoskeletal/Skin	1	0	1	1
Neurological	9	2	8	5
Respiratory	16	4	13	6
Sepsis	30	2	23	13
Trauma	1	1	1	1

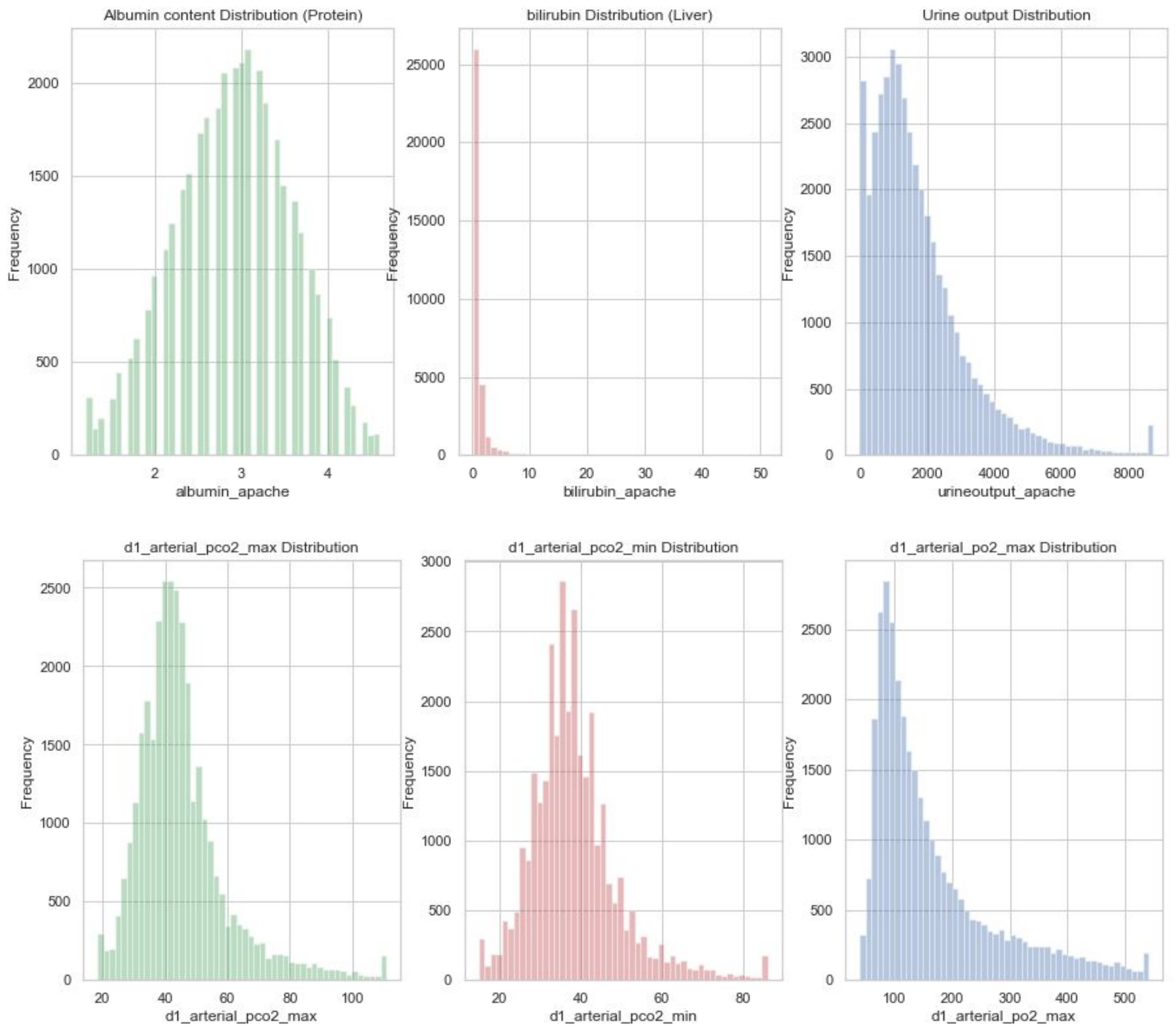


apache_3j_bodysystem	cirrhosis	Labs_missing	Vital_missing	Lab_gas_missing
Cardiovascular	165	26	101	100
Gastrointestinal	534	26	445	388
Genitourinary	38	1	32	20
Hematological	17	1	14	11
Metabolic	94	7	92	72
Musculoskeletal/Skin	13	0	7	8
Neurological	103	18	83	68
Respiratory	111	10	85	53
Sepsis	318	11	220	155
Trauma	26	9	21	22

Imputation:

Since most of the missing values were falling under the initial hour observation(h1) and these values were explained by the values present in d1, the variables missing more than 50% in h1_variables were dropped.

Other variables such as albumin_apache, bilirubin_apache, Distribution of variables were checked before imputation:

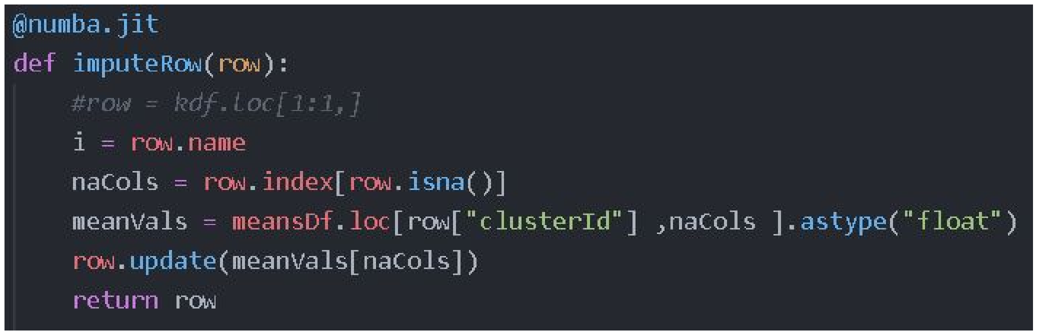


For the initial model Random forest was used for variable imputation.

Other imputation Approaches that were tried were

- **Clustering Imputation**
    - Idea was to cluster observations and then use cluster means to impute the observations

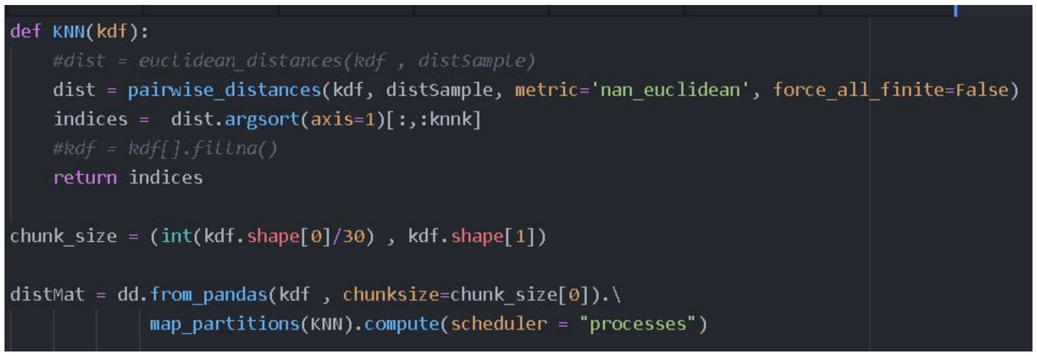
clusters = KMeans(n_clusters = 50).fit(ndf)



```
@numba.jit
def imputeRow(row):
    #row = kdf.loc[1:1,]
    i = row.name
    naCols = row.index[row.isna()]
    meanVals = meansDf.loc[row["clusterId"] ,naCols ].astype("float")
    row.update(meanVals[naCols])
    return row
```

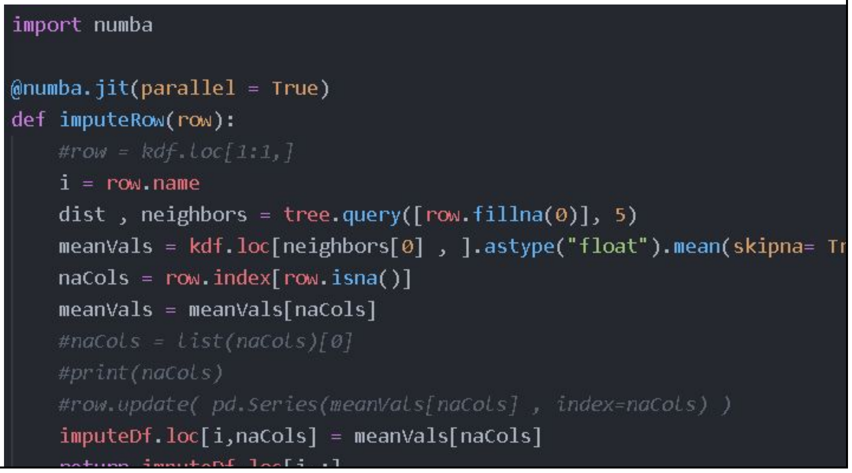
- **Neighbor based Imputation**
  - Following similar line of thought as with clustering we tried to implement K-nn Implementation
  - Since Sklearn's base implementation is all in memory therefore it needed 13.7 gb of space which was prohibitive - therefore we tried a number custom implementation using Dask (parallelization) , Numba (compilation) and KD Trees (approximation)

Dask Based



```
def KNN(kdf):
    #dist = euclidean_distances(kdf , distSample)
    dist = pairwise_distances(kdf, distSample, metric='nan_euclidean', force_all_finite=False)
    indices =  dist.argsort(axis=1)[:,:knnk]
    #kdf = kdf[].fillna()
    return indices

chunk_size = (int(kdf.shape[0]/30) , kdf.shape[1])

distMat = dd.from_pandas(kdf , chunksize=chunk_size[0]).\
            map_partitions(KNN).compute(scheduler = "processes")
```

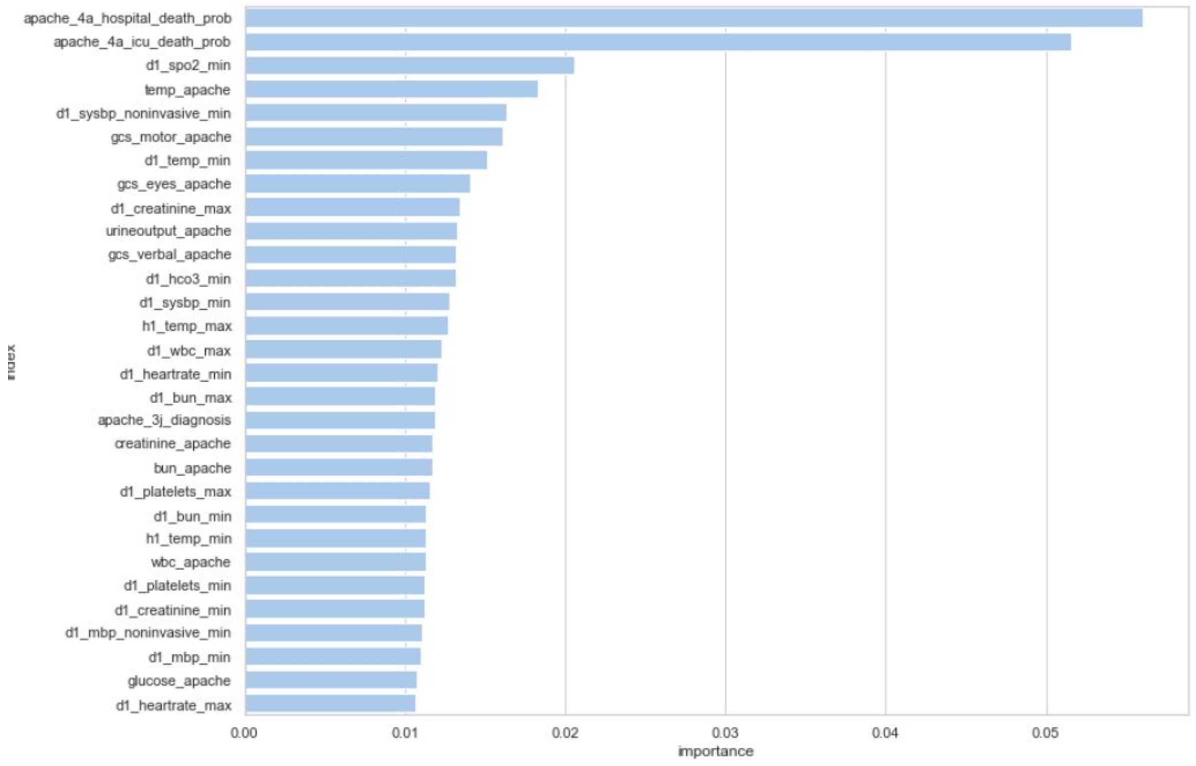
Numba Based



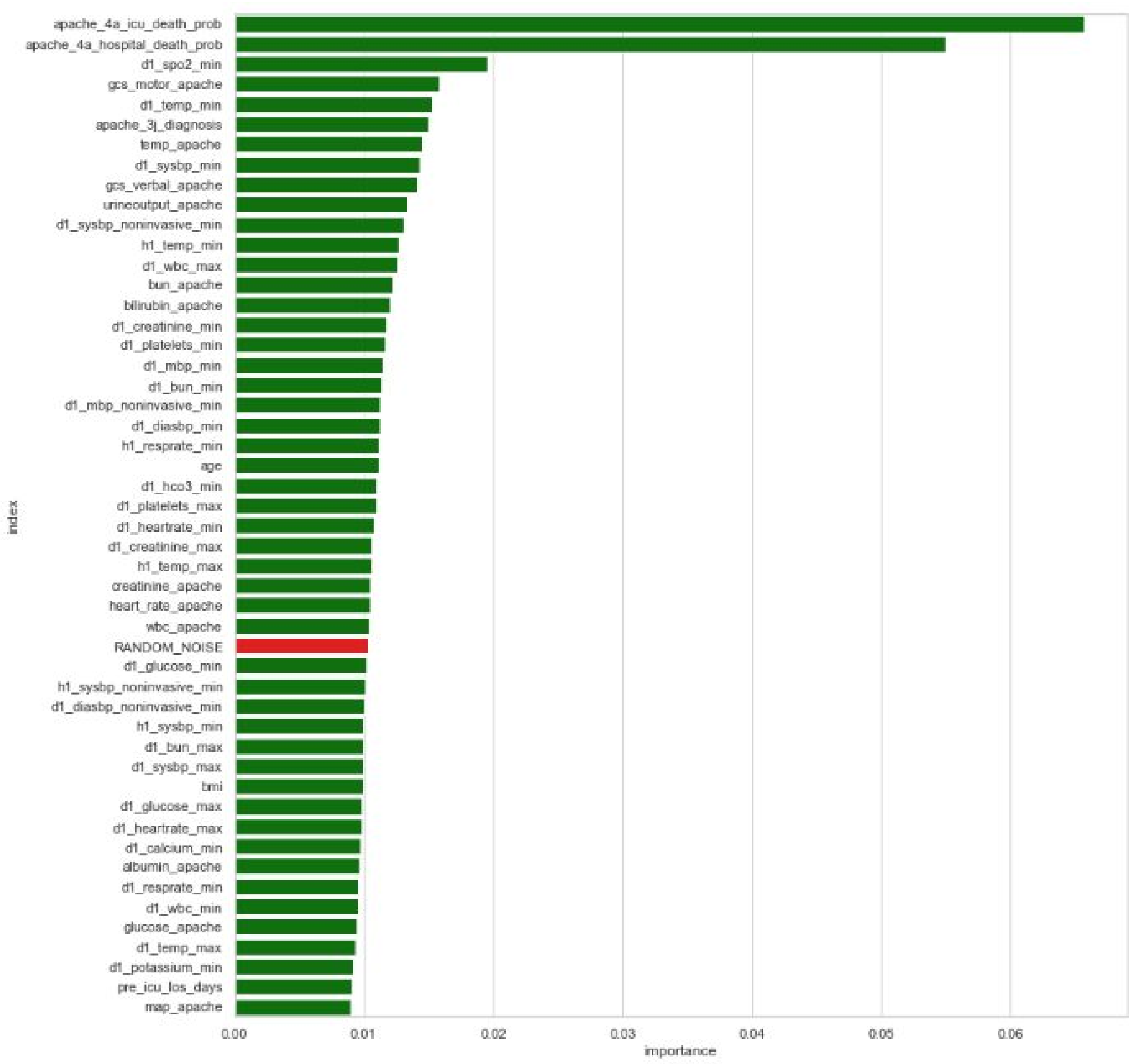
```
import numba

@numba.jit(parallel = True)
def imputeRow(row):
    #row = kdf.loc[1:1,]
    i = row.name
    dist , neighbors = tree.query([row.fillna(0)], 5)
    meanVals = kdf.loc[neighbors[0] , ].astype("float").mean(skipna= Tr
    naCols = row.index[row.isna()]
    meanVals = meanVals[naCols]
    #naCols = list(naCols)[0]
    #print(naCols)
    #row.update( pd.Series(meanVals[naCols] , index=naCols) )
    imputeDf.loc[i,naCols] = meanVals[naCols]
```

Feature selection:

Since there are many variables, Random forest was used to check the importance of features:

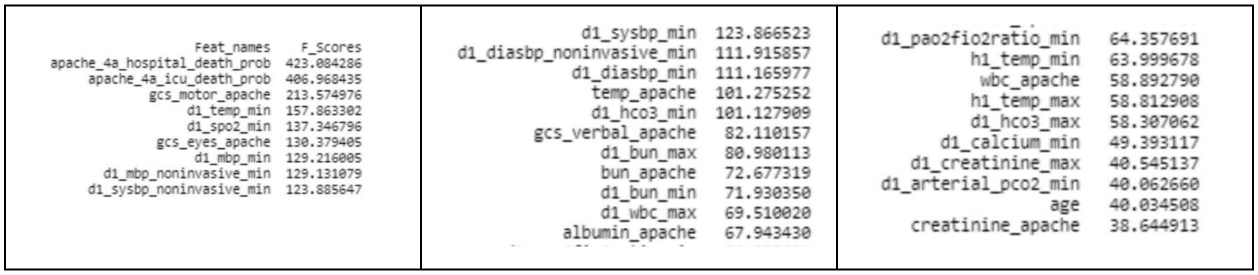


The most important features were apache_4a_hospital_death_prob and apache_4a_hospital_icu_prob.Since most of the feature importance was falling in the range of 1-2%, a random noise was introduced to see how its effect is different from the other variables



From the above chart,It was observed that most of the features have less importance as random noise introduced has more explanatory power than the other features following it.

To check if there is any variation in the feature importances if other techniques were used, ANOVA feature selection was used to compare the results:



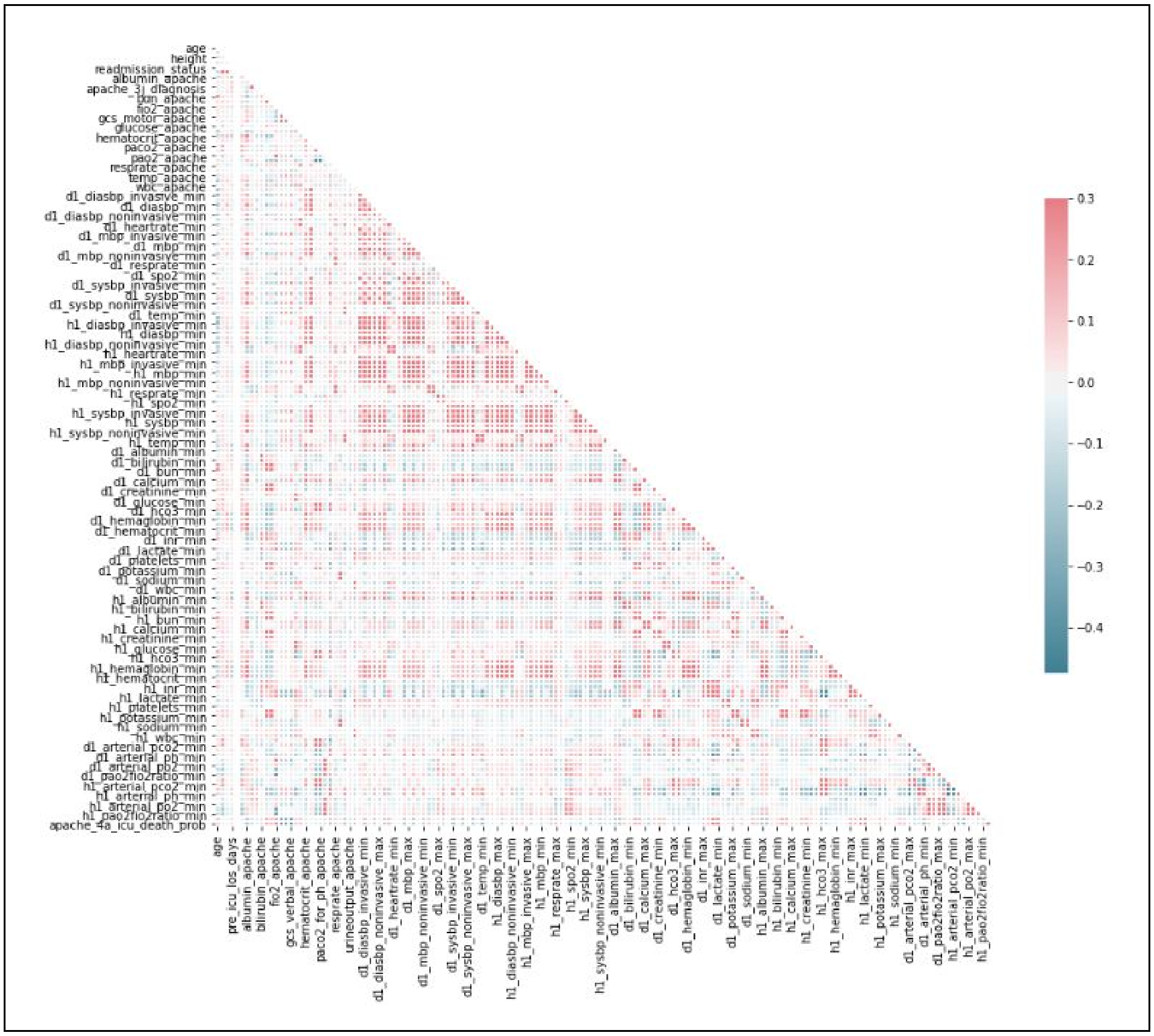
```
                    Feat_names    F_Scores
apache_4a_hospital_death_prob  423.084286
    apache_4a_icu_death_prob  406.968435
            gcs_motor_apache  213.574976
                 d1_temp_min  157.863302
                 d1_spo2_min  137.346796
             gcs_eyes_apache  130.379405
                  d1_mbp_min  129.216005
      d1_mbp_noninvasive_min  129.131079
    d1_sysbp_noninvasive_min  123.885647
```

```
             d1_sysbp_min  123.866523
d1_diasbp_noninvasive_min  111.915857
            d1_diasbp_min  111.165977
              temp_apache  101.275252
              d1_hco3_min  101.127909
        gcs_verbal_apache   82.110157
               d1_bun_max   80.980113
               bun_apache   72.677319
               d1_bun_min   71.930350
               d1_wbc_max   69.510020
           albumin_apache   67.943430
```

```
d1_pao2fio2ratio_min   64.357691
         h1_temp_min   63.999678
         wbc_apache    58.892790
         h1_temp_max   58.812908
         d1_hco3_max   58.307062
      d1_calcium_min   49.393117
   d1_creatinine_max   40.545137
 d1_arterial_pco2_min  40.062660
                 age   40.034508
    creatinine_apache  38.644913
```

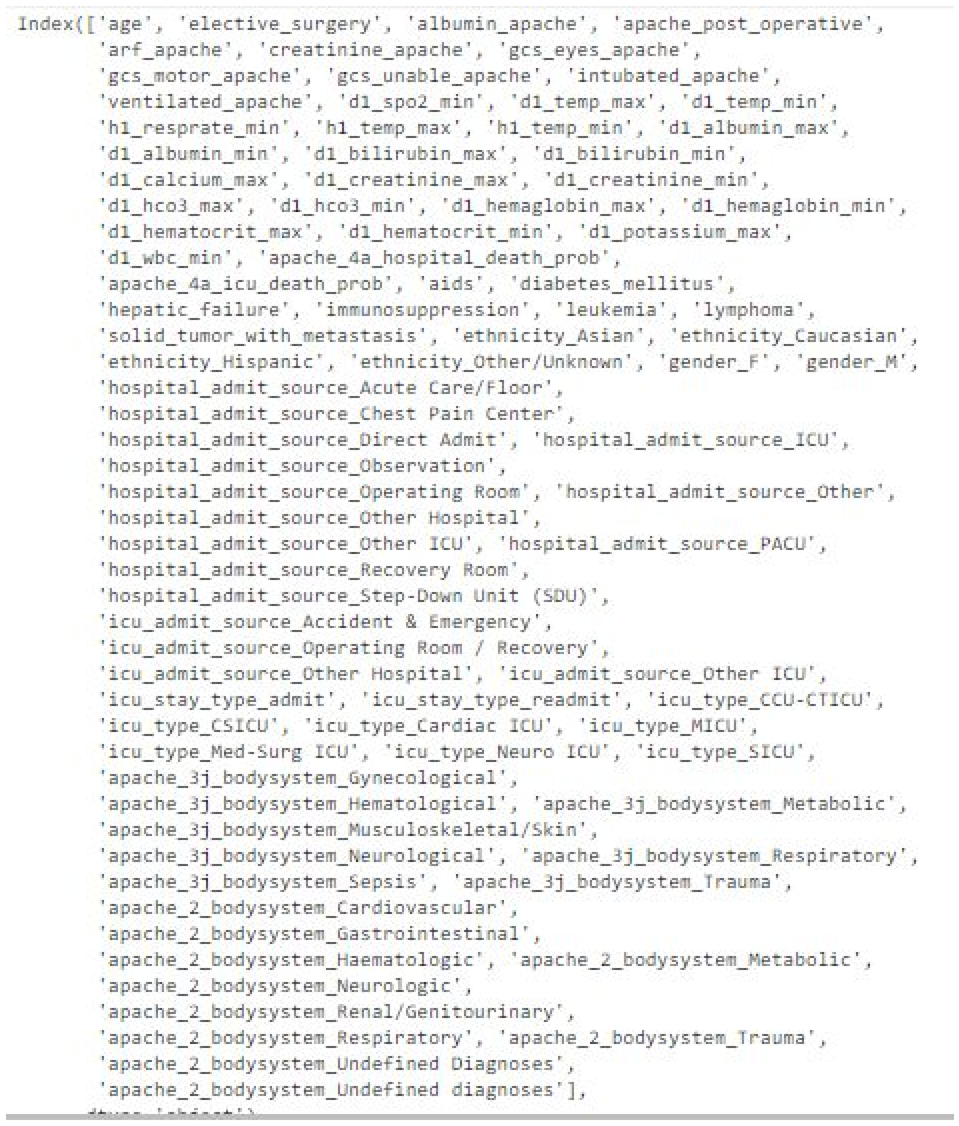
The results were quite similar.

Correlation heatmap



Recursive Feature Elimination

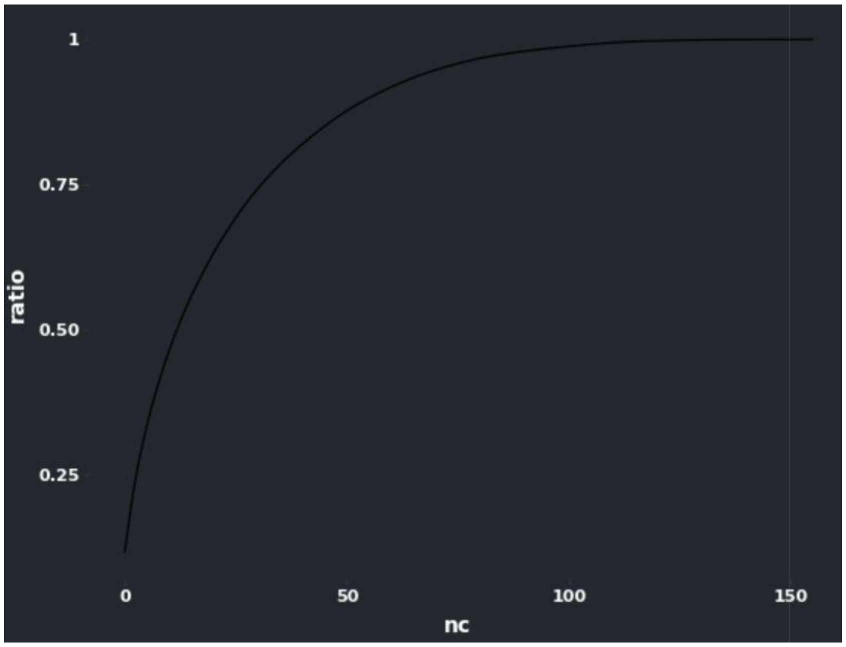
We used RFE with the logistic regression algorithm to select the top 90 features.



```
Index(['age', 'elective_surgery', 'albumin_apache', 'apache_post_operative',
       'arf_apache', 'creatinine_apache', 'gcs_eyes_apache',
       'gcs_motor_apache', 'gcs_unable_apache', 'intubated_apache',
       'ventilated_apache', 'd1_spo2_min', 'd1_temp_max', 'd1_temp_min',
       'h1_resprate_min', 'h1_temp_max', 'h1_temp_min', 'd1_albumin_max',
       'd1_albumin_min', 'd1_bilirubin_max', 'd1_bilirubin_min',
       'd1_calcium_max', 'd1_creatinine_max', 'd1_creatinine_min',
       'd1_hco3_max', 'd1_hco3_min', 'd1_hemaglobin_max', 'd1_hemaglobin_min',
       'd1_hematocrit_max', 'd1_hematocrit_min', 'd1_potassium_max',
       'd1_wbc_min', 'apache_4a_hospital_death_prob',
       'apache_4a_icu_death_prob', 'aids', 'diabetes_mellitus',
       'hepatic_failure', 'immunosuppression', 'leukemia', 'lymphoma',
       'solid_tumor_with_metastasis', 'ethnicity_Asian', 'ethnicity_Caucasian',
       'ethnicity_Hispanic', 'ethnicity_Other/Unknown', 'gender_F', 'gender_M',
       'hospital_admit_source_Acute Care/Floor',
       'hospital_admit_source_Chest Pain Center',
       'hospital_admit_source_Direct Admit', 'hospital_admit_source_ICU',
       'hospital_admit_source_Observation',
       'hospital_admit_source_Operating Room', 'hospital_admit_source_Other',
       'hospital_admit_source_Other Hospital',
       'hospital_admit_source_Other ICU', 'hospital_admit_source_PACU',
       'hospital_admit_source_Recovery Room',
       'hospital_admit_source_Step-Down Unit (SDU)',
       'icu_admit_source_Accident & Emergency',
       'icu_admit_source_Operating Room / Recovery',
       'icu_admit_source_Other Hospital', 'icu_admit_source_Other ICU',
       'icu_stay_type_admit', 'icu_stay_type_readmit', 'icu_type_CCU-CTICU',
       'icu_type_CSICU', 'icu_type_Cardiac ICU', 'icu_type_MICU',
       'icu_type_Med-Surg ICU', 'icu_type_Neuro ICU', 'icu_type_SICU',
       'apache_3j_bodysystem_Gynecological',
       'apache_3j_bodysystem_Hematological', 'apache_3j_bodysystem_Metabolic',
       'apache_3j_bodysystem_Musculoskeletal/Skin',
       'apache_3j_bodysystem_Neurological', 'apache_3j_bodysystem_Respiratory',
       'apache_3j_bodysystem_Sepsis', 'apache_3j_bodysystem_Trauma',
       'apache_2_bodysystem_Cardiovascular',
       'apache_2_bodysystem_Gastrointestinal',
       'apache_2_bodysystem_Haematologic', 'apache_2_bodysystem_Metabolic',
       'apache_2_bodysystem_Neurologic',
       'apache_2_bodysystem_Renal/Genitourinary',
       'apache_2_bodysystem_Respiratory', 'apache_2_bodysystem_Trauma',
       'apache_2_bodysystem_Undefined Diagnoses',
       'apache_2_bodysystem_Undefined diagnoses'],
```

PCA

We also applied PCA to reduce dimensionality. After having imputed and Standard Scaling the numerical features we ran PCA only on the numerical features and the results showed that 100 features out of 157 could explain 100% of the variance.



Baseline Models:

As a baseline model, the imputed data was split into test and train and Logistic Regression was applied to check the model performance.

```
 ---Logistic Regression Model---
Logistic Regression AUC = 0.57
              precision    recall  f1-score   support

           0       0.92      0.99      0.96     25139
           1       0.62      0.14      0.23      2375

    accuracy                           0.92     27514
   macro avg       0.77      0.57      0.59     27514
weighted avg       0.90      0.92      0.89     27514
```

The baseline models AUC is just 0.57 which is quite less.

Since the number of records for hospital_deaths in the dataset is quite less as compared to surviving patients, there is an imbalance in the data. So different sampling techniques were used to see if it improves the model performance.

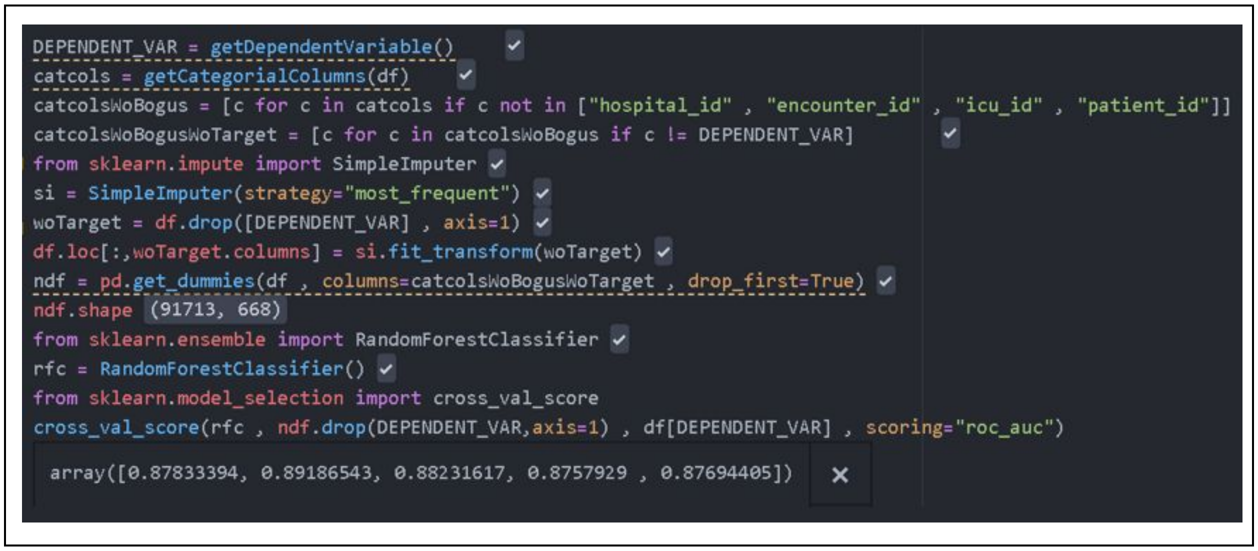
```
The best AUC Score for Original data:
0.8317507863743032
The best AUC Score for Upsampled data:
0.8465680582401258
The best AUC Score for SMOTE data:
0.8592121473886737
The best AUC Score for Downsampled data:
0.8462038473067549
```

When different sampling techniques were used on the data, we can see drastic improvement in results.

Yet Another Baseline

We tried another baseline which

1. Imputed Based on Most Frequent Values
2. Converted Categorical Values to Dummy Variables
3. Dropped ID Columns like Hospital_Id , Encounter_Id
4. Used a Random Forest Classifier
5. This gave us Cross Validated AUC of 0.878 on average



```
DEPENDENT_VAR = getDependentVariable()
catcols = getCategorialColumns(df)
catcolsWoBogus = [c for c in catcols if c not in ["hospital_id" , "encounter_id" , "icu_id" , "patient_id"]]
catcolsWoBogusWoTarget = [c for c in catcolsWoBogus if c != DEPENDENT_VAR]
from sklearn.impute import SimpleImputer
si = SimpleImputer(strategy="most_frequent")
woTarget = df.drop([DEPENDENT_VAR] , axis=1)
df.loc[:,woTarget.columns] = si.fit_transform(woTarget)
ndf = pd.get_dummies(df , columns=catcolsWoBogusWoTarget , drop_first=True)
ndf.shape (91713, 668)
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
from sklearn.model_selection import cross_val_score
cross_val_score(rfc , ndf.drop(DEPENDENT_VAR,axis=1) , df[DEPENDENT_VAR] , scoring="roc_auc")
  array([0.87833394, 0.89186543, 0.88231617, 0.8757929 , 0.87694405])
```

Another Baseline Model - RF and Logistic Regression

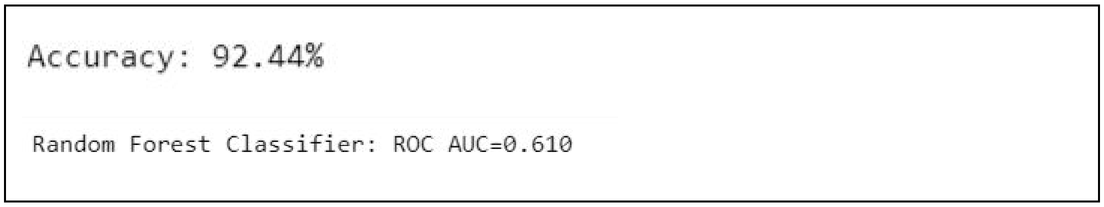
Imputation : Starting with handling missing values, we dropped columns that had more than 60% missing values, from 186 columns, 66 columns were dropped as they had more than 60% missing values, left with 120 columns.

Next, for the columns that had missing values, we imputed using average, max, backward filling and forward filling for different columns.
Since we were started by building a baseline model, we tried by imputing all the numerical columns with average value imputation, and other features with mode values.
Dummy Coding was used for Categorical variables.

We used Random Forest for feature importance, to use only the variables that were significant enough for the model.

Model 1 :

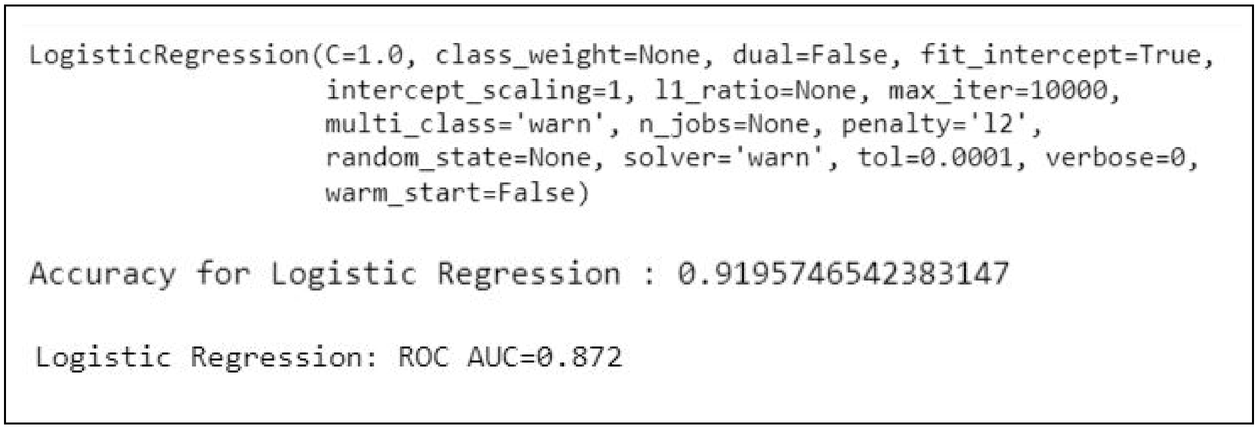
For modelling we first split the data into train and test , then build the model using Random Forest to check the performance however, did not give a good accuracy and AUC.



```
Accuracy: 92.44%

Random Forest Classifier: ROC AUC=0.610
```

Model 2:

Also, trying different models, we build Logistic Regression with similar imputation of average, mean and mode.
We used Random Forest for Feature Engineering i.e importance, below are the results for accuracy and AUC of Logistic Regression.



```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)

Accuracy for Logistic Regression : 0.9195746542383147

Logistic Regression: ROC AUC=0.872
```

Initial Kaggle Submission Model Summary

Pipeline >

- **Missing Values**
    - num_mean = **SimpleImputer**(strategy="mean")
    - cat_freq = **SimpleImputer**(strategy="most_frequent")
- **Add Apache Score column** - getAPACHEScore
- **Feature Transformations**
    - rs = **RobustScaler**()
        - Robust Scaler scales features using statistics that are robust to outliers.
    - pt = **PowerTransformer**()
        - Apply a power transform feature wise to make data more Gaussian-like
- **Feature Reduction**
    - We used Feature agglomeration/ clustering using correlation coefficient as the distance metric
    - fa = **FeatureAgglomeration**(n_clusters=None,affinity="precomputed",compute_full_tree=True, linkage="average" ,distance_threshold=1/corrcoefmin)
- **Categorical Columns**
    - ohe = **OneHotEncoder**(sparse=False , handle_unknown='ignore')
- **Modelling**
    - For the final model we used lightgbm
    - import lightgbm as lgb
    - The optimal parameters found using grid search were

```
params = {
  'max_depth': 10,
  'n_estimators ': 10,
  'objective': 'binary',
  'colsample_bytree': 0.8,
  "class_weight":{0:1 , 1:20},
  "base_score":0.2,
  "n_jobs":-1,
  "metric":"auc",
  "reg_alpha":0.4,
  "Reg_lambda":0.18,
}
```

The final Score we managed to get was 0.8978

Issues Faced:

- Since its healthcare data, **understanding most of the variables** needed spending some time going through the data dictionary.
- The data has many features which makes it **difficult to identify the important features.** Correlation among variables need to be considered while performing variable selection
- **Imbalance in the data -** we have a class imbalance where only 8% of the observations have the target class
- **Handling missing values**: Since there are multiple ways to handle missing data, the right way to handle it which goes in hand with the model selected is tricky.
- **Variable transformation:** Since there are multiple variables and each variable has problems of skewness and outliers associated with it, the right transformation technique would play a major role in model performance
- **Model execution and computational complexity**: Since the data is comparatively large with many features, few algorithms like SVM or grid search with Boosting algorithms would take a long time to execute

Strategies used to encounter

- Different Imputation and Feature selection techniques were tried by each member of the group and the results were compared to identify how its affects the model.
- Different models were tired to check its performance. Further hyperparameter tuning was performed manually to check if there is any improvement in model performance
- Data imbalance: Two approaches were used to handle this. One was upsampling of the data which showed significant improvement in results while the other method was to use Boosting and stacking algorithms which improves the performance by heavily penalizing the weak learners.

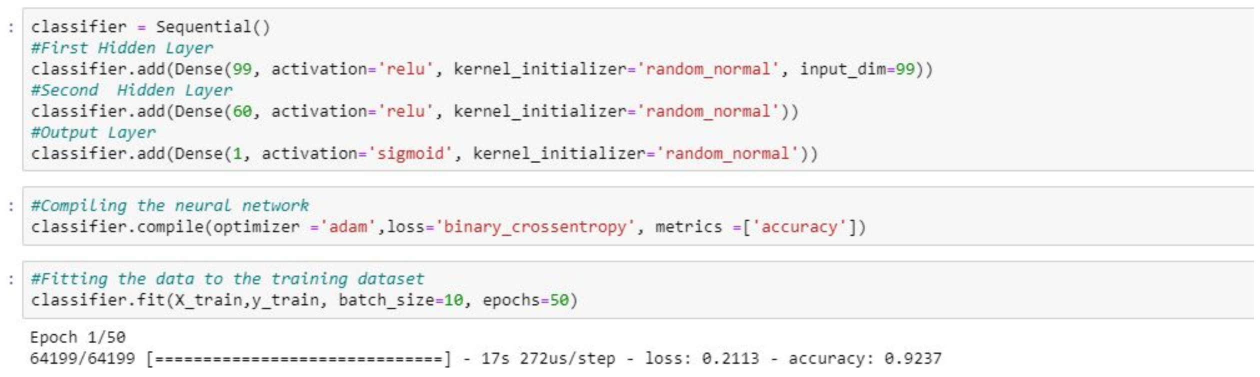
Project Progress - Phase 3:

Model selection and hyperparameter tuning:

Deep learning using KerasClassifier:
(Using different preprocessing techniques)

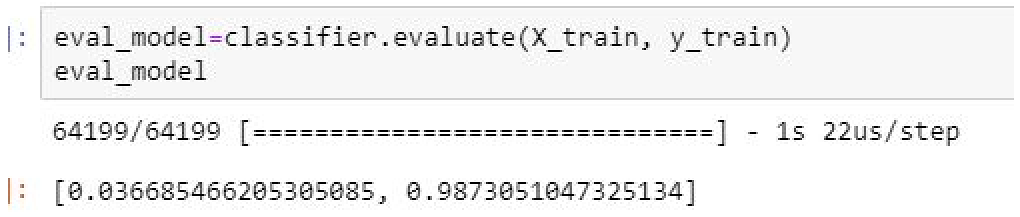
Correlated variables were removed and around 101 variables were left of the imputed data.The variables were scaled and finally, Deep Learning Model using Keras Classifier was implemented on the dataset.

Keras Sequential model was fitted:



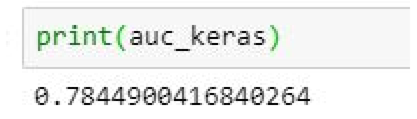
```
: classifier = Sequential()
  #First Hidden Layer
  classifier.add(Dense(99, activation='relu', kernel_initializer='random_normal', input_dim=99))
  #Second  Hidden Layer
  classifier.add(Dense(60, activation='relu', kernel_initializer='random_normal'))
  #Output Layer
  classifier.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))

: #Compiling the neural network
  classifier.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])

: #Fitting the data to the training dataset
  classifier.fit(X_train,y_train, batch_size=10, epochs=50)

  Epoch 1/50
  64199/64199 [==============================] - 17s 272us/step - loss: 0.2113 - accuracy: 0.9237
```

Accuracy = 98.7%



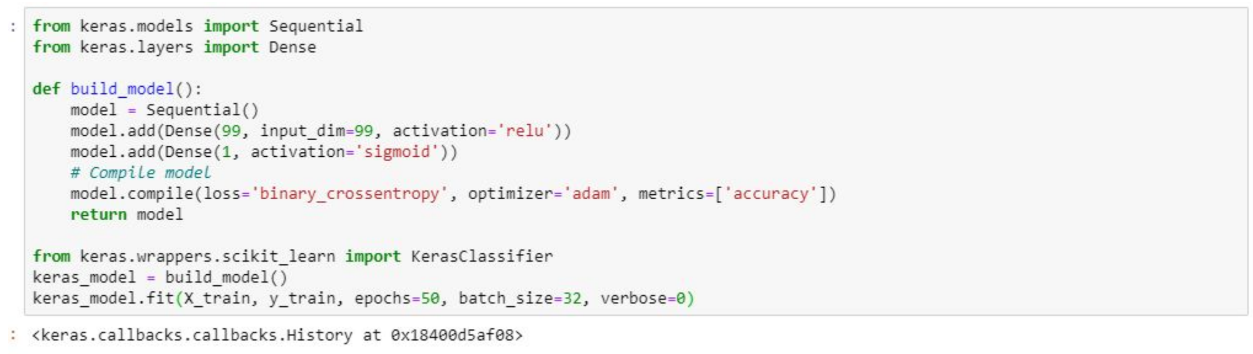
```
]: eval_model=classifier.evaluate(X_train, y_train)
   eval_model

   64199/64199 [==============================] - 1s 22us/step

]: [0.036685466205305085, 0.9873051047325134]
```

AUC = 78.44%

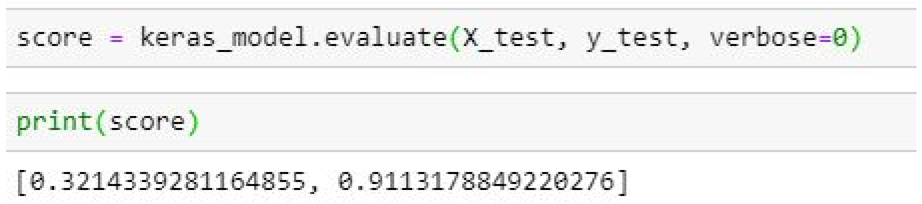


```
print(auc_keras)

0.7844900416840264
```

Different number of hidden layers were tested:

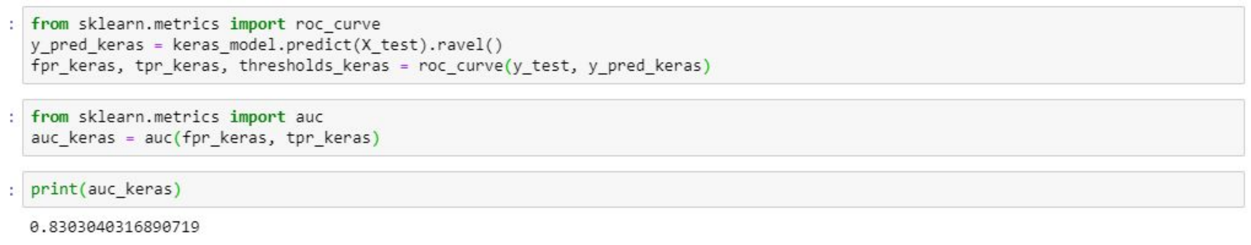


```
: from keras.models import Sequential
  from keras.layers import Dense

  def build_model():
      model = Sequential()
      model.add(Dense(99, input_dim=99, activation='relu'))
      model.add(Dense(1, activation='sigmoid'))
      # Compile model
      model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
      return model

  from keras.wrappers.scikit_learn import KerasClassifier
  keras_model = build_model()
  keras_model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

: <keras.callbacks.callbacks.History at 0x18400d5af08>
```



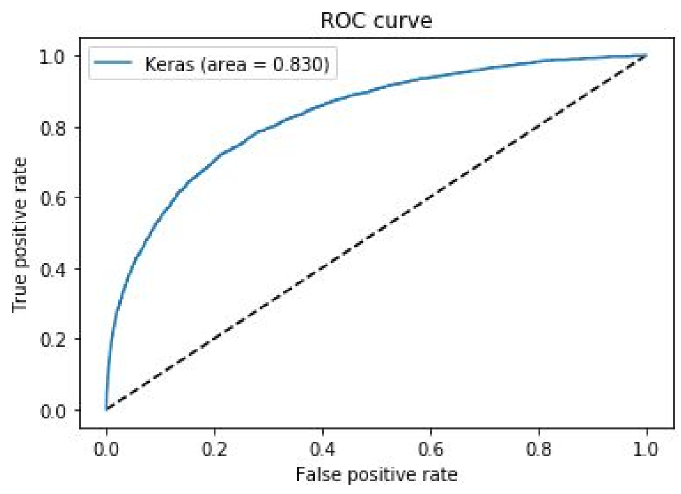
```
score = keras_model.evaluate(X_test, y_test, verbose=0)

print(score)

[0.3214339281164855, 0.9113178849220276]
```

The accuracy obtained was 91% .

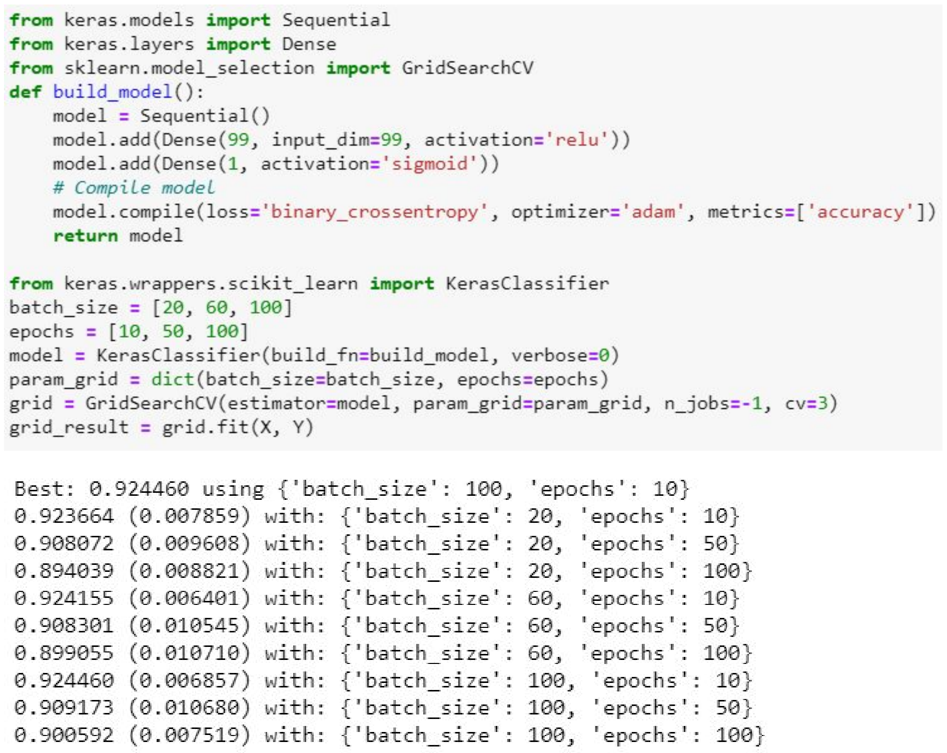


```
: from sklearn.metrics import roc_curve
  y_pred_keras = keras_model.predict(X_test).ravel()
  fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)

: from sklearn.metrics import auc
  auc_keras = auc(fpr_keras, tpr_keras)

: print(auc_keras)

  0.8303040316890719
```

AUC obtained was 83.03%



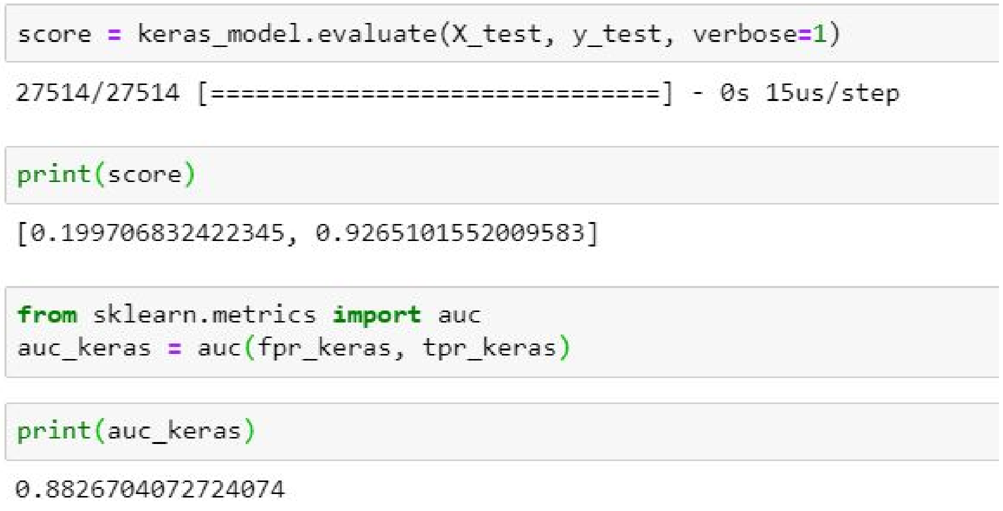
Using GridSearchCV to find the best parameters for Keras Classifier



```python
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import GridSearchCV
def build_model():
    model = Sequential()
    model.add(Dense(99, input_dim=99, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

from keras.wrappers.scikit_learn import KerasClassifier
batch_size = [20, 60, 100]
epochs = [10, 50, 100]
model = KerasClassifier(build_fn=build_model, verbose=0)
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, Y)
```

```
Best: 0.924460 using {'batch_size': 100, 'epochs': 10}
0.923664 (0.007859) with: {'batch_size': 20, 'epochs': 10}
0.908072 (0.009608) with: {'batch_size': 20, 'epochs': 50}
0.894039 (0.008821) with: {'batch_size': 20, 'epochs': 100}
0.924155 (0.006401) with: {'batch_size': 60, 'epochs': 10}
0.908301 (0.010545) with: {'batch_size': 60, 'epochs': 50}
0.899055 (0.010710) with: {'batch_size': 60, 'epochs': 100}
0.924460 (0.006857) with: {'batch_size': 100, 'epochs': 10}
0.909173 (0.010680) with: {'batch_size': 100, 'epochs': 50}
0.900592 (0.007519) with: {'batch_size': 100, 'epochs': 100}
```

After getting the best parameters we got an Accuracy of 92.45 and AUC = 88.26



```python
score = keras_model.evaluate(X_test, y_test, verbose=1)
```

```
27514/27514 [==============================] - 0s 15us/step
```

```python
print(score)
```

```
[0.199706832422345, 0.9265101552009583]
```

```python
from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)
```

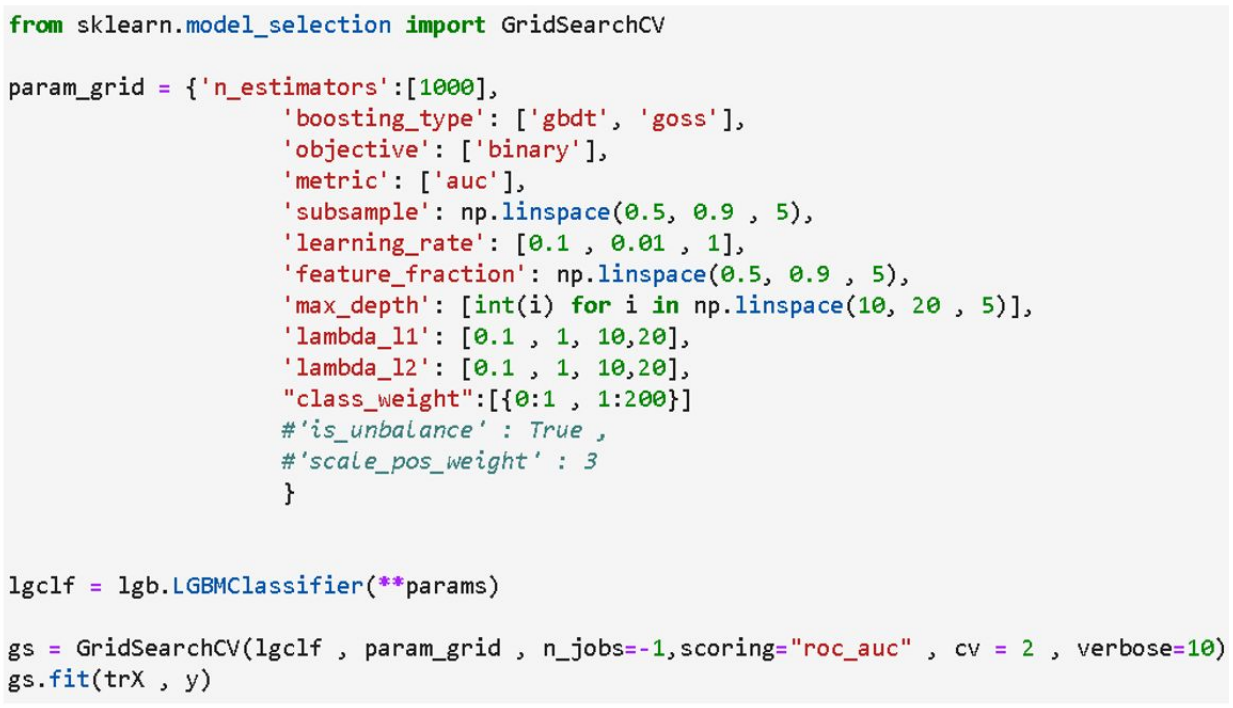
```python
print(auc_keras)
```

```
0.8826704072724074
```

Using Cloud:

Hyperparameter tuning and genetic algorithms were tested

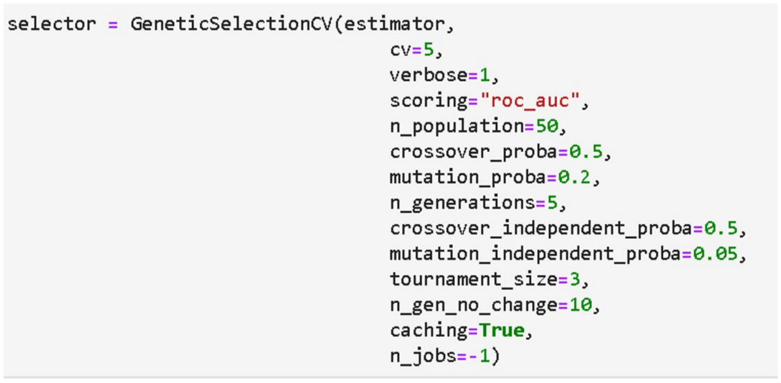
- Tuning of LGB model



```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators':[1000],
                    'boosting_type': ['gbdt', 'goss'],
                    'objective': ['binary'],
                    'metric': ['auc'],
                    'subsample': np.linspace(0.5, 0.9 , 5),
                    'learning_rate': [0.1 , 0.01 , 1],
                    'feature_fraction': np.linspace(0.5, 0.9 , 5),
                    'max_depth': [int(i) for i in np.linspace(10, 20 , 5)],
                    'lambda_l1': [0.1 , 1, 10,20],
                    'lambda_l2': [0.1 , 1, 10,20],
                    "class_weight":[{0:1 , 1:200}]
                    #'is_unbalance' : True ,
                    #'scale_pos_weight' : 3
                    }


lgclf = lgb.LGBMClassifier(**params)

gs = GridSearchCV(lgclf , param_grid , n_jobs=-1,scoring="roc_auc" , cv = 2 , verbose=10)
gs.fit(trX , y)
```

Genetic Algorithms:

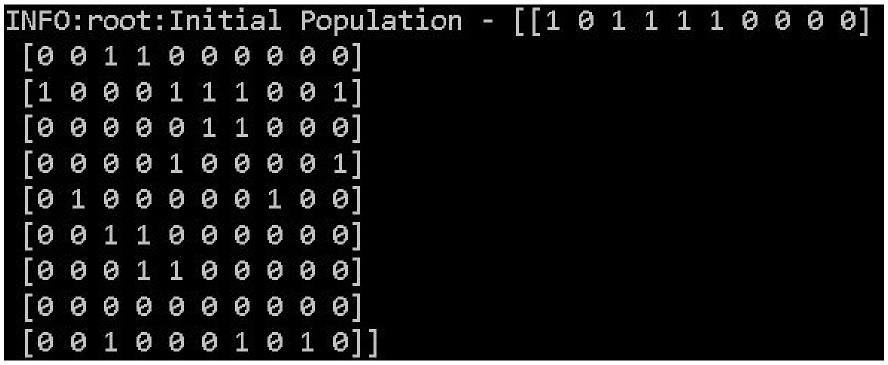
- Also genetic algorithms were used for feature selection to see if they can find a feature subset that can improve the performance



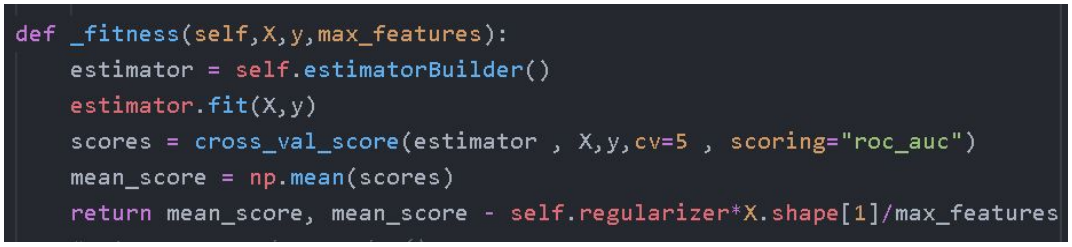
```
selector = GeneticSelectionCV(estimator,
                              cv=5,
                              verbose=1,
                              scoring="roc_auc",
                              n_population=50,
                              crossover_proba=0.5,
                              mutation_proba=0.2,
                              n_generations=5,
                              crossover_independent_proba=0.5,
                              mutation_independent_proba=0.05,
                              tournament_size=3,
                              n_gen_no_change=10,
                              caching=True,
                              n_jobs=-1)
```

Even own implementation of genetic algorithm was tested

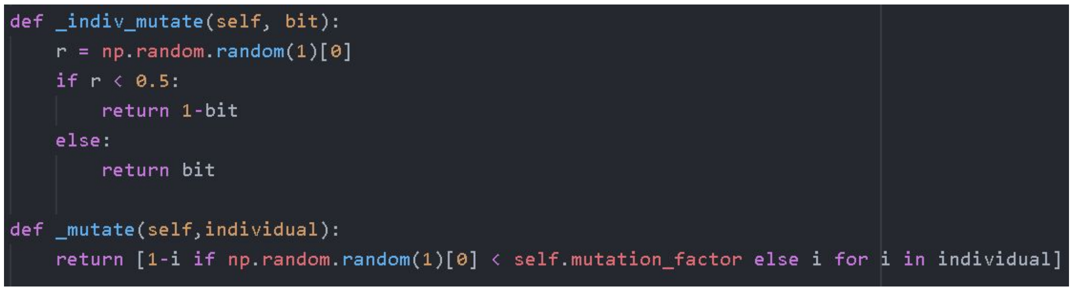
- Each individual in the population is a feature subset
- **Initially the population** consists of random individuals

- 

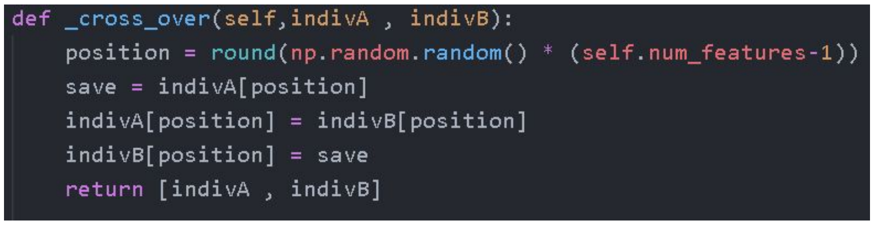
```
INFO:root:Initial Population - [[1 0 1 1 1 1 0 0 0 0]
 [0 0 1 1 0 0 0 0 0 0]
 [1 0 0 0 1 1 1 0 0 1]
 [0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 1 0 0 0 0 1]
 [0 1 0 0 0 0 0 1 0 0]
 [0 0 1 1 0 0 0 0 0 0]
 [0 0 0 1 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 1 0 1 0]]
```

    - (results from a dummy problem with only 10 features with 5 informative features)
- Then these individuals are evaluated with a fitness score
- Most fit individuals are mutated and crossed over and the algorithm repeats for a number of epochs
- **Fitness function** looks like
  - 

```
def _fitness(self,X,y,max_features):
    estimator = self.estimatorBuilder()
    estimator.fit(X,y)
    scores = cross_val_score(estimator , X,y,cv=5 , scoring="roc_auc")
    mean_score = np.mean(scores)
    return mean_score, mean_score - self.regularizer*X.shape[1]/max_features
```

  - Basically a wrapper approach where any estimator can be used
- **Mutation function**
  - 

```
def _indiv_mutate(self, bit):
    r = np.random.random(1)[0]
    if r < 0.5:
        return 1-bit
    else:
        return bit

def _mutate(self,individual):
    return [1-i if np.random.random(1)[0] < self.mutation_factor else i for i in individual]
```

    - Essentially the features are randomly selected/deselected based on self.mutation_factor -[0,1]
- **Cross Over function**
  - 

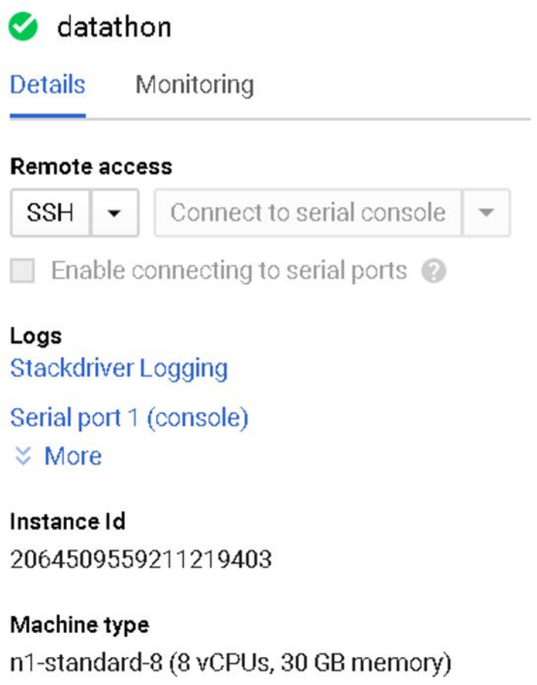
```
def _cross_over(self,indivA , indivB):
    position = round(np.random.random() * (self.num_features-1))
    save = indivA[position]
    indivA[position] = indivB[position]
    indivB[position] = save
    return [indivA , indivB]
```

    - Basically takes two feature subsets and crosses them over at a randomly selected position

- Results on the dummy problems were good enough to warrant trial on the datathon dataset

Since there are a large number of computations needed for both hyperparameter tuning and for genetic algorithms,a cloud machine that is “powerful” enough with 16 vcores and 30gigs of RAM was created. GPUs were not available as apparently google cloud is extensively being utilized in coronavirus research.

Specs of the VM-



Remote access

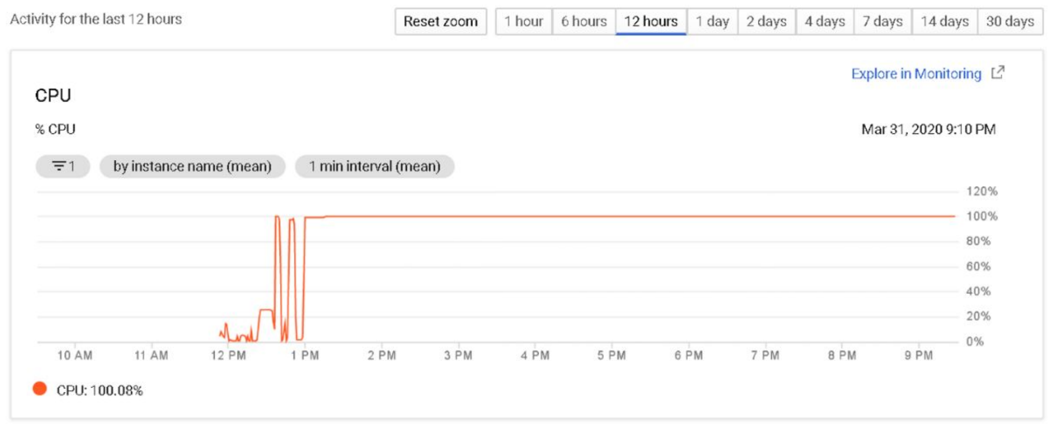
Enable connecting to serial ports

Logs

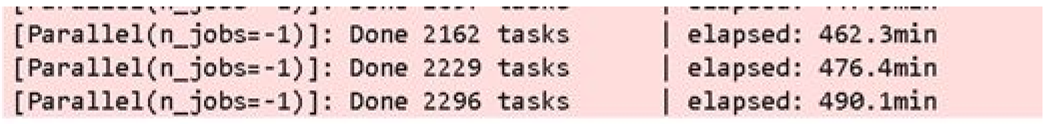
Instance Id
2064509559211219403

Machine type
n1-standard-8 (8 vCPUs, 30 GB memory)

Models were queued to run…..



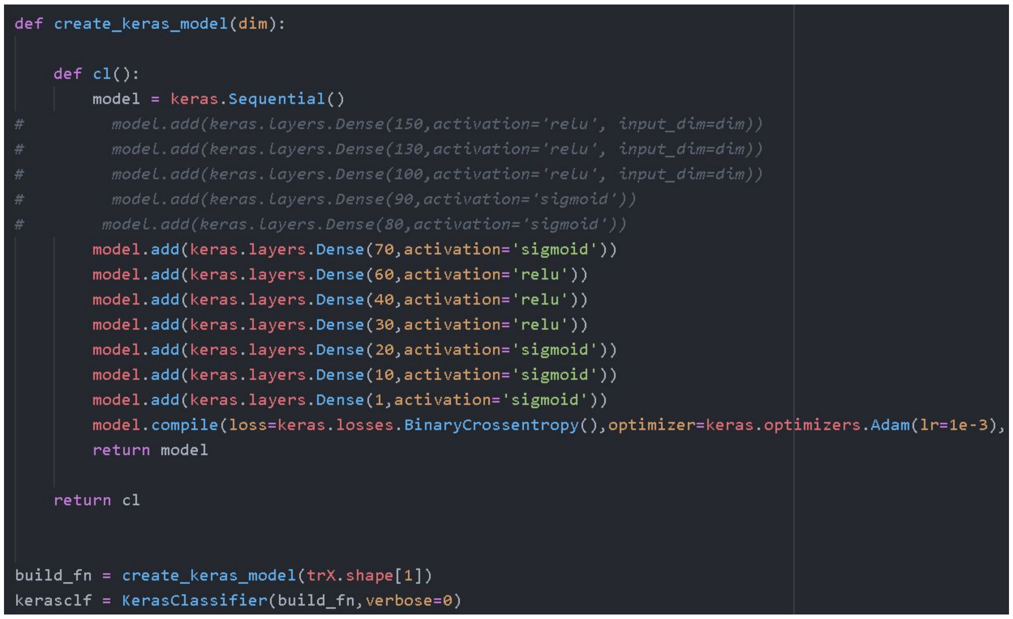
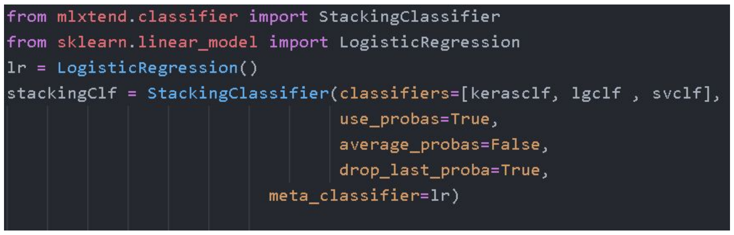
After 8 hours of waiting….
The LGB model had only evaluated 2296 possibilities out of the 12000


```
[Parallel(n_jobs=-1)]: Done 2162 tasks      | elapsed: 462.3min
[Parallel(n_jobs=-1)]: Done 2229 tasks      | elapsed: 476.4min
[Parallel(n_jobs=-1)]: Done 2296 tasks      | elapsed: 490.1min
```

And the genetic algorithm had reached only 32nd generation with avg AUC of 0.882

Due to computational time this approach was abandoned

Deep Learning and Stacking - Using Keras , Mlextend:

a. 
```
def create_keras_model(dim):

    def cl():
        model = keras.Sequential()
#           model.add(keras.layers.Dense(150,activation='relu', input_dim=dim))
#           model.add(keras.layers.Dense(130,activation='relu', input_dim=dim))
#           model.add(keras.layers.Dense(100,activation='relu', input_dim=dim))
#           model.add(keras.layers.Dense(90,activation='sigmoid'))
#          model.add(keras.layers.Dense(80,activation='sigmoid'))
        model.add(keras.layers.Dense(70,activation='sigmoid'))
        model.add(keras.layers.Dense(60,activation='relu'))
        model.add(keras.layers.Dense(40,activation='relu'))
        model.add(keras.layers.Dense(30,activation='relu'))
        model.add(keras.layers.Dense(20,activation='sigmoid'))
        model.add(keras.layers.Dense(10,activation='sigmoid'))
        model.add(keras.layers.Dense(1,activation='sigmoid'))
        model.compile(loss=keras.losses.BinaryCrossentropy(),optimizer=keras.optimizers.Adam(lr=1e-3),
        return model

    return cl


build_fn = create_keras_model(trX.shape[1])
kerasclf = KerasClassifier(build_fn,verbose=0)
```
b. Number of layers and activation units in each layer were varied
c. Even tried stacking Sequential model with Light Gradient Boosting and SVM
    i. 
```
from mlxtend.classifier import StackingClassifier
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
stackingClf = StackingClassifier(classifiers=[kerasclf, lgclf , svclf],
                                 use_probas=True,
                                 average_probas=False,
                                 drop_last_proba=True,
                          meta_classifier=lr)
```

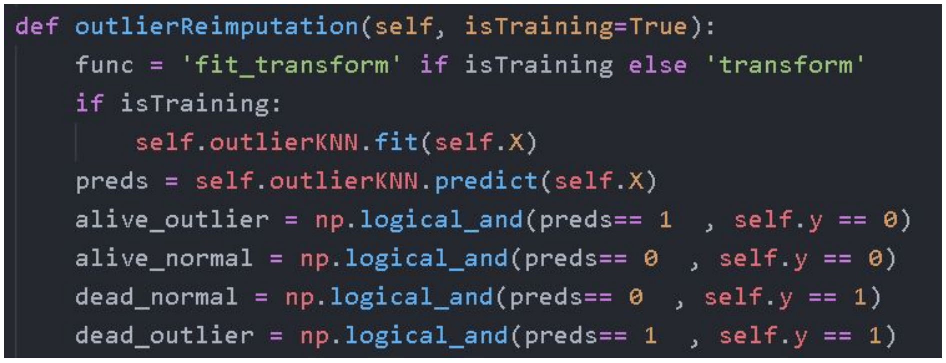
Few other approaches tried:

1. Using Different Categorical Encoding
    a. https://contrib.scikit-learn.org/categorical-encoding/
    b. Weight of Evidence encoding and Target Encoding were tried without any improvement in the AUC ROC score
2. Outlier Detection and Imputation

c. https://pyod.readthedocs.io/en/latest/

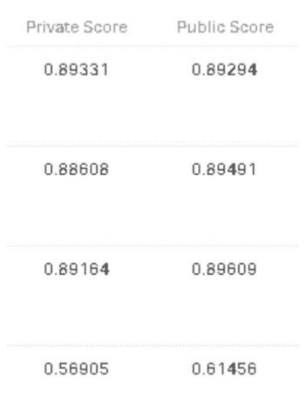
d. K-NN based outlier detection approach was tried

   i. In this approach KNN outlier detection was employed and then 4 groups were created and each group was separately imputed so as not to directly impute with Mean or Most Frequent of the entire dataset, therefore minimizing data distortion

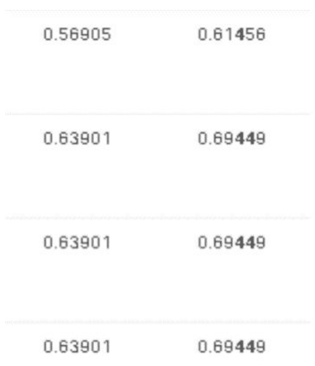
   ii. 

```python
def outlierReimputation(self, isTraining=True):
    func = 'fit_transform' if isTraining else 'transform'
    if isTraining:
        self.outlierKNN.fit(self.X)
    preds = self.outlierKNN.predict(self.X)
    alive_outlier = np.logical_and(preds== 1  , self.y == 0)
    alive_normal = np.logical_and(preds== 0  , self.y == 0)
    dead_normal = np.logical_and(preds== 0  , self.y == 1)
    dead_outlier = np.logical_and(preds== 1  , self.y == 1)
```

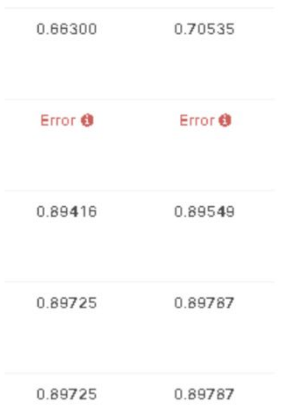
After having tested out most of the possibilities, without any improvement in score, this approach was abandoned.Scores were stuck near 0.89….



Private Score	Public Score
0.89331	0.89294
0.88608	0.89491
0.89164	0.89609
0.56905	0.61456



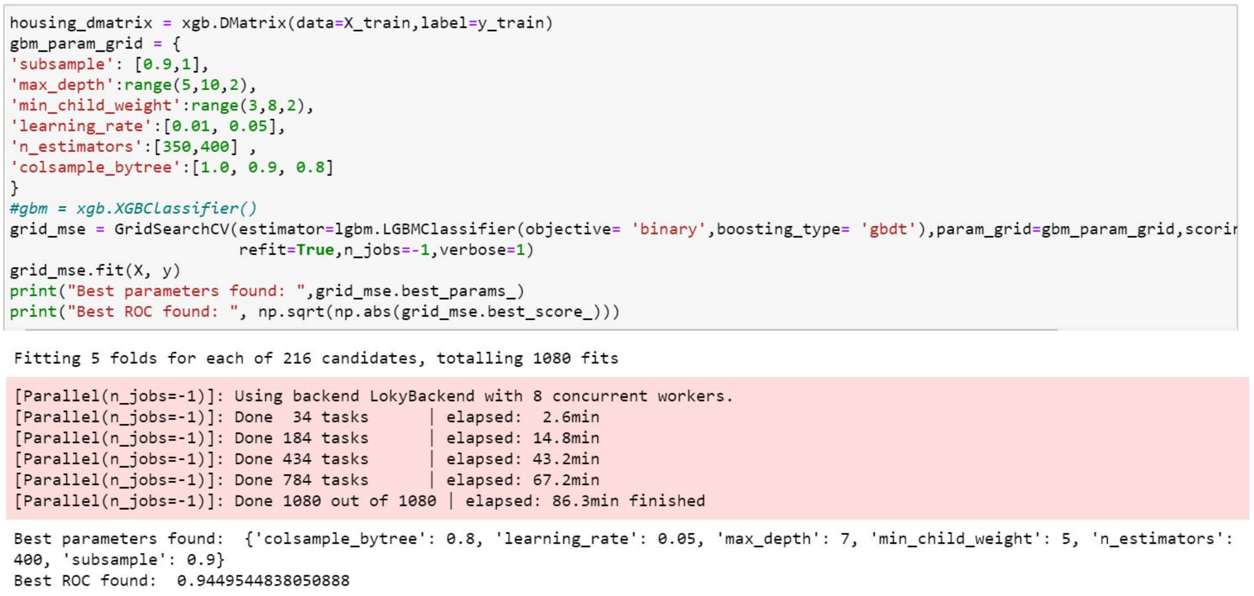
0.56905	0.61456
0.63901	0.69449
0.63901	0.69449
0.63901	0.69449



0.66300	0.70535
Error	Error
0.89416	0.89549
0.89725	0.89787
0.89725	0.89787

The top Kaggle score in the leaderboard is 0.91497. With all the different approaches tested the results obtained by our team is 0.90150.

The best results obtained using boosting algorithms like XGBoost and LightGBM.



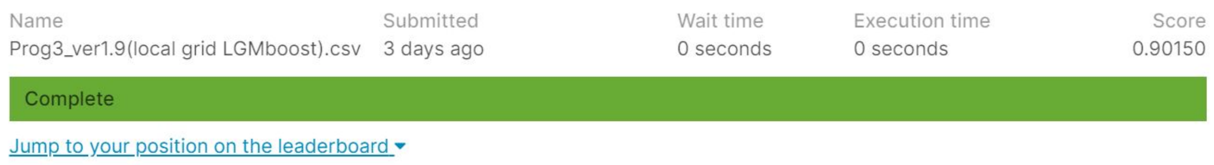
```
housing_dmatrix = xgb.DMatrix(data=X_train,label=y_train)
gbm_param_grid = {
'subsample': [0.9,1],
'max_depth':range(5,10,2),
'min_child_weight':range(3,8,2),
'learning_rate':[0.01, 0.05],
'n_estimators':[350,400] ,
'colsample_bytree':[1.0, 0.9, 0.8]
}
#gbm = xgb.XGBClassifier()
grid_mse = GridSearchCV(estimator=lgbm.LGBMClassifier(objective= 'binary',boosting_type= 'gbdt'),param_grid=gbm_param_grid,scorir
                        refit=True,n_jobs=-1,verbose=1)
grid_mse.fit(X, y)
print("Best parameters found: ",grid_mse.best_params_)
print("Best ROC found: ", np.sqrt(np.abs(grid_mse.best_score_)))
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 14.8min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 43.2min
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed: 67.2min
[Parallel(n_jobs=-1)]: Done 1080 out of 1080 | elapsed: 86.3min finished
```

```
Best parameters found:  {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 5, 'n_estimators':
400, 'subsample': 0.9}
Best ROC found:  0.9449544838050888
```

The accuracy obtained on Kaggle Leaderboard:



Name	Submitted	Wait time	Execution time	Score
Prog3_ver1.9(local grid LGMboost).csv	3 days ago	0 seconds	0 seconds	0.90150

Complete

Jump to your position on the leaderboard

Link to the updated python file:
https://github.com/abhijeetdtu/Datathon/tree/master/Datathon/EDA
Filename: Final model tuning

Key Takeaways

1. Feature Engineering is Key to everything

a. Data issues

With datasets that have so many features as well as compounding data issues, it becomes imperative that we spend considerable efforts in getting the data in the right shape. Not only that but also have to be mindful of the fact that we need a bit of domain knowledge to be able to determine if the values that we are seeing are valid or not, in fact we could be seeing inconsistencies when looking at multiple columns together and resolving such issues, hard as it is, can significantly boost the signal in the data.

b. We don't know what we don't know

We were supposed to get a mentor for the project but didn't and we feel that had we had a couple of sessions with an medical expert we could have identified issues that have clearly escaped our scrutiny.

c. Correlation doesn't mean you drop features

One of the most important takeaway so far has been that correlation doesn't mean that we drop the features. We have to be very careful when removing any data, especially entire columns. Columns that might be correlated with other columns, can reveal new information when combined with other columns. Therefore extra caution has to be taken before deciding to drop data.

d. Missing values are a pain

This particular dataset had so many columns that were almost sparse with data missing completely at random. In such cases it becomes challenging to identify when to impute and when to consider dropping. In some cases you might want to replace the column with a 'isMissing' column. Different strategies work on different columns and nothing seems to be grounded in theory but in empirically testing out different combinations.

e. Common Sense and Occam's Razor

Even though complex transformations are what we usually think of, like power transformers or robust transforms and standard scalers and what not, but it's important to keep in mind Occam's razor and look at the data pragmatically. Some very simple transformations can help boost the signal. One such transformation that worked for us was binning the 'apache_4a_hospital_death_prob' column. This column showed up as the most important feature in our analysis and ranged from 0 - 1 but simple binning of the column into 2 categories helped increase the score by 0.030 which is significant considering that we were stuck in a plateau of 5th decimal place.

2. Hyper-parameter tuning

a. Telescopic Search

We all know that Grid Search is exhaustive search, therefore it becomes crucial that we , as data scientists, know how to limit the search space. Exhaustive search of everything is not possible.

One technique that really helps is telescoping search where we try wildly different combinations in the first pass and then zoom in on the parameter values. This helps in initial elimination of vast search space.

b. Cloud is your new best friend

No matter how powerful your personal computer is, to be able to do Hyper parameter tuning satisfactorily and efficiently it’s wise to use cloud resources. Initial EDA, Feature engineering can be done locally but computationally expensive tuning workload should be taken to cloud.

We extensively used Google Datalab and Google Colab in parameter tuning. Even in cloud we had long running workloads that would take hours before we would get the results but that’s the best bet.

3. Debug your model

When hit with a plateau, instead of randomly tweaking the model, understand why it’s failing. Looking at confusion matrix and learning curves, they will at least enlighten you if the problem is with the model or with the data. Look at misclassified examples and understand why that is happening.

Team Roles and Contributions

	Team Member Name (800 id)	Responsible For
1	Abhijeet - 801155955	EDA using Google Cloud Data Studio,Python Feature engineering -KNN , PCA, RandomForest,Clustering,Genetic Algorithm Exploring Dask and Numba for faster computation GridSearch and Hyperparameter Tuning for LightGBM Github Documentation
2	Ridhi - 801151577	EDA - Python Feature engineering - PCA, Random Forest Model Selection - Logistic Regression, Deep learning - Keras Classifier, Github Documentation
3	Vivek - 801151371	EDA-Python Feature engineering - ANOVA and Random Forest Model - Logistic Regression, XGBoost, Random Forest Github Documentation
4	Mansi - 801136257	EDA using Google Cloud Data Studio, Python Feature engineering - Recursive Feature Elimination, Model Selection - Random Forest , XGboost, Keras Classifier, GridSearchCV Github Documentation

References

1. Johnson AEW, Mark RG. Real-time mortality prediction in the Intensive Care Unit. *AMIA Annu Symp Proc*. 2018;2017:994–1003. Published 2018 Apr 16.
2. Bouvy ML, Heerdink ER, Leufkens HGM*, et al*Predicting mortality in patients with heart failure: a pragmatic approach*Heart* 2003;**89:**605-609.
3. C E Douma, W K Redekop, J H van der Meulen, R W van Olden, J Haeck, D G Struijk and R T Krediet JASN January 1997, 8 (1) 111-117;
4. Cannon, Chad M., et al. "Utility of the shock index in predicting mortality in traumatically injured patients." *Journal of Trauma and Acute Care Surgery* 67.6 (2009): 1426-1430.
5. A clinical prediction model to identify patients at high risk of death in the emergency department : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4477719/
6. Emergency department triage prediction of clinical outcomes using machine learning models:https://ccforum.biomedcentral.com/articles/10.1186/s13054-019-2351-7