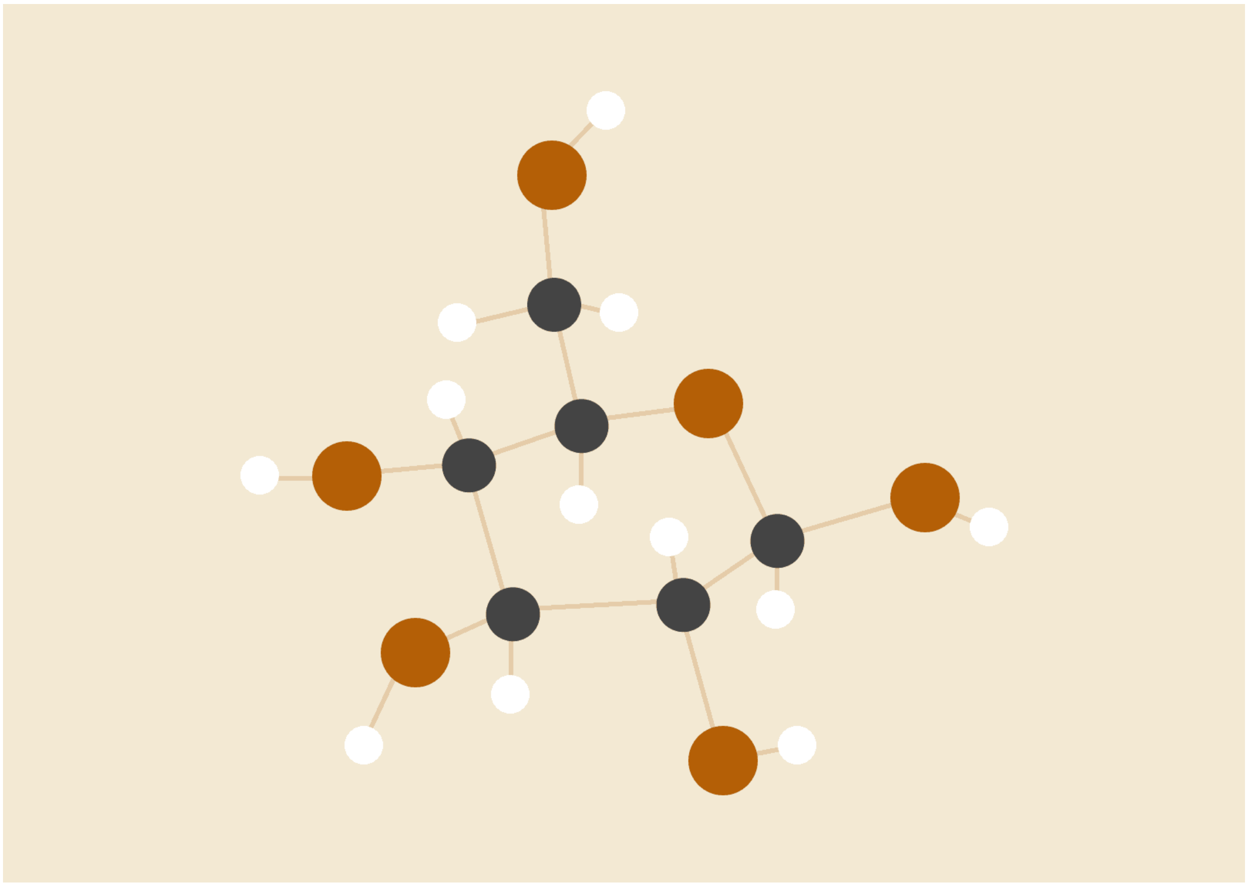
Hospital Length of Stay Prediction

EED363 – Applied Machine Learning (Spring 2021)

Final Project Report



SANSKAR TEWATIA - 1810110215

MILIND SONI - 1810110131

Group - 14

CONTENTS

INTRODUCTION

"The goal of this project is to create a model that predicts the length-of-stay for each patient at time of admission."

Every time a patient is admitted to a hospital, the first question that comes to the mind is the number of days the patient will be admitted. This information is of utmost importance to the hospital, because they need to allocate resources accordingly, and plan future admittances based on the number of free beds. If there is one thing we learnt from the Pandemic, it's the fact that there can be situations where a huge chunk of the population needs to be hospitalized simultaneously. In such cases, there is a need to predict the hospital capacity weeks in advance, so as to provide proper care to patients and refer new patients to another hospital in case there are no beds free.

There can be significant variation of LOS across various facilities and across disease conditions and specialties even within the same healthcare system. Advanced LOS prediction at the time of admission can greatly enhance the quality of care as well as operational workload efficiency and help with accurate planning for discharges, resulting in lowering of various other quality measures such as readmissions. Using various machine learning models, we will try to make predictions on the number of days a patient is going to be admitted to a hospital. These models will take into account various types of personal information about the patient like age, marital status, etc and also their past history of medical conditions like asthma, depression, malnutrition, pneumonia, etc.

DATASET

We have chosen the MIMIC III dataset released by MIT. The dataset consists of 48 features, 1 target column - Length of Stay, and 51,037 rows of such information. It has a robust amount of information and is described as

“ An openly available dataset developed by the MIT Lab for Computational Physiology, comprising de-identified health data associated with ~60,000 intensive care unit admissions. It includes demographics, vital signs, laboratory tests, medications, and more.”

Link to the dataset:
https://github.com/sdasara95/Machine-Learning-in-Healthcare/tree/master/data

DATA EXPLORATION

NULL VALUES -

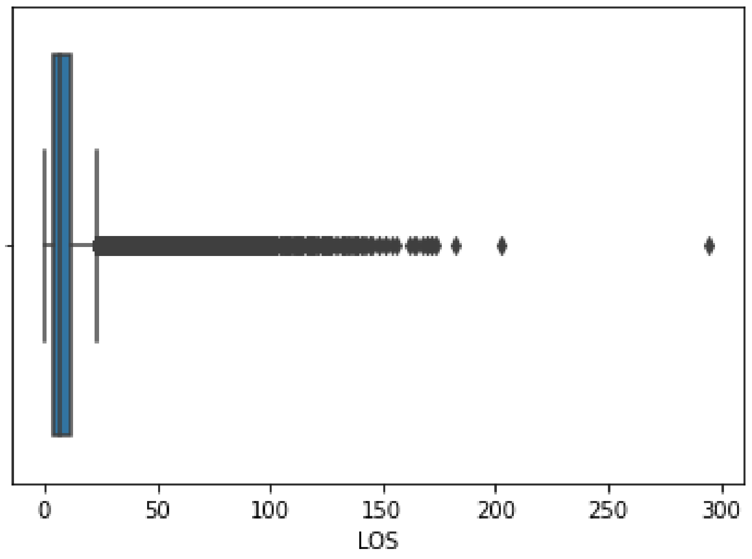
First we find out the number of null values in the entire dataset. Since there are no null values in the dataset, there is no need to fill any missing information.

```
df.isnull().values.sum()
```

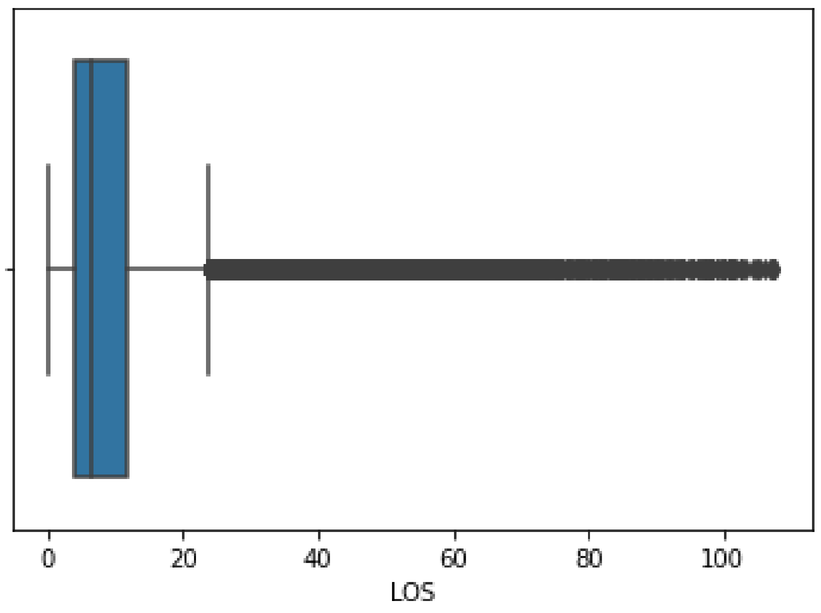
```
0
```

OUTLIER VISUALIZATION AND HANDLING -

Next we plot a boxplot of the target variable LOS in order to see its distribution on the number line. From the plot we can see that the majority of the values of the target variable are concentrated in the 10-15 day range, and that there are also a few outliers whose values can negatively affect the training of the model.



Hence we take only the 99.8 Percentile of the dataset and again print the boxplot. It can be seen that the majority of the outliers have been removed from the entire dataset.



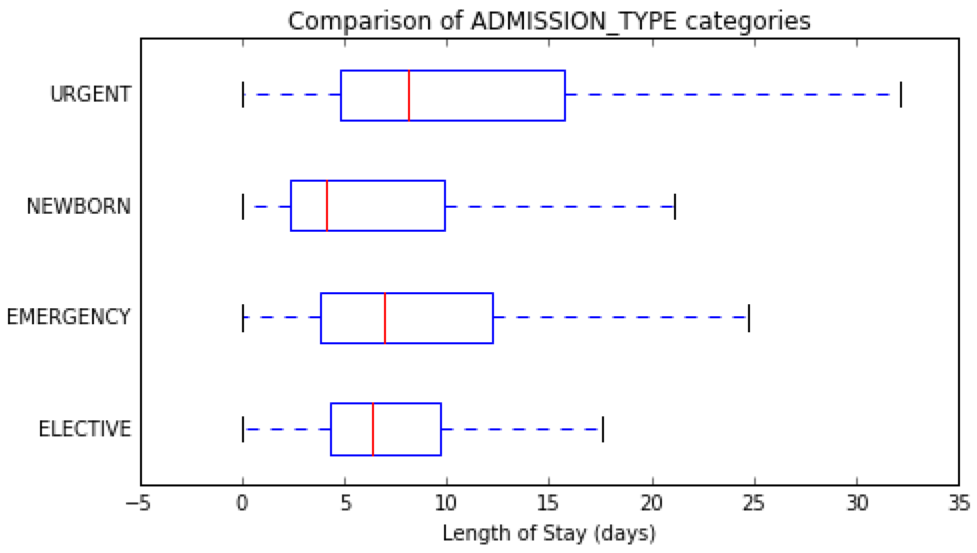
STATISTICS ABOUT TARGET VARIABLE -

We now find out some statistics about the target variable LOS - Length of Stay to get an idea about the number of days people get admitted to the hospitals, and also to get gain statistical insights about the target variable -

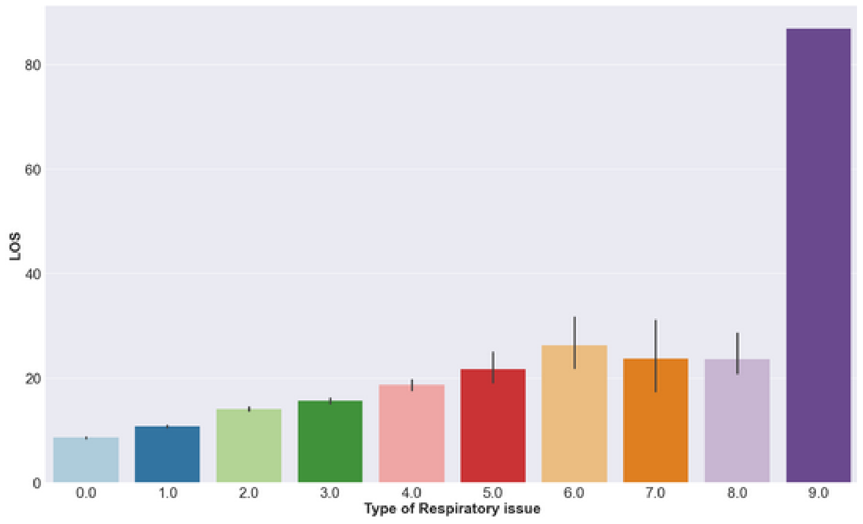
```
count    50934.000000
mean         9.982970
std         11.156500
min          0.014583
25%          3.852083
50%          6.557639
75%         11.769444
max        107.604861
```

VISUALIZATION OF DISTRIBUTION OF LOS W.R.T. FEATURES -

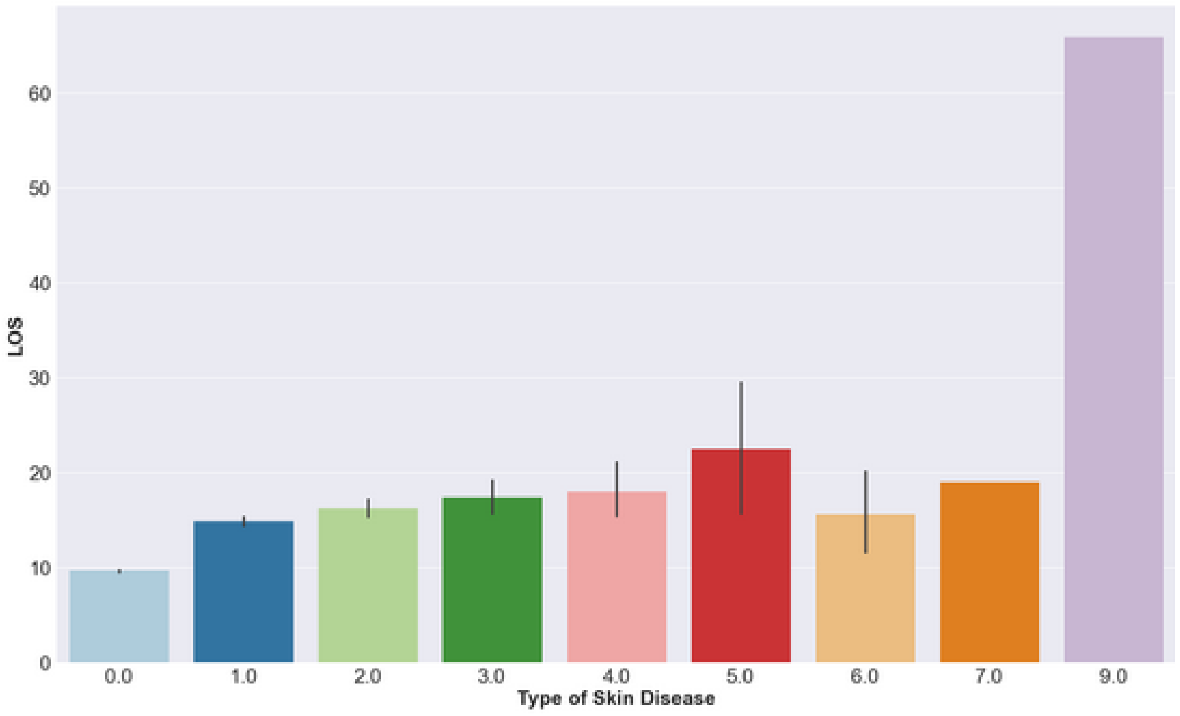
According to the boxplot we find that the Newborns category has the lowest median LOS. We are able to get an insight into the median of the categories of the dataset using a boxplot.



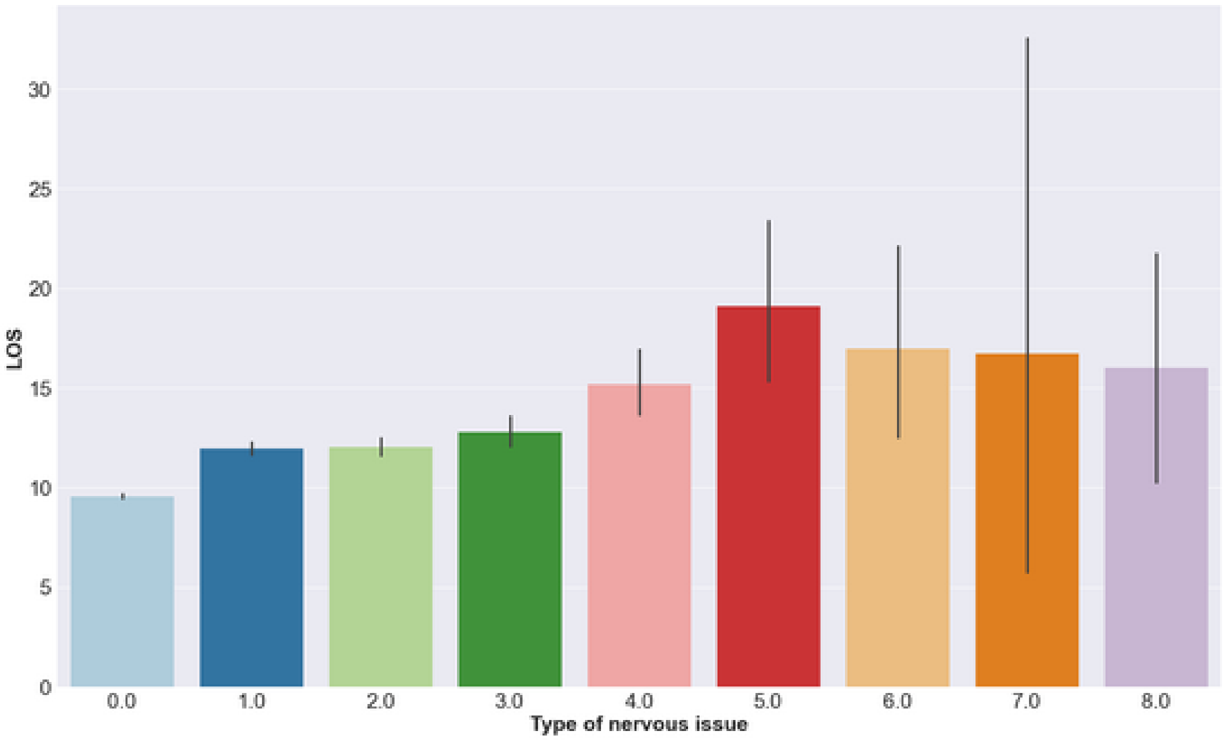
Now we will proceed to print various bar graphs of the Length of Stay for various subcategories of the predictors.



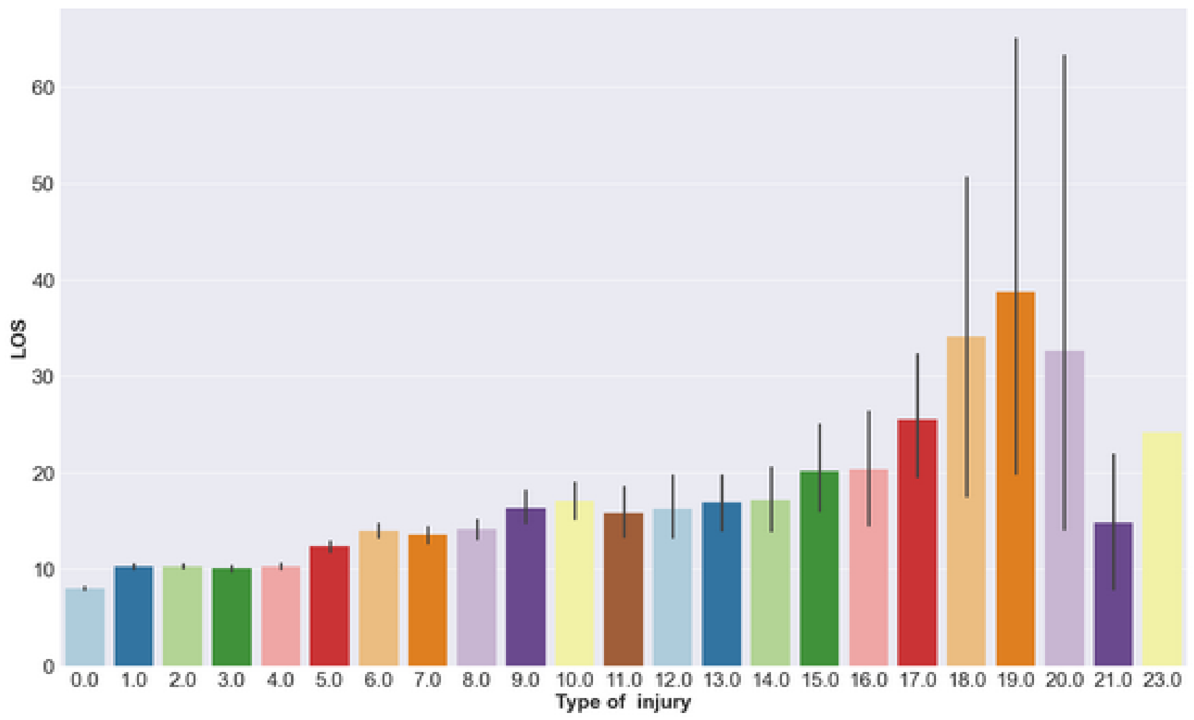
One of the respiratory issues has the average LOS much higher than other types.



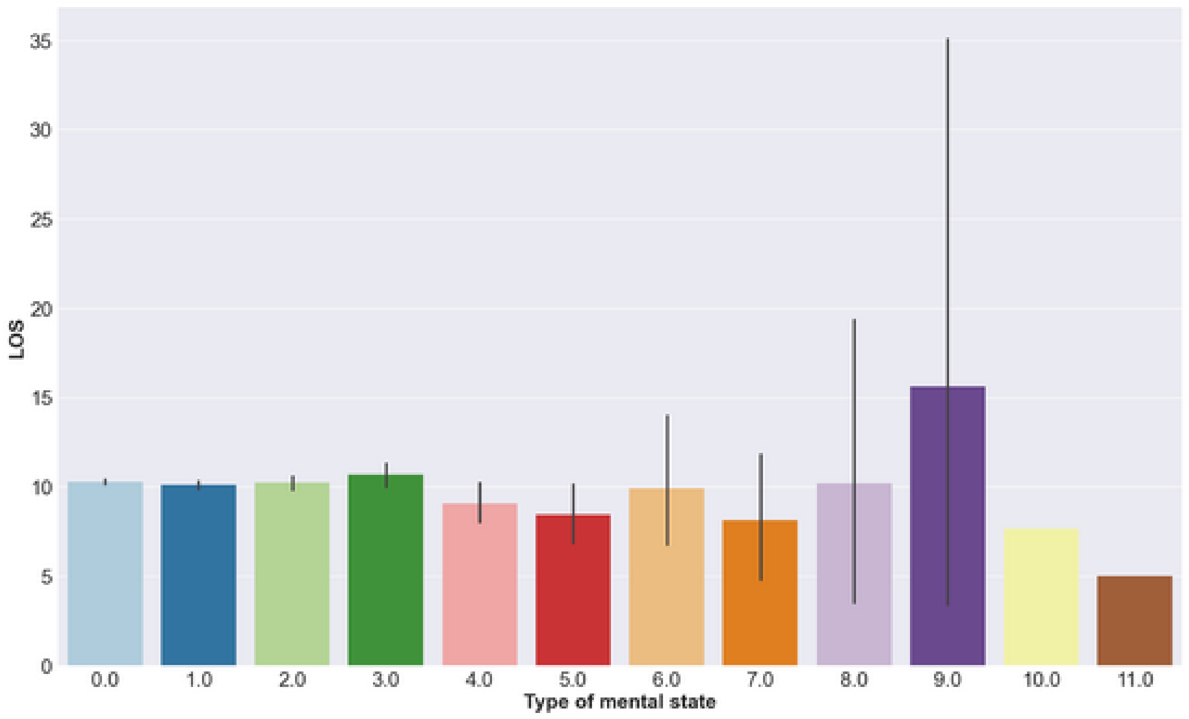
One type of skin disease causes the LOS to be much higher.



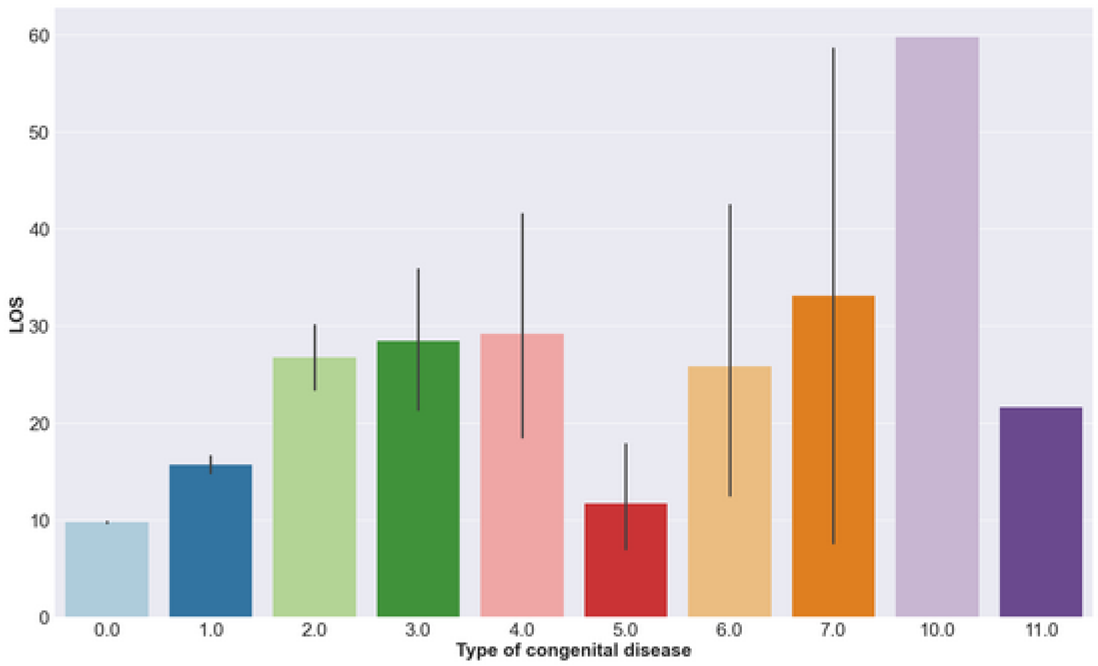
Patients with nervous issues of type 5 - type 8 have higher LOS.



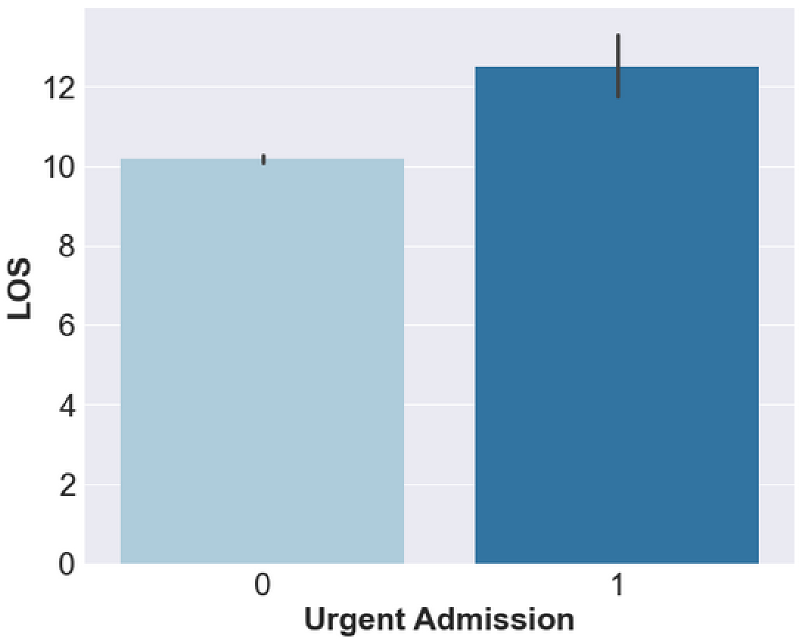
Some types of injuries clearly cause patients to have a higher LOS.



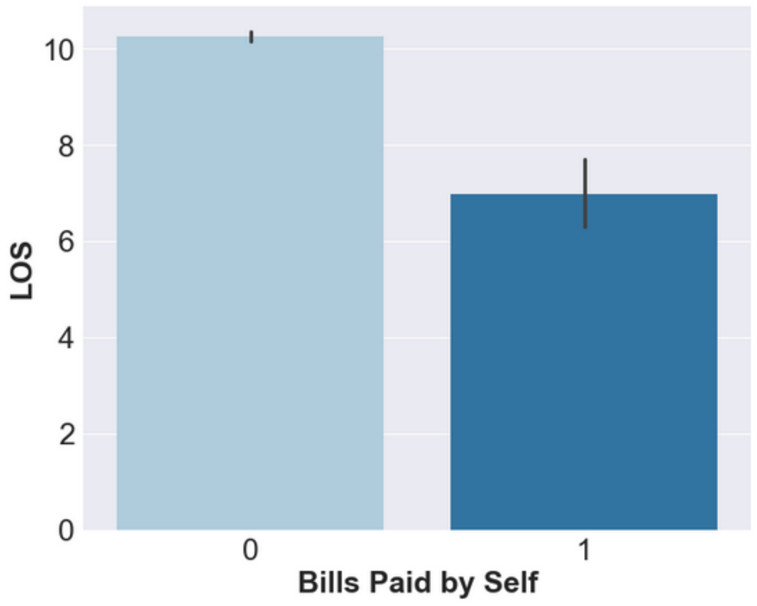
Mental State of the patient also seems to have an effect on the LOS.



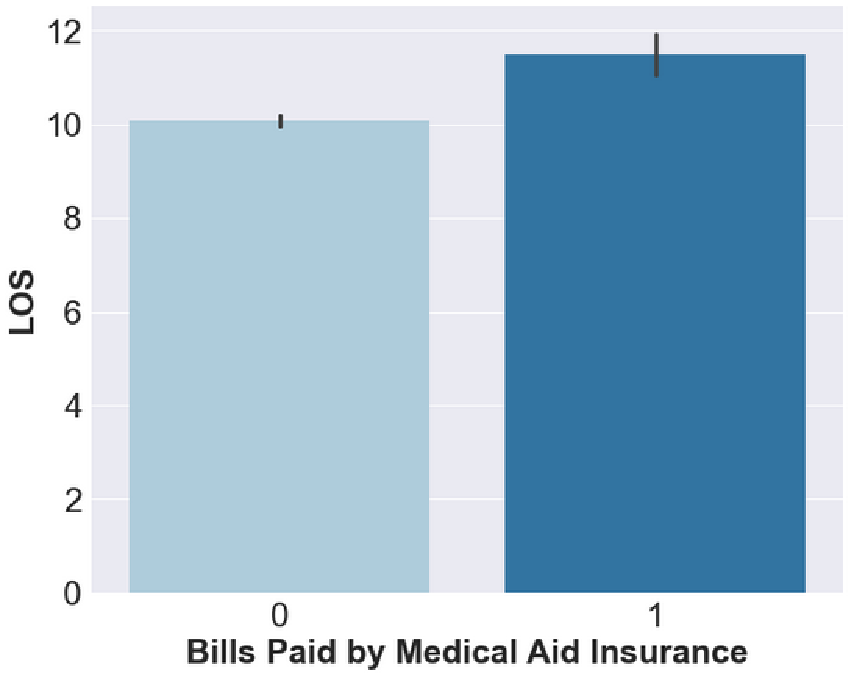
A very high variation in the average LOS can be visualized amongst patients with various congenital diseases.



Urgent Admissions also have a higher LOS.



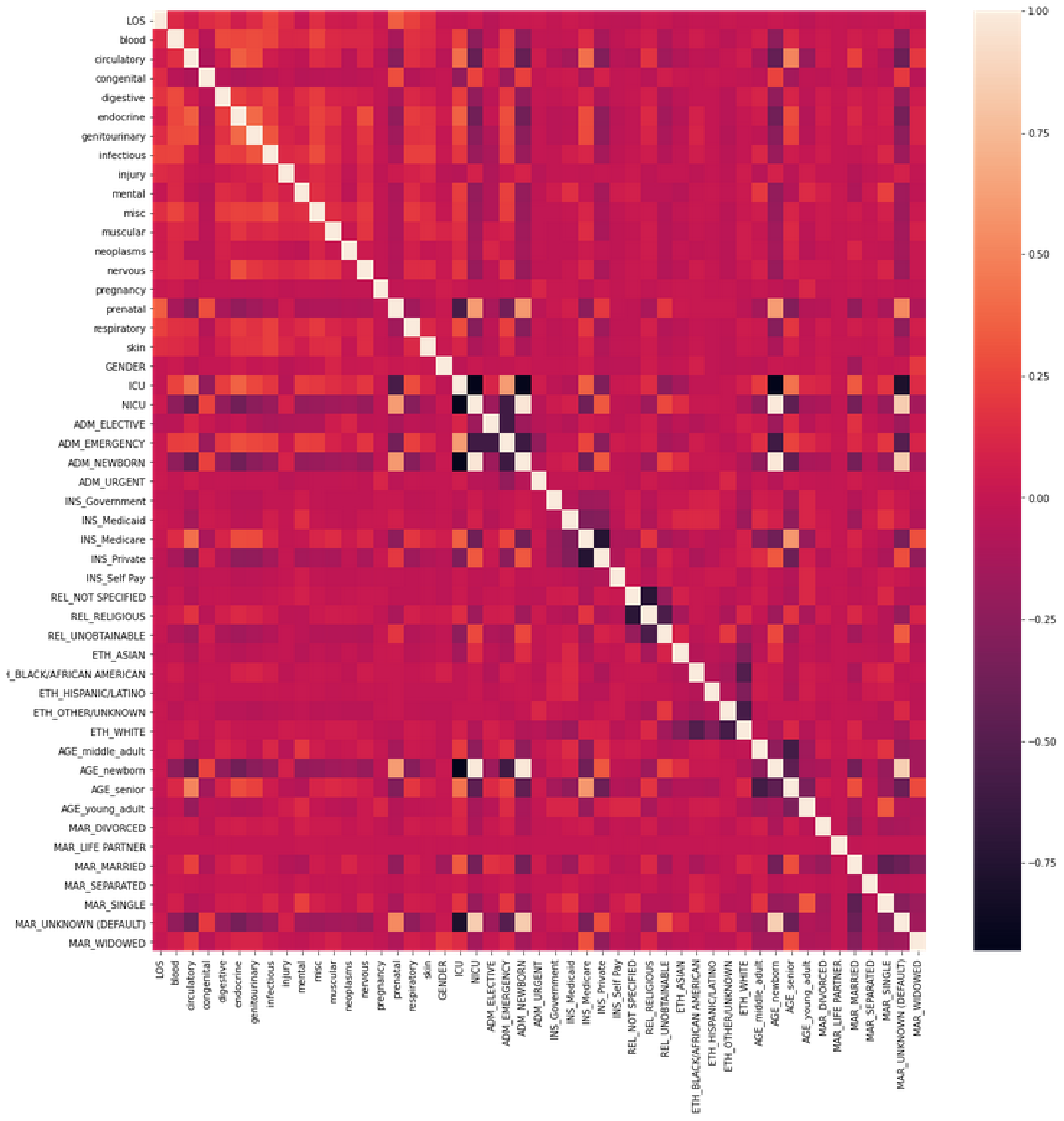
One more interesting statistical observation is that when the patients have to pay for their hospital bills, the average LOS is much lesser. This simply implies that the medical costs are so high that patients who have to pay themselves tend to stay less at the hospital, possibly due to them being unable to afford hospital stays.



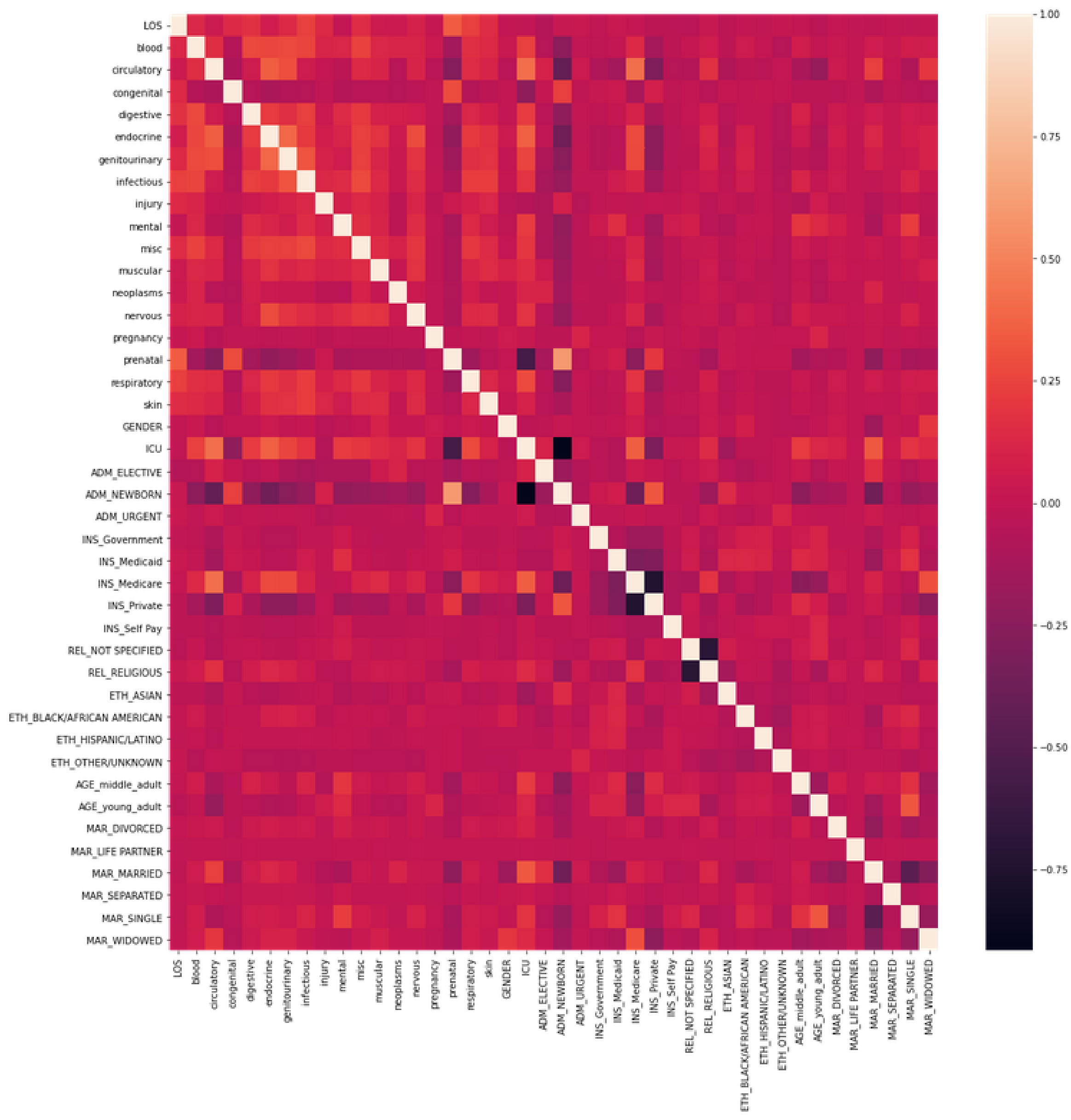
Again, when the costs are covered by insurance, the patients tend to stay at the hospital more.

CORRELATION BETWEEN FEATURES -

Next we print the correlation matrix of the dataset. This matrix illustrates that there are a few variables which are highly correlated with each other. Generally in machine learning, it is helpful to remove features that correlate amongst themselves, and keep those variables that are correlated with the target variable.



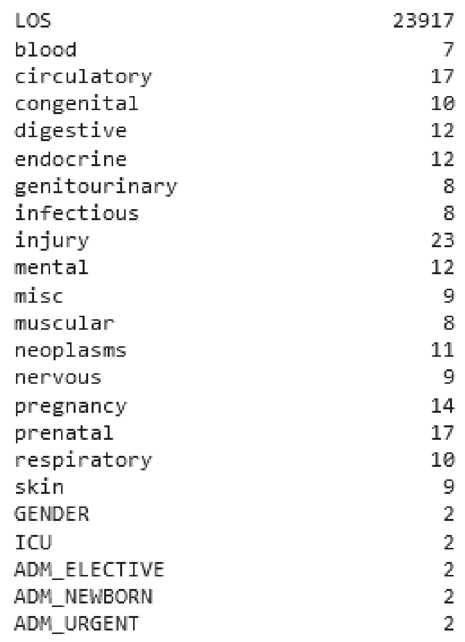
After removing the features - 'ADM_EMERGENCY', 'MAR_UNKNOWN (DEFAULT)', 'NICU', 'AGE_newborn', 'ETH_WHITE', 'REL_UNOBTAINABLE' and 'AGE_senior', we get a much better correlation matrix as shown in the figure below -



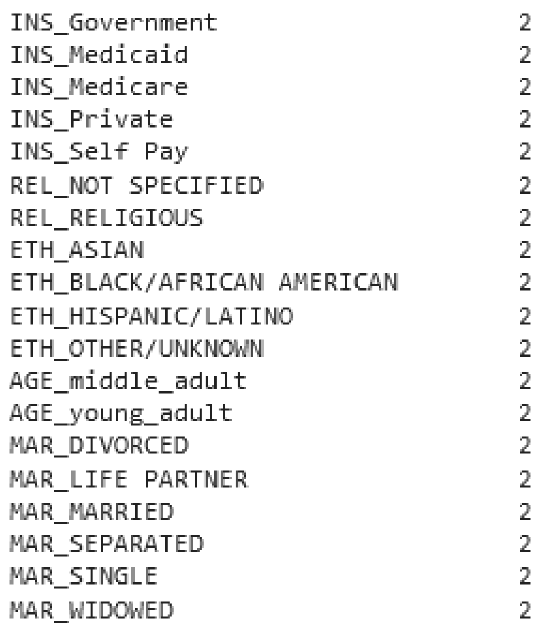
Since all the features of the dataset are already encoded well, and they are of the integer and/or float format, this dataset is ready for training machine learning models, and does not require any further preprocessing.

NUMBER OF UNIQUE OBSERVATIONS FOR EACH FEATURE COLUMN -

We will now find out the number of unique observations in the dataset for each of the listed features. This will give insights about the type of data in various columns, which will help make a decision about which model to choose.



```
LOS                              23917
blood                                7
circulatory                         17
congenital                          10
digestive                           12
endocrine                           12
genitourinary                        8
infectious                           8
injury                              23
mental                              12
misc                                 9
muscular                             8
neoplasms                           11
nervous                              9
pregnancy                           14
prenatal                            17
respiratory                         10
skin                                 9
GENDER                               2
ICU                                  2
ADM_ELECTIVE                         2
ADM_NEWBORN                          2
ADM_URGENT                           2
```

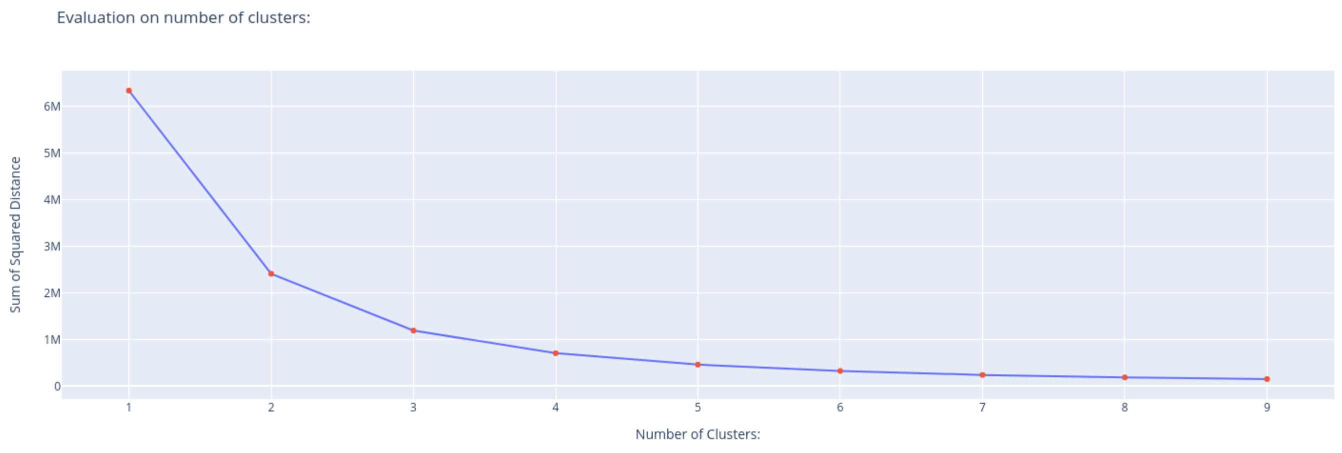


```
INS_Government                     2
INS_Medicaid                       2
INS_Medicare                       2
INS_Private                        2
INS_Self Pay                       2
REL_NOT SPECIFIED                  2
REL_RELIGIOUS                      2
ETH_ASIAN                          2
ETH_BLACK/AFRICAN AMERICAN         2
ETH_HISPANIC/LATINO                2
ETH_OTHER/UNKNOWN                  2
AGE_middle_adult                   2
AGE_young_adult                    2
MAR_DIVORCED                       2
MAR_LIFE PARTNER                   2
MAR_MARRIED                        2
MAR_SEPARATED                      2
MAR_SINGLE                         2
MAR_WIDOWED                        2
```

K-MEANS CLUSTERING

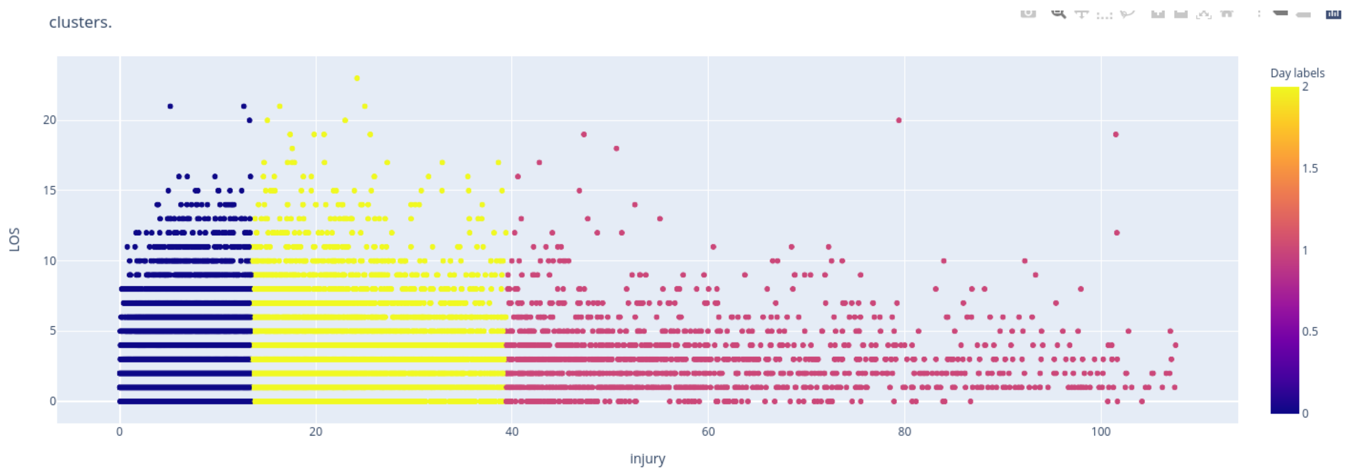
Clustering is used to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance.

K-means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group.



When you plot SSE(sum of squared errors/residuals) as a function of the number of clusters, notice that SSE continues to decrease as you increase k. As more centroids are added, the distance from each point to its closest centroid will decrease.

There’s a sweet spot where the SSE curve starts to bend known as the elbow point. The x-value of this point is thought to be a reasonable trade-off between error and number of clusters. In this example, the elbow is located at x=3:



MACHINE LEARNING MODELS

1. LINEAR REGRESSION

Linear regression is the study of linear, additive relationships between variables. It is a commonly used algorithm and can be imported from the Linear Regression class. A single input variable(the significant one) is used to predict one or more output variables, assuming that the input variable isn't correlated with each other. It is represented as :

$Y_i = \alpha + \beta_i \times x_i + \varepsilon_i$

Here α,β_i are the coefficients/weights or parameters of the regression and ε_i is the error term. The values of these coefficients are found using Ordinary Least Squares method. This means that given a regression line through the data, one calculates the distance from each data point to the regression line, squares it, and sums all of the squared errors together. This is the quantity that ordinary least squares seeks to minimize, also known as the cost function -

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2$$

Cost function for simple linear model

$$\hat{\alpha} = \min_{\alpha} \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2 = \min_{\alpha} \sum_{i=1}^{n} \varepsilon_i^2$$

$$\hat{\beta} = \min_{\beta} \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2 = \min_{\beta} \sum_{i=1}^{n} \varepsilon_i^2$$

The most optimal values are taken when the error term is minimized, which is done by differentiating the cost function w.r.t each of the coefficients and equating it to zero.

2. LASSO REGRESSION

This algorithm is the extension of the linear regression algorithm but it uses Shrinkage, where data values are shrunk to a central value. LASSO stands for - **L**east **A**bsolute **S**hrinkage and **S**election **O**perator. Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model.

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} |w_j|$$

Cost function for Lasso regression

Here the first term of the addition is the sum of squared error and the second term is the regularisation term. As λ increases, more and more coefficients are set to zero and consequently, eliminated.

Tuned parameter -

‘Alpha’ - float, default=1.0

Constant that multiplies the L1 term. Defaults to 1.0. alpha = 0 is equivalent to an ordinary least square, solved by the LinearRegression object.

Values tried -

'alpha': [0.4, 0.6, 0.8, 1]

3. RIDGE REGRESSION

Similar to lasso regression, this is also an extension of linear regression,but it performs L2 regularization, which adds a penalty to the cost function. That is, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients.

As the value of this variable Lambda is reduced, it tries to resemble more like the Linear regression algorithm. Ridge regression shrinks the coefficients and it helps to reduce the model complexity.

But for very large λ, the minimisation would force the coefficient of λ to shrink towards 0. This means that each w shrinks towards 0. So, if some parameters become 0, then the corresponding terms get dropped from the model which basically helps in variable selection.

$$\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j \times x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} w_j^2$$

Cost function for ridge regression

Tuned parameter -

'Alpha' - {float, ndarray of shape (n_targets,)}, default=1.0

Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization.

Values tried -

'alpha': [0.2, 0.4, 0.6, 0.8, 1]

4. K-NEAREST-NEIGHBOURS REGRESSOR

This algorithm can be used for both classification and regression problems. It uses the concept of "feature similarity" in order to predict new data points based on their resemblance to data points in the training set, hence the name. The main steps involved in this algorithm are as follows -

- Finding distance between new point and those in the training set
- ‘K’ closest data points are selected, and the average of these ‘k’ data points becomes the final prediction of the new point.

There are various factors that can be tuned, most importantly the type of distance to be calculated, and the value of ‘k’.

Tuned parameters -

‘N_neighbors’ - int, default=5

Number of neighbors to use by default for kneighbors queries.

‘P’ - int, default=2

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

‘Leaf_size’ - int, default=30

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

Values tried -

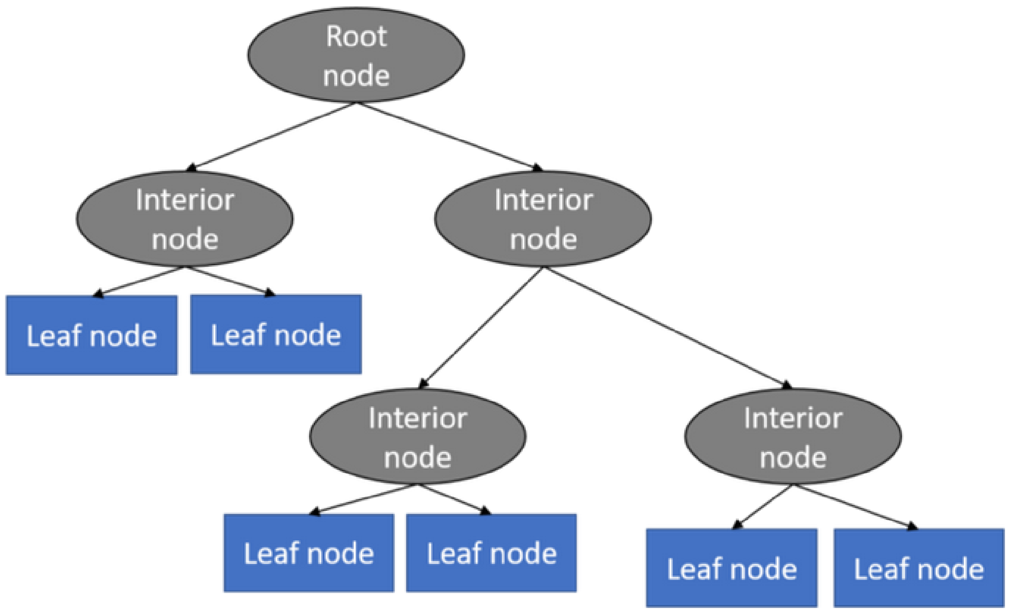
'n_neighbors': [5,7,11,15]

'P':[1,2]

'Leaf_size':[5,15,25,30,45]

5. DECISION TREE REGRESSOR

Decision trees can build either regression or classification models in the form of a tree with **decision nodes, interior** and **leaf nodes**. A **decision node** has two or more branches , each representing values for the attribute tested. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data. The **interior nodes** represent the features of a data set and the branches represent the decision rules. Finally, the **leaf nodes** represent the outcome.



Tuned Parameters -

‘Max_depth’ - int, default=None

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

‘Min_samples_split’ - int or float, default=2

The minimum number of samples required to split an internal node:

> If int, then consider min_samples_split as the minimum number.
> If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Values tried -
'max_depth': [6,8,10]
'min_samples_leaf':[1,5,10,20,50]

6. RANDOM FOREST REGRESSOR

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn’t depend on one decision tree but multiple decision trees. In the case of a regression problem, the final output is the mean of all the outputs (Aggregation). Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model (Bootstrap).

Thus, this is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees using the technique of Bootstrap and Aggregation, commonly known as **bagging**.

Random forests provide an improvement over bagged trees by way of a random forest small tweak that decorrelates the trees. In bagging, we build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Tuned Parameters -

'Max_depth' - int, default=None

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

'Min_samples_split' - int or float, default=2

The minimum number of samples required to split an internal node:

> If int, then consider min_samples_split as the minimum number.
> If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Values tried -
'max_depth': [8,10]
'min_samples_leaf':[5,10,20]

7. XGB REGRESSOR

Extreme Gradient Boosting is an implementation of gradient boosting decision tree algorithm. This software library is mainly focused on computational speed and model performance. Boosting is an ensemble-based technique in which new models are trained sequentially, so as to correct the errors made by existing/old models. In this approach, new models are added sequentially, until no further improvements are noticed.

The main advantages of this algorithm are –support for parallel processing (all threads on the CPU, along with GPU), inbuilt cross-validation, has a variety of regularization techniques to reduce over-fitting. Gradient boosting assists in the prediction of optimal gradient for additive models, unlike classical gradient descent techniques where output errors are reduced at each iteration.

Tuned Parameters -

‘alpha’ - [default=0, alias: reg_alpha]
L1 regularization term on weights. Increasing this value will make the model more conservative.

‘Eta’ - [default=0.3, alias: learning_rate, range: [0,1]]
Step size shrinkage used in updates to prevent overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.

‘Max_depth’ - [default=6, range: [0,∞]]
Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 is only accepted in lossguided growing policy when tree_method is set as hist or gpu_hist and it indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree.

‘Lambda’ - [default=1, alias: reg_lambda]
L2 regularization term on weights. Increasing this value will make the model more conservative.

Values tried -

'max_depth': [10]

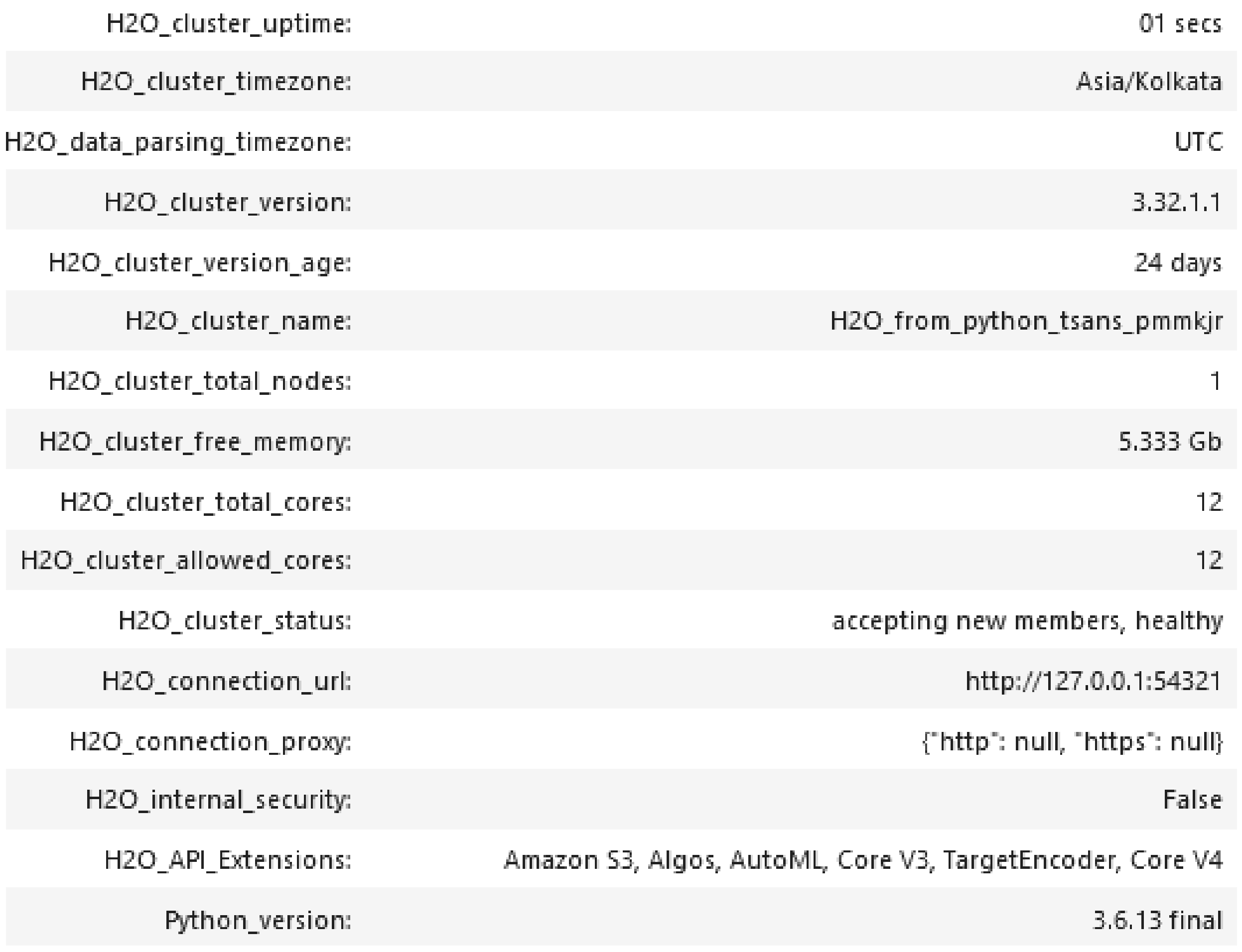
'eta': [0.05, 0.1]

'lambda': [0.9, 1]

'alpha': [0.1,0.02]

8. H2O AUTOML

H2O’s AutoML is an extremely helpful tool for providing a simple wrapper function that performs a large number of modelling-related tasks, automating the machine learning workflow, which includes automatic training and tuning of multiple models bound by a user-specified limit of number of models. Stacked Ensembles based on all previously trained models are automatically trained on collections of individual models to produce highly predictive ensemble models. Usually, these ensemble models are the top performing models in the AutoML Leaderboard.



H2O_cluster_uptime:	01 secs
H2O_cluster_timezone:	Asia/Kolkata
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.32.1.1
H2O_cluster_version_age:	24 days
H2O_cluster_name:	H2O_from_python_tsans_pmmkjr
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	5.333 Gb
H2O_cluster_total_cores:	12
H2O_cluster_allowed_cores:	12
H2O_cluster_status:	accepting new members, healthy
H2O_connection_url:	http://127.0.0.1:54321
H2O_connection_proxy:	{"http": null, "https": null}
H2O_internal_security:	False
H2O_API_Extensions:	Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
Python_version:	3.6.13 final

Entire training and testing data was converted to an H2O Frame. A local H2O instance was set up on our personal device and 6GB RAM was allocated to it. The only parameter was max_models, which was set to 12. AutoML's Leaderboard helped provide valuable information about each prediction model.

9. NEURAL NETWORKS

The basic building block of a neural network can be termed a neuron. It's task is to take inputs and produce a linear regression equation. Multiple perceptrons in various layers of a neural network have to generate multiple linear regression equations at multiple points.

The first step of making a neural network is starting with a structure with an appropriate amount of layers and corresponding amount of neurons per layer. Each neuron in the input/first layer represents an attribute/feature of the dataset. The coefficients of the linear regression equation need to be initialized at first. Also known as the weights, we can either initialize them to zero, or to some random values.

In the next step, also known as feed forward, each layer produces a particular output, which acts as an input to the next layer of the network. Since we are working on a regression problem, the last layer in our case has only one output, that is the LOS column.

Error is calculated at the output layer, and is sent back to the previous layers, one by one, and the goal is to refine the values of the coefficients of the neural network, to reduce this error. This process is known as Back Propagation. There are multiple algorithms that can be used for this step. For example, in the Gradient Descent Algorithm, we continuously update the initialised weights in the negative direction of the slope to reach the minimal point.

Keras is a powerful and open-source Python library for development and evaluation of deep neural networks. It wraps the efficient numerical computation libraries for simple implementation. The various functions, their default values, customizable parameters, etc can be found using their documentation. These various parameters and the physical size of the neural networks need to be realized after extensive experimentation.

SCORING METRICS

1. MEAN SQUARED ERROR

The evaluation metric used for this project was the Mean Squared Error (MSE). MSE is an extremely popular metric when training machine learning models and deep neural networks. MSE is essentially the variation of the prediction errors. Prediction errors are the difference between theoretical values and predicted/experimental values. MSE is obtained by squaring the prediction errors and taking their mean, hence the name - Mean of Squared Errors.

$$MSE = \frac{\sum_{i=0}^{N} (Y_i - \widehat{Y})^2}{N}$$

- $\widehat{Y}$ is predicted value
- N is total number of observations

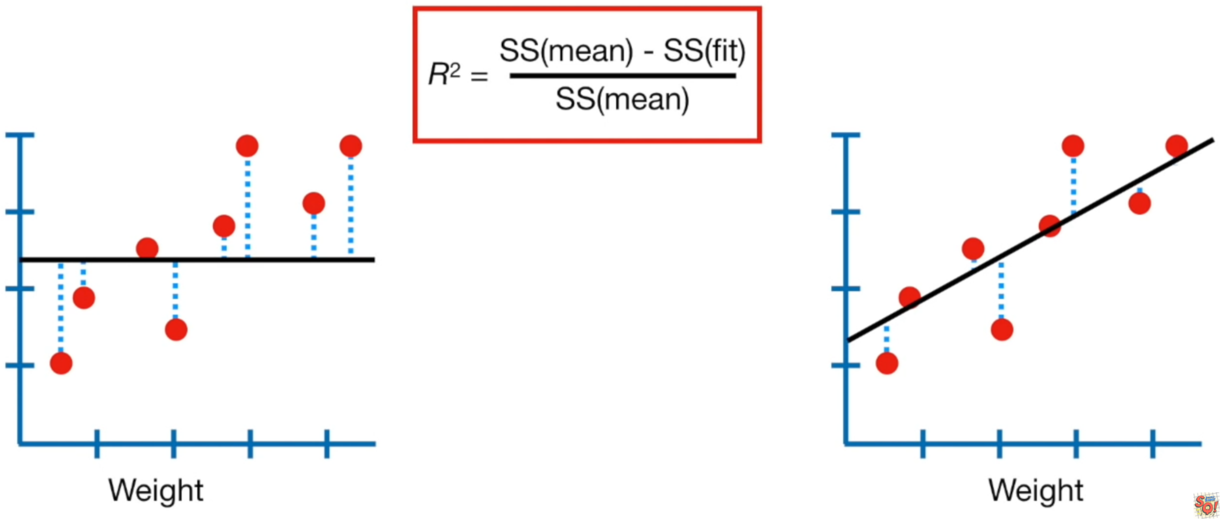
2. R2 SCORE

The other metric for comparison of models was chosen to subbe the R2 Score, which in mathematical terms is known as the Coefficient of Determination. It is calculated as “1 - residual sum of square / total sum of squares”. It is used as a way of measuring the fitting of a particular model on a given dataset. Usually, a higher R-squared indicates a better fit for the model.

$$R^2 = 1 - \frac{\sum_{i=0}^{N} (Y_i - \widehat{Y})^2}{\sum_{i=0}^{N} (Y_i - \overline{Y})^2}$$

- $\widehat{Y}$ is predicted value
- $\overline{Y}$ is mean value of Y
- N is total number of observations

R2 score can also be visualized in a much more visually appealing way, especially for regression models. It can be interpreted as follows, where SS means Sum of Squared Residuals.



$$R^2 = \frac{SS(mean) - SS(fit)}{SS(mean)}$$

EXPERIMENTATION

SPLITTING OF COMMON TRAIN AND TEST DATA -

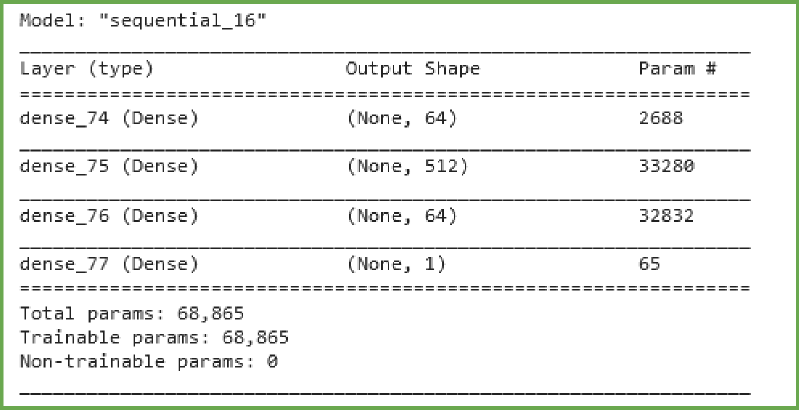
First the data was split into a common 80% training set, and 20% testing set. All the training, hyperparameter tuning and cross validation would take place on this common training set and in order to compare the performance of models against one another, the MSE on this common test set would be considered and compared against the MSE score from other models.

HYPERPARAMETER TUNING -

For each of the applied ML models, first the appropriate values of hyperparameters were found for each model, using gridsearchCV. The process involved us specifying the possible values in the form of lists, then using GridsearchCV, multiple models were trained with all possible combinations of those hyperparameters, and the best performing model was chosen to give the appropriate parameters for each model. The parameters for each of the models are listed in the table below.

MODEL	**OPTIMAL VALUES OF HYPERPARAMETERS**
Lasso Regression	**{'alpha': 1}**
Ridge Regression	**{'alpha': 0.2}**
KNeighborsRegressor	**{'leaf_size': 5, 'n_neighbors': 5, 'p': 1}**
DecisionTreeRegressor	**{'max_depth': 10, 'min_samples_leaf': 1}**

RandomForest	**{'max_depth': 8, 'min_samples_leaf': 20}**
XGBRegressor	**{'alpha': 0.1, 'eta': 0.1, 'lambda': 0.9, 'max_depth': 10}**

Neural Networks	
	```
	Model: "sequential_16"
	_________________________________________________________________
	Layer (type)                 Output Shape              Param #
	=================================================================
	dense_74 (Dense)             (None, 64)                2688
	_________________________________________________________________
	dense_75 (Dense)             (None, 512)               33280
	_________________________________________________________________
	dense_76 (Dense)             (None, 64)                32832
	_________________________________________________________________
	dense_77 (Dense)             (None, 1)                 65
	=================================================================
	Total params: 68,865
	Trainable params: 68,865
	Non-trainable params: 0
	_________________________________________________________________
	```
	 >>opt = keras.optimizers.Adam(learning_rate=0.01) >>model.compile(optimizer=opt,loss='mean_squared_error') >>estp = EarlyStopping(monitor='val_loss', min_delta=0,patience=5, verbose=1, mode='auto',restore_best_weights=True) >>model.fit(X_train,y_train,validation_split=0.15,shuffle='True',verbose=2,epochs=200, callbacks=[estp])

CROSS-VALIDATION -

Using these parameters, the model was fit on the training set and we then used cross validation The working of cross validation is as follows-

- Divide the dataset into “k” folds, k equal to 10 in our case

- Fit the model on “k-1” folds, and validate the model using the remaining “k”th fold.
- Repeat this process until each fold has served as a validation set once, and keep saving the validation score (MSE in our case).
- Take the average score out of all these validation scores, this value will be the final performance metric of the particular model.

We used cross validation with the value of “k” set to 10, to make sure the model is not too biased, and printed the Mean Squared Errors from each model’s cross validation.

PREDICTIONS ON COMMON TEST SET -

After this, predictions were made on the common test set, for each different Machine Learning Model in order to find the Mean Squared Error and R2 Score, to evaluate and compare the performance against the other machine learning models.

RESULTS AND DISCUSSIONS

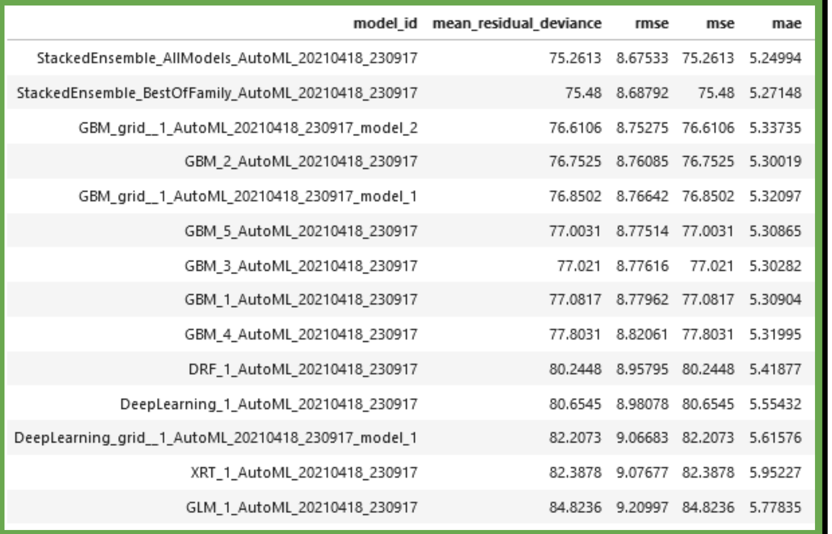
Attached below are the cross validation scores (MSE), and then, the MSE and R2 score on the common test set for each of the chosen machine learning models.

MODEL	**SCORES**
Linear Regression	**Cross-Validation Performance on Train Set- Average MSE: (84.827), Std: (4.581) [86.33833066, 83.0994207, 82.0277603, 79.63299763, 80.64801847, 84.77253291, 95.9750763, 86.52937859, 81.08468335, 88.1628727]**
	Performance on test set - MSE: 80.736870 R2: 0.320258
Lasso Regression	**Cross-Validation Performance on Train Set - Average MSE: (93.221), Std: (5.059) [96.31551509, 90.76589628, 89.75318692, 85.97663016, 88.95836889, 93.56606187, 104.53428747, 95.62223045, 89.91683688, 96.80410916]**
	Performance on test set - MSE: 88.584775 R2: 0.254184

Ridge Regression	**Cross-Validation Performance on Train Set - Average MSE: (84.824), Std: (4.572) [86.31885392, 83.10569241, 82.02911999, 79.63449897, 80.65964831, 84.78107202 95.95983317, 86.54533858, 81.08924216 88.11261049]**
	Performance on test set - MSE: 80.737011 R2: 0.320256
KNeighborsRegressor	**Cross-Validation Performance on Train Set - Average MSE: (92.833), Std: (5.254) [97.98944568, 85.88757291, 90.75227029 87.39146019, 87.26811143, 96.86356607 103.47298733, 93.38492904, 90.8542695 94.46642408]**
	Performance on test set - MSE: 87.193987 R2: 0.265894
DecisionTreeRegressor	**Cross-Validation Performance on Train Set - Average MSE: (91.999), Std: (6.164) [90.48306121, 93.41849389, 92.20483166 87.8251745, 82.63292982, 95.85661098 104.0074765, 92.58386111, 83.19337222 97.7869355]**
	Performance on test set - MSE: 89.917556 R2: 0.242963

RandomForest	**Cross-Validation Performance - Average MSE: (80.511), Std: (3.955) [80.87619701 79.83891909 79.77666988 77.80566044 73.99302857 82.45182817 89.44769776 82.15626557 76.49900623 82.26431406]**
	Performance on test set - MSE: 77.658753 R2: 0.346173
XGBRegressor	**Cross-Validation Performance on Train Set - Average MSE: (79.479), Std: (3.695) [81.70358165, 76.9868551, 76.95338325 77.2394411, 73.94794335, 81.61912017 87.54907857, 82.31552222, 77.11287465 79.36293046]**
	Performance on test set - MSE: 75.192617 R2: 0.366936
Neural Network	**Performance on test set - MSE: 74.571474 R2: 0.372165**

H2O_AutoML



model_id	mean_residual_deviance	rmse	mse	mae
StackedEnsemble_AllModels_AutoML_20210418_230917	75.2613	8.67533	75.2613	5.24994
StackedEnsemble_BestOfFamily_AutoML_20210418_230917	75.48	8.68792	75.48	5.27148
GBM_grid__1_AutoML_20210418_230917_model_2	76.6106	8.75275	76.6106	5.33735
GBM_2_AutoML_20210418_230917	76.7525	8.76085	76.7525	5.30019
GBM_grid__1_AutoML_20210418_230917_model_1	76.8502	8.76642	76.8502	5.32097
GBM_5_AutoML_20210418_230917	77.0031	8.77514	77.0031	5.30865
GBM_3_AutoML_20210418_230917	77.021	8.77616	77.021	5.30282
GBM_1_AutoML_20210418_230917	77.0817	8.77962	77.0817	5.30904
GBM_4_AutoML_20210418_230917	77.8031	8.82061	77.8031	5.31995
DRF_1_AutoML_20210418_230917	80.2448	8.95795	80.2448	5.41877
DeepLearning_1_AutoML_20210418_230917	80.6545	8.98078	80.6545	5.55432
DeepLearning_grid__1_AutoML_20210418_230917_model_1	82.2073	9.06683	82.2073	5.61576
XRT_1_AutoML_20210418_230917	82.3878	9.07677	82.3878	5.95227
GLM_1_AutoML_20210418_230917	84.8236	9.20997	84.8236	5.77835

Performance on test set -
MSE: 71.344349
R2: 0.399335

FURTHER RESULTS

We shall now be comparing some models theoretically that we have used

LR vs Decision Tree :

- Decision trees supports non linearity, where LR supports only linear solutions.
- When there are large number of features with less data-sets(with low noise), linear regressions may outperform Decision trees/random forests. In general cases, Decision trees will be having better average accuracy.
- For categorical independent variables, decision trees are better than linear regression.

- Decision trees handles colinearity better than LR.

LR vs KNN :

- KNN is a non -parametric model, whereas LR is a parametric model.
- KNN is slow in real time as it have to keep track of all training data and find the neighbor nodes, whereas LR can easily extract output from the tuned θ coefficients.

LR vs Neural Networks :

- Neural networks need large training data compared to LR model, whereas LR can work well even with less training data.
- NN will be slow compared to LR.
- Average accuracy will be always better with neural networks.

Lasso vs Ridge regression

- The Lasso's penalty term is based on the sum of absolute coefficients, and the specification of a penalty coefficient is similar to that of Ridge regression.
- However, the Lasso is more computationally intensive so it is slower to train.

Decision tree vs Random Forest :

- Random Forest is a collection of decision trees and average/majority vote of the forest is selected as the predicted output.
- Random Forest model will be less prone to overfitting than Decision tree, and gives a more generalized solution.

- Random Forest is more robust and accurate than decision trees.

Random forest vs Gradient boosting(XGB)

- Random forest build trees in parallel and thus are fast and also efficient. Parallelism can also be achieved in boosted trees such as XGB regressor.
- One drawback of gradient boosted trees is that they have a number of hyperparameters to tune, while random forest has only one hyperparameter
- Though both random forests and boosting trees are prone to overfitting, boosting models are more prone.

Decision tree vs KNN :

- Both are non-parametric methods.

- Decision tree supports automatic feature interaction, whereas KNN cant.

- Decision tree is faster due to KNN’s expensive real time execution.

REFERENCES

- https://www.python-course.eu/Decision_Trees.php
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
- https://xgboost.readthedocs.io/en/latest/parameter.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
- http://docs.h2o.ai/h2o-tutorials/latest-stable/h2o-world-2017/automl/index.html
- https://en.wikipedia.org/wiki/Coefficient_of_determination
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://mimic.physionet.org/
- https://github.com/sdasara95/Machine-Learning-in-Healthcare/tree/master/data