Stroke Prediction using Machine Learning Models in Python

Milestone: Project proposal

Group 24

Student Name1: Samruddhi Kulkarni

Student Name2: Kartik Vadlamani

857-209-4257

857-376-9931

kulkarni.samr@northeastern.edu

vadlamani.k@northeastern.edu

Percentage of effort contributed by Student 1: 50%

Percentage of effort contributed by Student 2: 50%

Signature of Student 1: Samruddhi Kulkarni

Signature of Student 2: Kartik Vadlamani

Submission date: 1st May, 2022

Problem Setting

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. This is alarming but with the advent of various ML techniques it is possible to implement a prediction model to predict stroke and prevent irreversible damage to the body or even death!

Problem Definition

Our problem is best framed as a binary classification problem which predicts whether a patient is likely to get a stroke based on the input parameters like gender, age, various diseases, and smoking status.

Data Source

Dataset is 5110x12 records table taken from an online platform for data science projects and competitions - Kaggle

Link: Stroke Prediction Dataset | Kaggle

Data Description

Sr. No.	Fields	Description	Datatype
1	id	Patient id	int64
2	gender	Patient gender	object
3	age	Patient age	float64
4	hyper tension	If patient suffers from Hypertension	int64
5	heart_ disease	If patient suffers from Heart disease	int64
6	ever_ married	If patient is married	object
7	work_type	Patient's employment type	object
8	Residence_type	Does patient live in rural or urban area	object

9	avg_glucose_level	Patient's average glucose level in blood	float64
10	smoking_status	If patient smokes, formerly smoked or never smoked	object
11	bmi	Patient's Body Mass Index value	float64
12	stroke	If patient had a stroke	int64

```
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
```

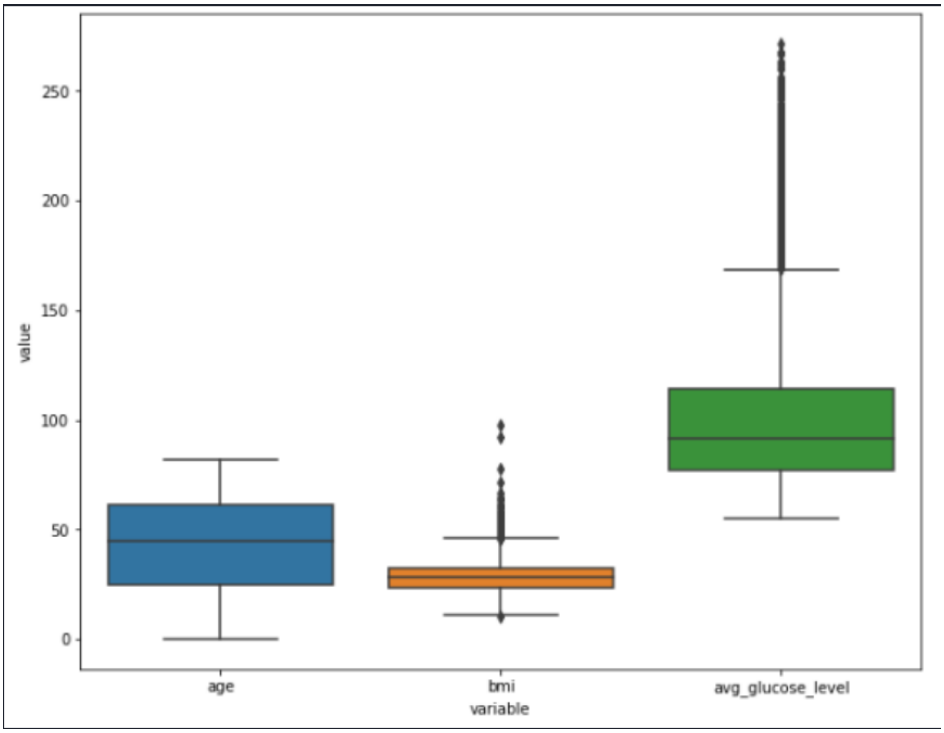
Data Exploration

1. **Descriptive Statistics of the data**

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

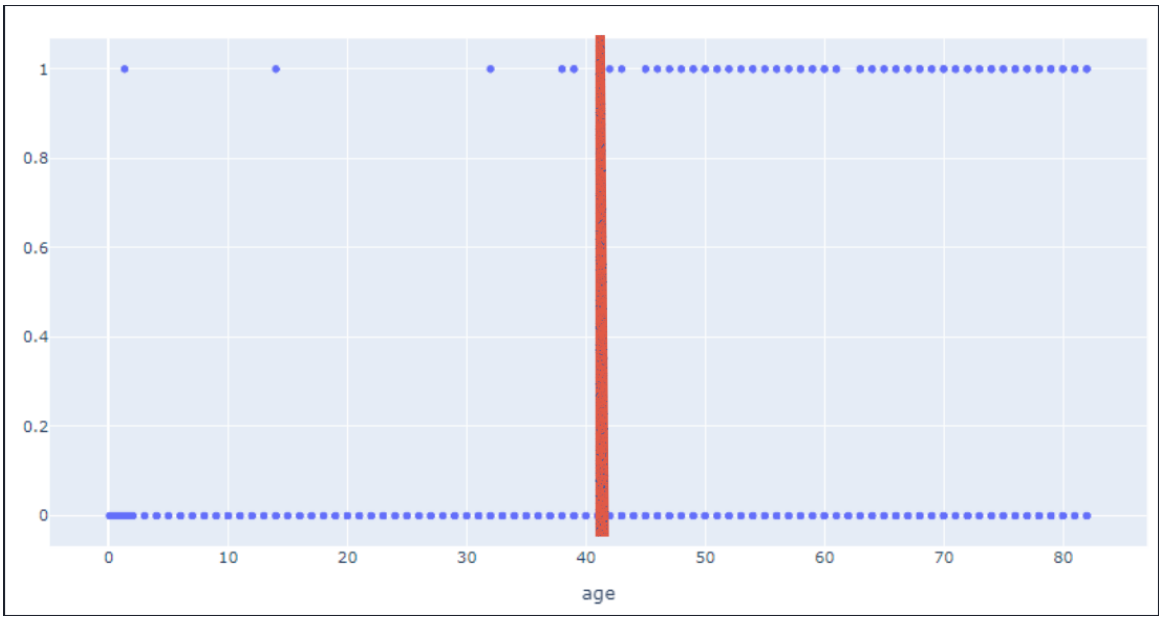
2. **Data duplicates**
   There are no duplicates found in the data.
3. **Imputation of null values**
   201 N/A values for BMI field replaced with median value
4. **Handling Inconsistent data**
   'Other' category from gender field with just one record removed for simplification

5. **Removing Redundant data**
   'id' field is removed as redundant
6. **Data Visualization**



Boxplots of Numerical Variables in the data

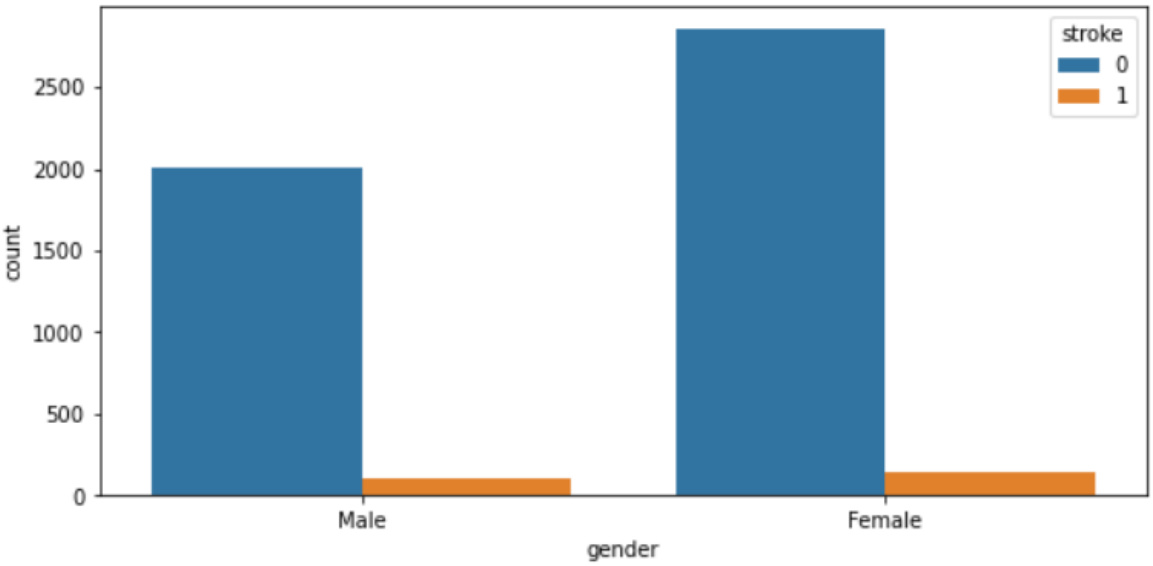
- The above boxplot indicates that the feature age has no outliers compared to that of the bmi and avg_glucose_level.
- These should naturally vary as they are the measurements of a natural human body and will vary from person to person. Hence the features are retained and not removed.



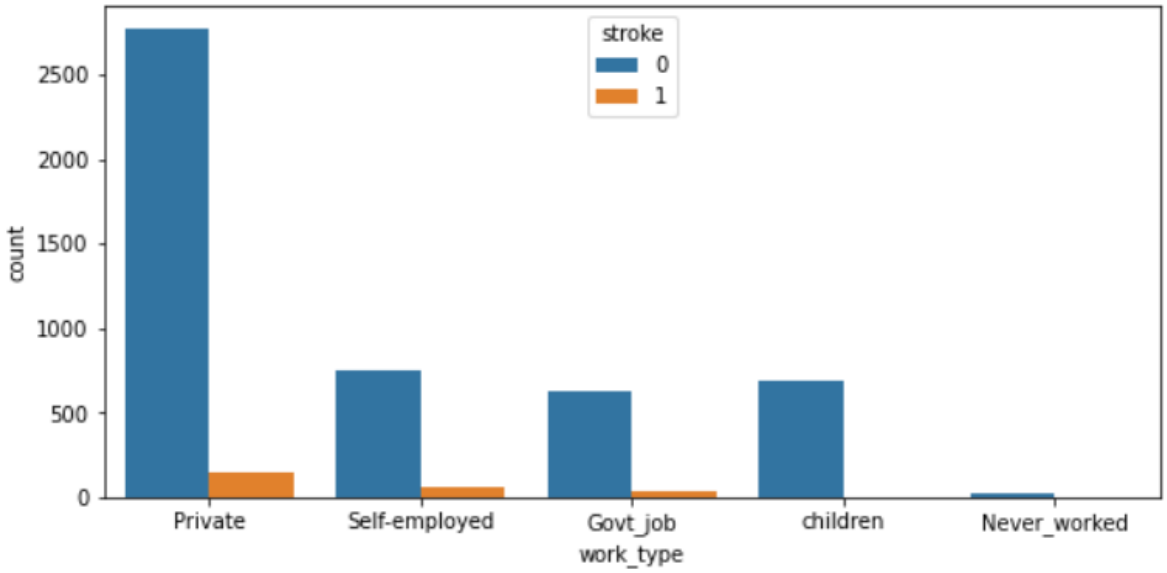
Scatterplot of Age vs Stroke

- This scatterplot implies that most people of age above 40 are more prone to getting a stroke than that compared to people below 40.

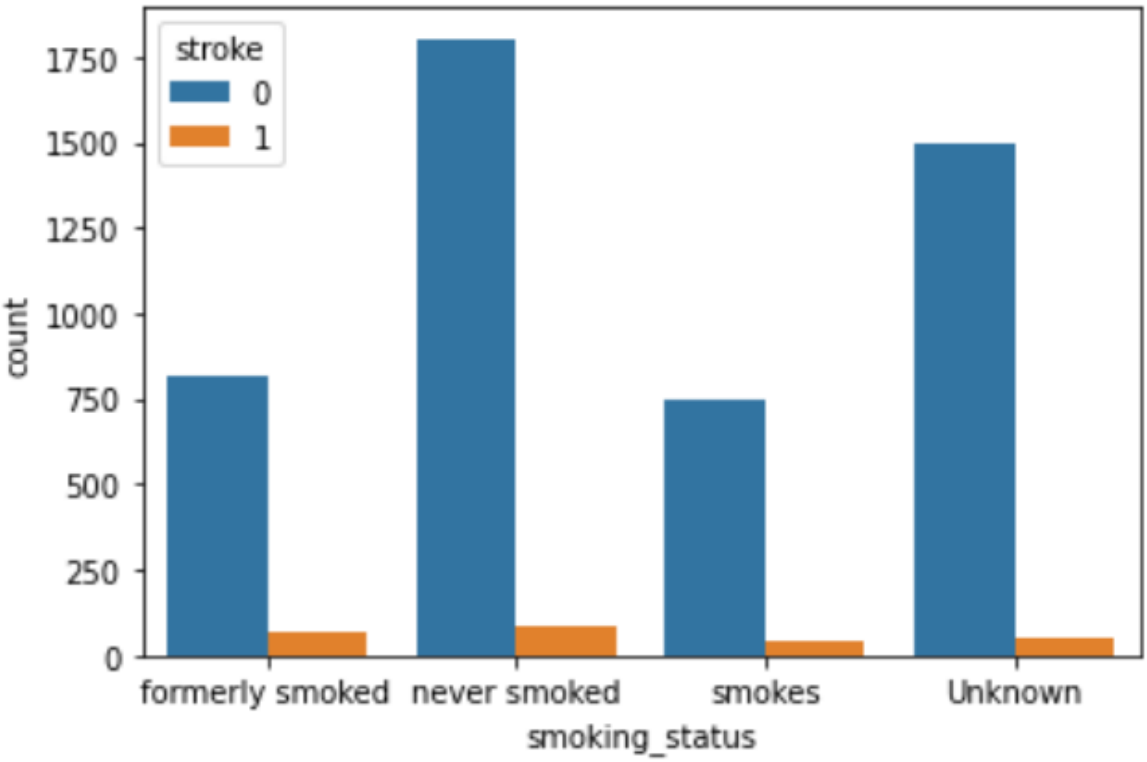
Stroke Vs Categorical Data:



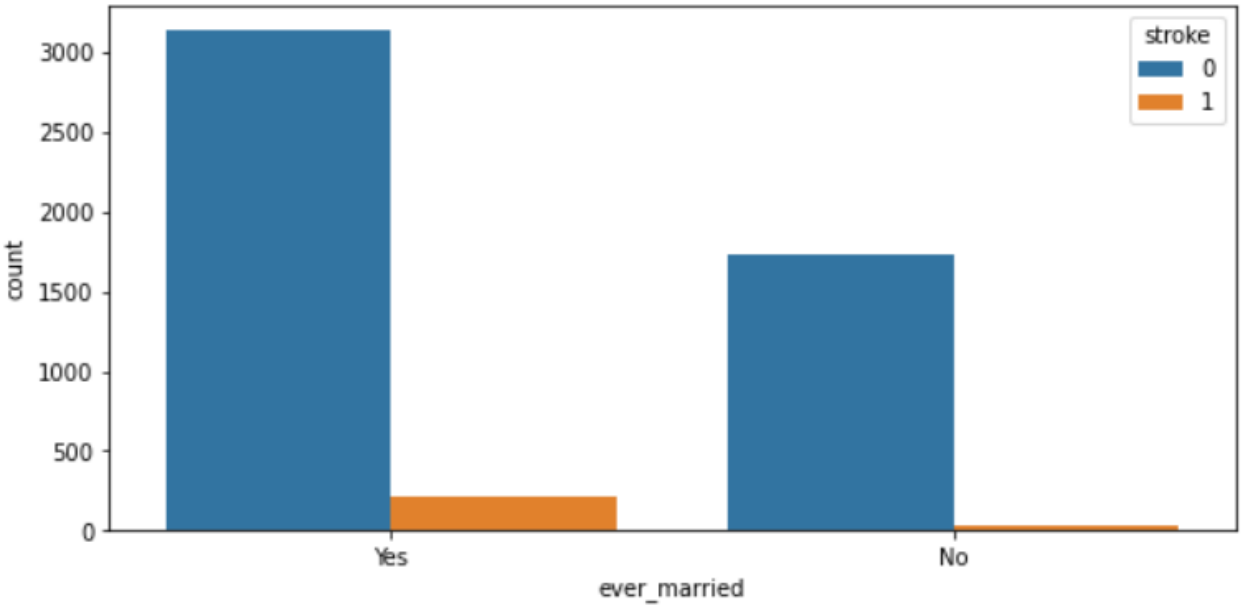
- The bar plot above indicates that the percentage of males getting stroke is more than percentage of females getting stroke in this dataset.



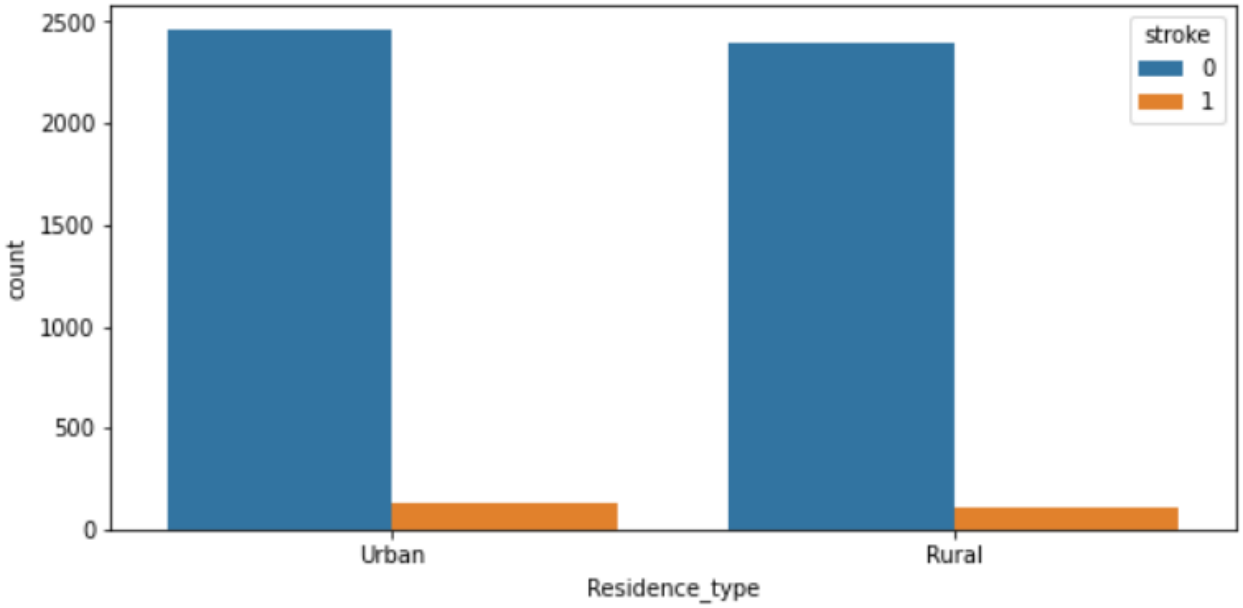
- This above bar plot shows that people who are working in a private sector are more likely to get a stroke than compared to all the other work types in this dataset.



- The above bar plot indicates that percentage of people who smoke/formerly smoked and have got a stroke is higher than other categories in this dataset .

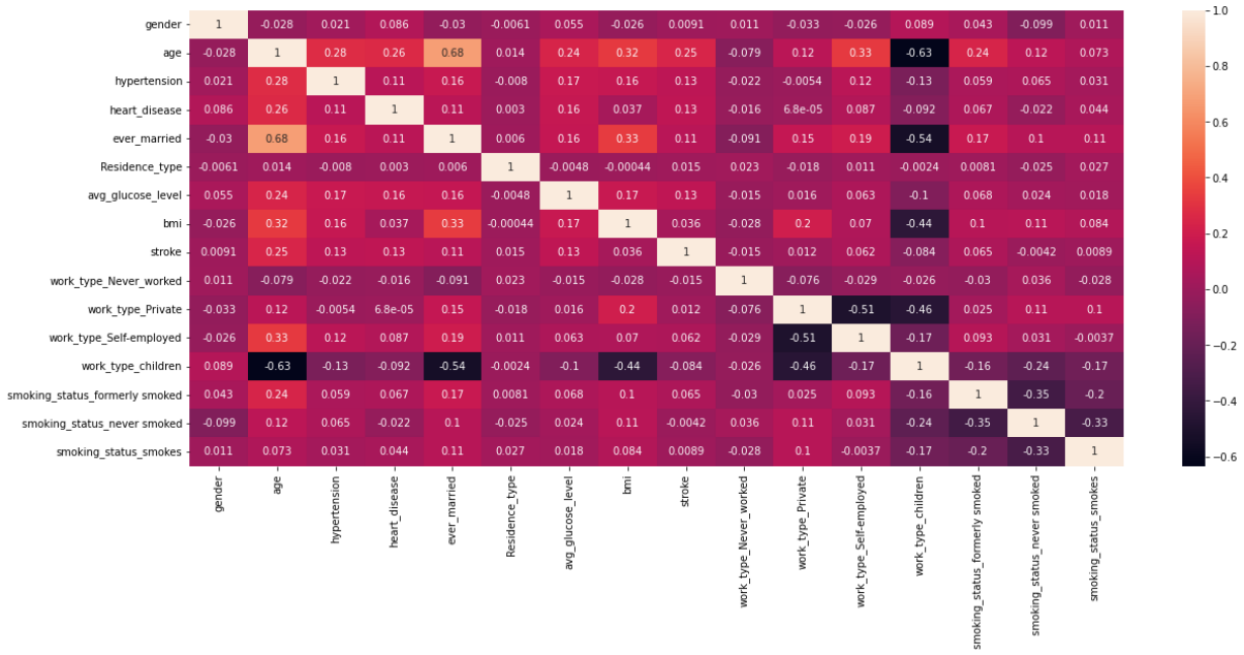


- From the above bar plot, we can conclude that married people have a higher chance of getting a stroke than unmarried people in this dataset.



- From the above bar plot, we can infer that people living in urban areas are more prone to get a stroke than those living in rural areas in this dataset.

Heatmap for all variables in the dataset



- There is no significant correlation between variables to have considered them for feature reduction.

Data Mining Tasks

1. Encoding of Categorical variables

The performance of the model does not only depend on the model and the hyperparameters but also the input that is fed to the model. It is therefore important to prepare data to help the model perform its best. Most of the machine learning models accept numerical variables, thus preprocessing of categorical variables becomes a necessary step. The category variables can be divided as ordinal(order of category matters) and nominal(order of category does not matter).

In our case, we have nominal categorical variables- gender, ever_married, work_type, Residence_type, smoking_status and we our converting them to binary numerical values using below encoders from SciKit-learn library:

Label-encoding:

- Converted datatype of gender, ever_married,Residence_type from ‘object’ to ‘category’ datatype.
- Applied cat.codes function.

One-hot encoding:

- Applied get_dummies function on work_type, smoking_status categorical variables.

2. Normalizing data

In our dataset we have 3 numerical variables viz. Age, bmi, avg_glucose_level with different ranges of data as seen in the descriptive statistics table below.
Applying the MinMaxScaler function to shift the ranges to 0-1 as shown in table 'after normalization' below.

	age	bmi	avg_glucose_level
count	5109.000000	5109.000000	5109.000000
mean	43.229986	28.863300	106.140399
std	22.613575	7.699785	45.285004
min	0.080000	10.300000	55.120000
25%	25.000000	23.800000	77.240000
50%	45.000000	28.100000	91.880000
75%	61.000000	32.800000	114.090000
max	82.000000	97.600000	271.740000

Before normalization

	age	bmi	avg_glucose_level
count	5109.000000	5109.000000	5109.000000
mean	0.526733	0.212638	0.235529
std	0.276045	0.088199	0.209053
min	0.000000	0.000000	0.000000
25%	0.304199	0.154639	0.102114
50%	0.548340	0.203895	0.169698
75%	0.743652	0.257732	0.272228
max	1.000000	1.000000	1.000000

After normalization

3. Splitting data

This is an important step before implementing the machine learning models. We are splitting our data into 75% training and 25% testing. We will train the model using the training data and evaluate the performance of the model using the testing data.

Data Mining Models

1. KNN

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors[2]

2. SVM

Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. SVM maps training examples to points in space to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall[2].

3. **Decision Trees**

Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels[2].

4. **Random Forest**

The basic idea in random forests is to: 1. Draw multiple random samples, with replacement, from the data (this sampling approach is called the bootstrap). 2. Using a random subset of predictors at each stage, fit a classification (or regression) tree to each sample (and thus obtain a “forest”). 3. Combine the predictions/classifications from the individual trees to obtain improved predictions. Use voting for classification and averaging for prediction[2].

5. **Logistic Regression**

In logistic regression, we take two steps: the first step yields estimate of the propensities or probabilities of belonging to each class. In the binary case, we get an estimate of p = P (Y = 1), the probability of belonging to class 1 (which also tells us the probability of belonging to class 0). In the next step, we use a cutoff value on these probabilities to classify each case into one of the classes[2].

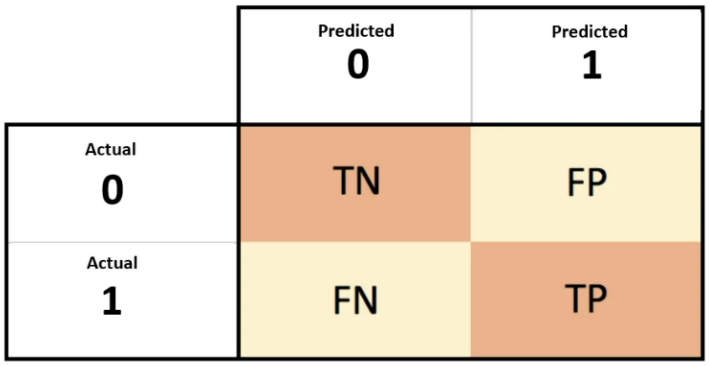
Performance Evaluation

This step is important to get answers to questions like:
How well is my model doing? Is it a useful model?
How can I improve its performance?

The metrics that are commonly used for classification problems are:

1. **Accuracy**: It is the most common metric used for classification problems.
   Accuracy = Number of correct predictions/Total number of predictions

2. **Confusion matrix**: It is a summary of prediction results for a classification problem. For a two-class problem, the confusion matrix is a very easy way to understand and gives insights not only on the errors but also the type of errors made by the classifier. With this breakdown, it overcomes the drawback of using classification accuracy alone!
   Confusion matrix looks like below:



TN (True Negatives): The count of outcomes that were originally negative and predictive negative.
TP (True Positives): The count of outcomes that were originally positive and predictive positive.
FP (False Positives): The count of outcomes that were originally negative but predictive positive. This error is also called a Type I error.
FN (False Negatives): The count of outcomes that were originally positive but predictive negative. This error is also called a Type II error.

3. **Precision**: Used to find the percentage of truly positive predictions out of total positive predictions.
   Precision = TP / (TP + FP)

4. **Recall**: Used to find the percentage of positive predictions out of all positive predictions. This is a crucial metric as our class of interest is 1 and we don't want to miss predicting a stroke which can be deadly too.
   Recall = TP / (TP + FN)

5. **F1-score**: It is the harmonic mean of Precision and Recall. It performs well on an imbalance dataset and a very useful metric for our problem.
   F1-score = 2 / ((1/Precision) + (1/Recall))

6. **ROC curve**: ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a plot of Precision vs 1-Recall

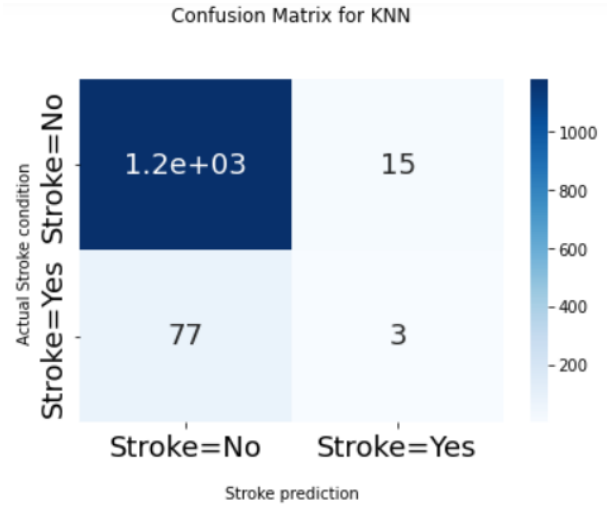
7. **AUC score**: AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

KNN

KNN was implemented using the KNeighborsClassifier() function of sklearn.neighbors in Python. After running for k from 1-40, we chose k = 3 for evaluating the performance of the model on the testing data.

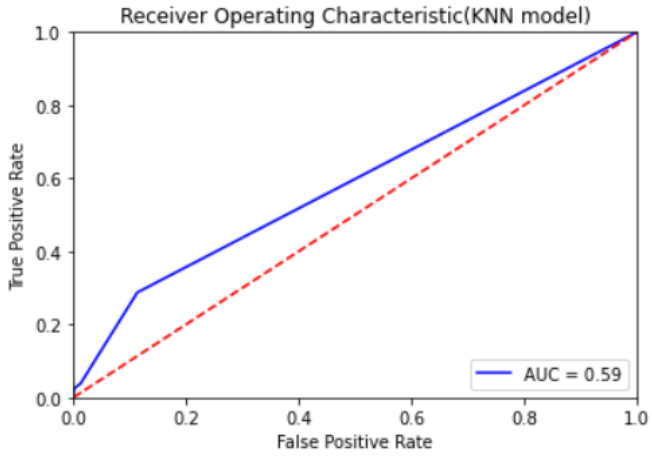
Below are the metrics obtained for KNN model with k = 3:

1. Confusion matrix



The above confusion matrix indicates that only 3 out of the total stroke cases are detected correctly.

2. ROC

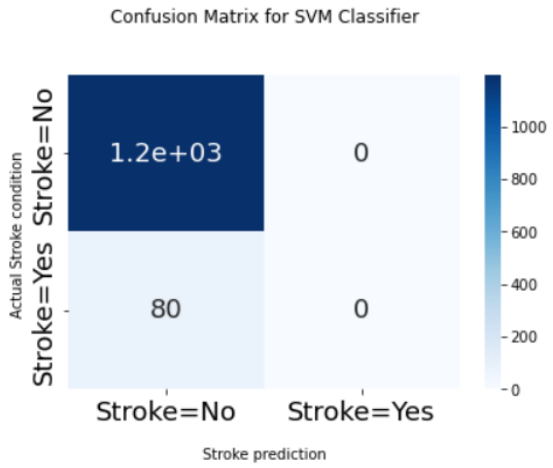


SVM

SVM was implemented using the SCV() function of sklearn.svm in Python.

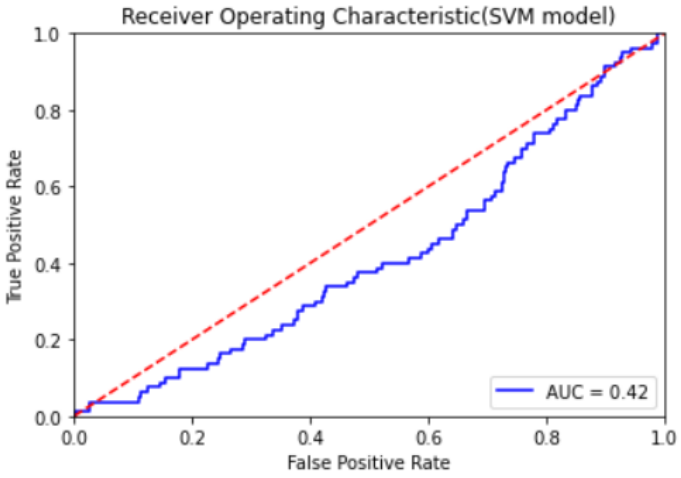
Below are the metrics obtained for SVM model:

1. Confusion matrix



The above confusion matrix indicates none out of the total stroke cases are detected correctly!

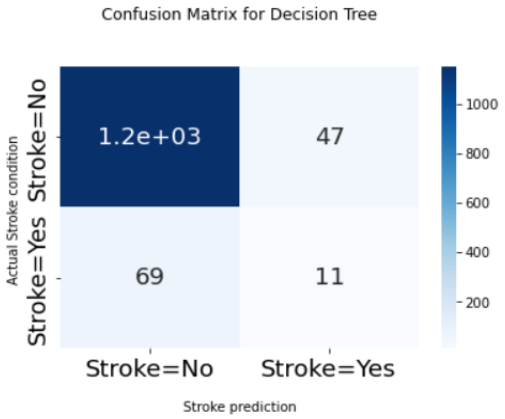
2. ROC



Decision Trees

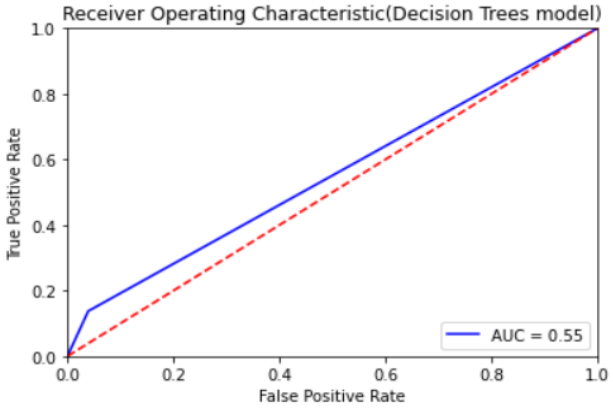
Decision Trees was implemented using the DecisionTreeClassifier() function of sklearn.tree. Below are the metrics obtained for Decision Trees model:

1. Confusion matrix



The above confusion matrix indicates that only 11 out of the total stroke cases are detected correctly.

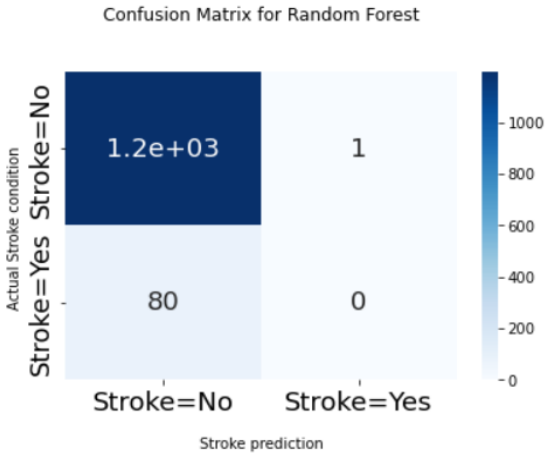
2. ROC



Random Forest

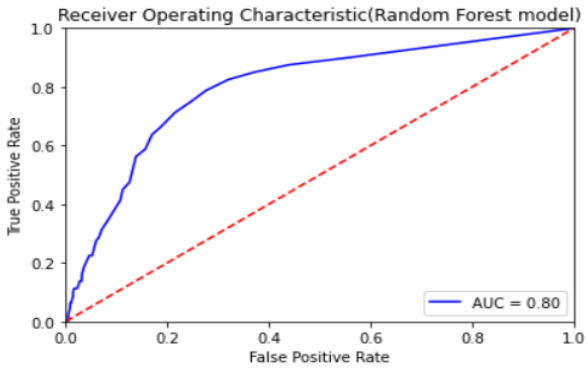
Random Forest was implemented using the RandomForestClassifier() function of sklearn.ensemble in Python. Below are the metrics obtained for Random Forest model:

1. Confusion matrix



The above confusion matrix indicates that none out of the total stroke cases are detected correctly!

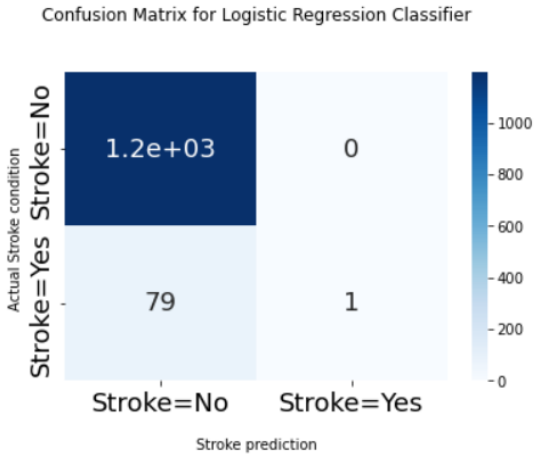
2. ROC



Logistic Regression

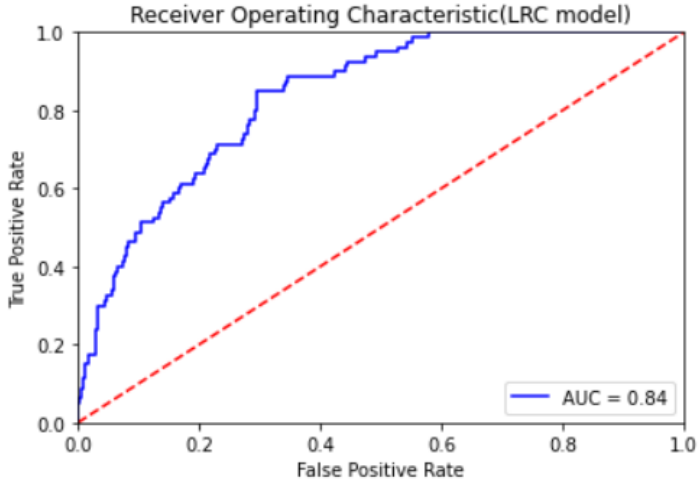
Logistic Regression was implemented using the LogisticRegression() function of sklearn.linear_model in Python. Below are the metrics obtained for Logistic Regression model:

1. Confusion matrix



The above confusion matrix indicates that only 1 out of the total stroke cases are detected correctly.

2. ROC



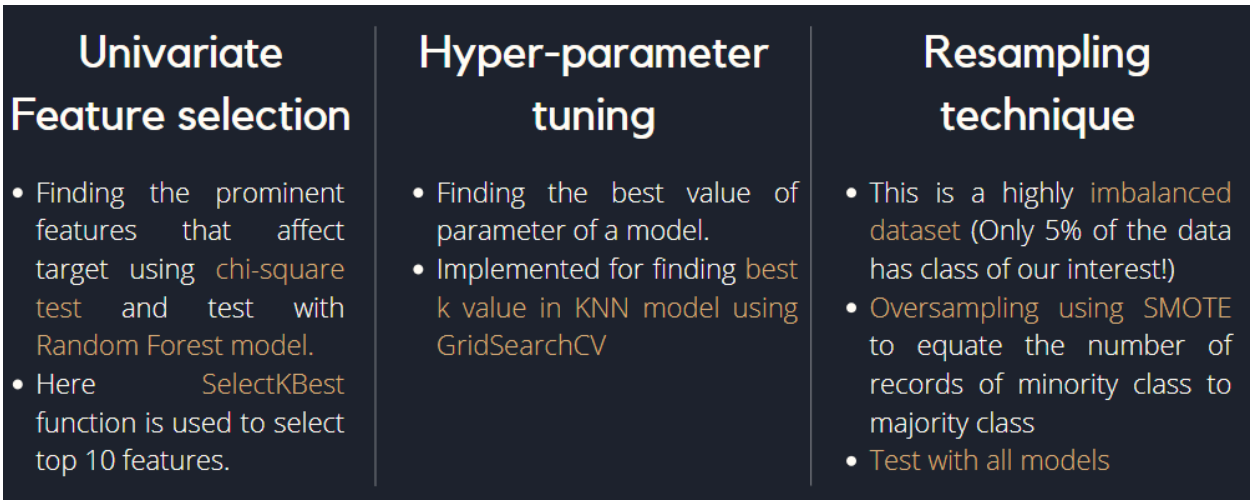
Model	KNN	SVM	Decision Tree	Random Forest	Logistic Regression
Accuracy	0.93	0.94	0.9	0.91	0.94
Precision	0.13	0	0.21	1	0
Recall	0.03	0	0.21	0.02	0
F1 score	0.05	0	0.21	0.03	0

AUC	0.56	0.47	0.58	0.8	0.85

Table 1: Performance before oversampling

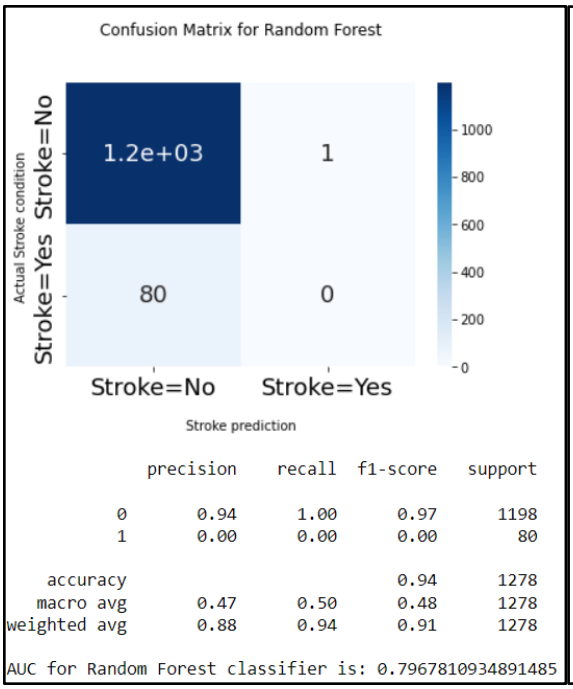
Inference: The above table indicates that the models are performing very poorly. We can conclude from the metric values that Accuracy is not a good measure of model performance in our case. Instead, Recall, F1-score and ROC curve are better indicators of the performance of our model. The poor results obtained are due to the highly imbalanced nature of the dataset which we will address in the next section.

Methods to improve performance of Implemented models[1]



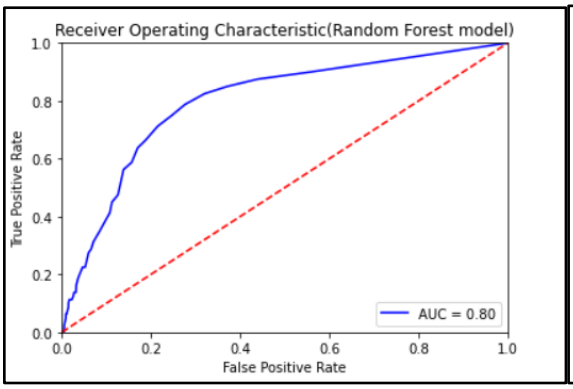
1. Univariate feature selection

Univariate feature selection works by selecting the best features based on univariate statistical tests. We compare each feature to the target variable, to see whether there is any statistically significant relationship between them. It is also called analysis of variance (ANOVA). When we analyze the relationship between one feature and the target variable, we ignore the other features. That is why it is called 'univariate'. Each feature has its test score.
Finally, all the test scores are compared, and the features with top scores will be selected.

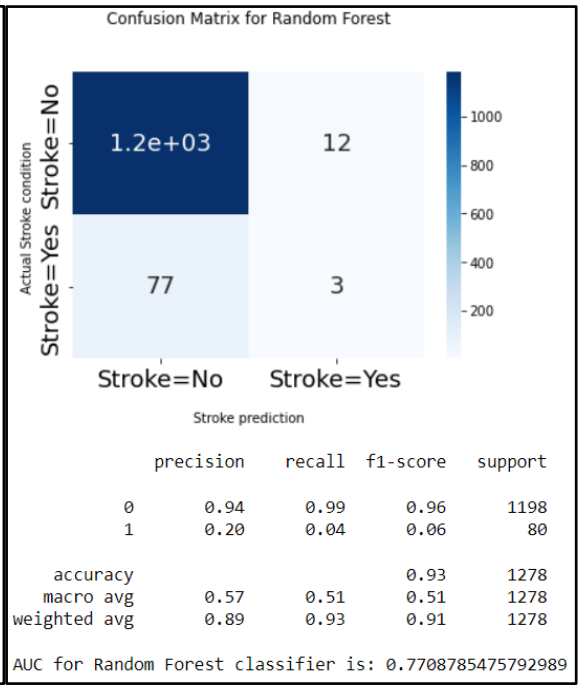


	precision	recall	f1-score	support
0	0.94	1.00	0.97	1198
1	0.00	0.00	0.00	80
accuracy			0.94	1278
macro avg	0.47	0.50	0.48	1278
weighted avg	0.88	0.94	0.91	1278

AUC for Random Forest classifier is: 0.7967810934891485

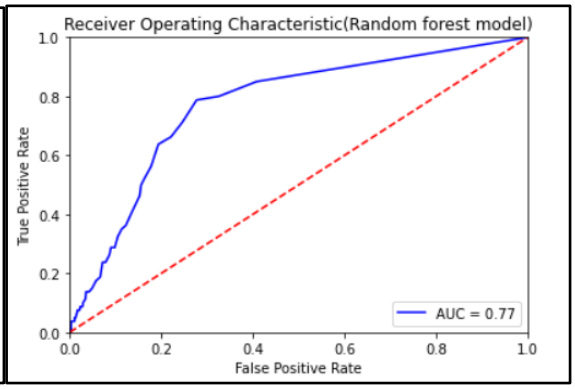


Before univariate feature selection



	precision	recall	f1-score	support
0	0.94	0.99	0.96	1198
1	0.20	0.04	0.06	80
accuracy			0.93	1278
macro avg	0.57	0.51	0.51	1278
weighted avg	0.89	0.93	0.91	1278

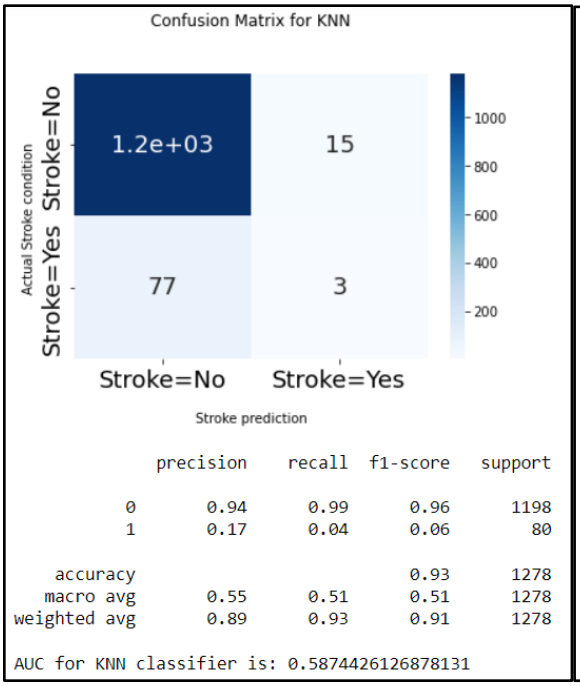
AUC for Random Forest classifier is: 0.7708785475792989



After univariate feature selection

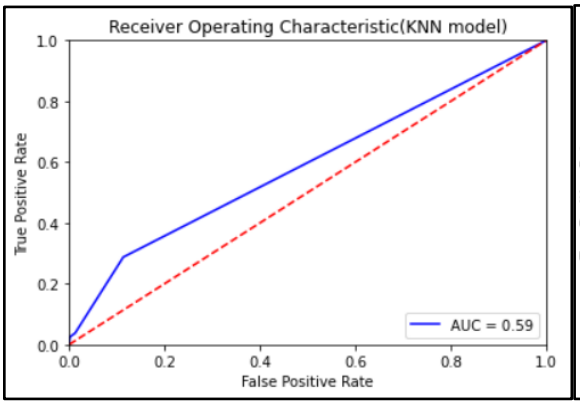
From the above results we can see that the F1-Score slightly improved but the ROC curve degraded. Hence, we will reject this method for our problem resolution.

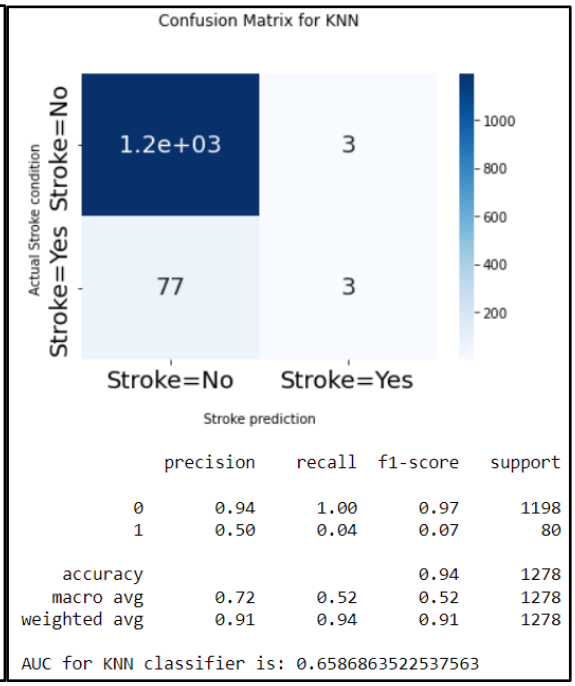
2. Hyper-parameter tuning



	precision	recall	f1-score	support
0	0.94	0.99	0.96	1198
1	0.17	0.04	0.06	80
accuracy			0.93	1278
macro avg	0.55	0.51	0.51	1278
weighted avg	0.89	0.93	0.91	1278

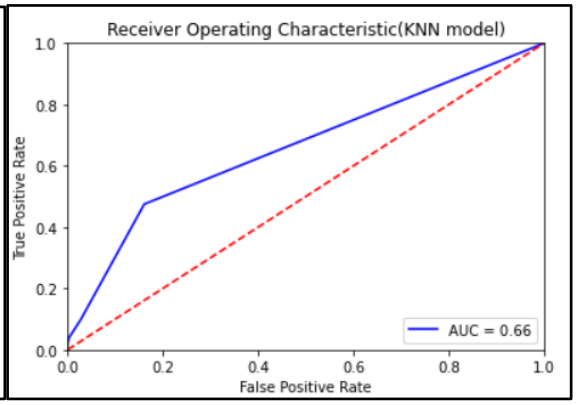
AUC for KNN classifier is: 0.5874426126878131





	precision	recall	f1-score	support
0	0.94	1.00	0.97	1198
1	0.50	0.04	0.07	80
accuracy			0.94	1278
macro avg	0.72	0.52	0.52	1278
weighted avg	0.91	0.94	0.91	1278

AUC for KNN classifier is: 0.6586863522537563



Before Hyper-parameter tuning

After Hyper-parameter tuning

From the above results we can see that the F1-Score and ROC curve have improved slightly but not greatly as desired. Hence, we will reject this method for our problem resolution.

3. Sampling technique

Oversampling technique used with all the testing models yielded below results.

Model	**KNN**	**SVM**	**Decision Tree**	**Random Forest**	**Logistic Regression**
Accuracy	0.88	0.8	0.87	0.92	0.79

Precision	0.82	0.76	0.86	0.89	0.76
Recall	0.97	0.87	0.88	0.96	0.84
F1 score	0.89	0.81	0.87	0.92	0.8
AUC	0.93	0.85	0.87	0.98	0.85

Table 2: Performance after oversampling

It can be seen from the above table that all the models performed exceptionally better than before sampling technique was applied. Random Forest performed the best with F1-Score of 92% and AUC of 98% followed by KNN with F1-Score of 89% and AUC of 93%. Thus, we choose sampling technique for improving out model performance for our problem resolution.

Project Results

- For all the implemented models like KNN, SVM, Logistic Regression, Decision Trees and Random Forest, accuracy metric was >90% which was impressive at first but after looking at the confusion matric, precision-recall, F1-scores, it was clear that accuracy was a misleading metric as most of the models were not predicting our class of interest (class = 1) correctly. The reason for such discrepancy was the highly imbalanced nature of our dataset with only 5% of the records belonging to class = 1.
- Thus, Precision-Recall, F1-score are good measures of model performance than Accuracy score which can be misleading for an imbalanced dataset.
- Out of the many different methods for improving model accuracy, we choose univariate feature selection and hyper-parameter tuning to be applied on the better performing models viz. KNN and Random Forest which did improve the performance but not greatly.
- We then decided to implement oversampling technique on all the models and obtained great improvement in the model performance.
- In conclusion, Random Forest model worked best for oversampled dataset with 92% F1-score and 98% AUC.
- Analyzing data (cleaning, handling imbalance) before splitting is a crucial step to achieve desired results

Impact of project outcomes

- The goal of the project was to successfully predict stroke in patients with the different ages, disease conditions, work styles, lifestyles etc. and we could predict the stroke with high confidence.

- This model can be used in healthcare field to not only predict stroke but also other disease conditions to take preventive action and save the lives of the patients.
- The model performed well after sampling technique was implemented. It is also possible in real-time to increase accuracy of prediction without oversampling but with higher number records continuously taken from hospitals, healthcare centers.
- Model performance can also be improved by:
  Other methods for feature selection like f-test, mutual_info_classif test, recursive feature selection, feature selection using SelectFromModel can be implemented and tested. Performance of other classification models like XG Boost, Extra Tree, Ada Boost classifiers can be explored and compared. Performance of advanced/hybrid sampling techniques can be evaluated.

References

[1]Feature selection using Python for classification problems | by Richard Liang | Towards Data Science
[2]https://en.wikipedia.org/