1. Executive Summary

Stroke is a leading cause of death in the United States and is a major cause of serious disability for adults. A stroke occurs when the blood supply to part of the brain is suddenly interrupted or when a blood vessel in the brain bursts, spilling blood into the spaces surrounding brain cells. Brain cells die when they no longer receive oxygen and nutrients from the blood or there is sudden bleeding into or around the brain. The symptoms of a stroke include sudden numbness or weakness, especially on one side of the body; sudden confusion or trouble speaking or understanding speech; sudden trouble seeing in one or both eyes; trouble with walking, dizziness, or loss of balance or coordination; or sudden severe headache with no known cause. Stroke risks are greatly reduced by making lifestyle changes.

In this project we trained five models for predicting the outcome of a stroke. We explored the five most common classification models to see which one produces the most accurate predictions. We chose classification models because our central problem is categorical. We are trying to predict if the stroke outcome is 0 or 1, that is if a given patient will have a stroke or not.The five models we chose are: Logistic Regression, K Nearest Neighbor, Random Forest, Decision Tree, and Gradient Boosting Classifier.

Before we started modeling, we had to employ a few data pre-processing methods to deal with a few concerning problems our data had, like missing values and class imbalance. We performed KNN imputation, binary encoding, one hot encoding, and oversampling as method to deal with the class imbalance.

2. The Dataset

The stroke prediction dataset contains both numerical and categorical variables that may contribute to knowing whether or not the patient will have a stroke. Our target (label) variable is "stroke". In this dataset, the target variable is coded as 1 for positive cases (having stroke) and 0 for negative cases (not having stroke).

There are 201 missing values in the"bmi" column. These missing values will be handled via knn imputation.
Dataset Name: Stroke Prediction Dataset
Dimensions: 5110 12
There are 12 Variables, they are:
1) **id:** unique identifier
2) **gender:** "Male", "Female" or "Other"
3) **age:** age of the patient
4) **hypertension**: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) **heart disease:** 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6) **evermarried:** "No" or "Yes"
7) **worktype:** "children", "Govt Job", "Neverworked", "Private" or "Self-employed"
8) **Residence Type**: "Rural" or "Urban"
9) **avg glucose level**: average glucose level in blood
10) **bmi**: body mass index
11) **smoking status**: "formerly smoked", "never smoked", "smokes" or "Unknown" *

12) **stroke:** 1 if the patient had a stroke or 0 if not
Our goal is to train five models for predicting the outcome of a stroke and then explore which of the five selected models performed the best using Accuracy score and F1-score as performance metric.

3. Possible Approaches (5 Classification Models)

Here are the five selected approaches for our central problem. They are used to solve our classification problem, they also all fall under Supervised Learning.

Logistic Regression is used to model a binary outcome, that is whether patients are likely to get a stroke or not, it models the logit transformed probability as a linear relationship with the predictor variables.

$$\text{logit}(p) = \log p/1\text{-}p = b_0 + b_1x_1 + b_2x_2$$

Decision Tree
Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.
Random Forest
Random forest is an ensemble machine learning algorithm. It operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.
K Nearest Neighbor
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.
Gradient Boosting Classifier
Gradient Boosting for classification.
Gradient Boosting builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function.

4. Selected Approach (Best Performance Method)

After accounting for the imbalanced data and retraining our 5 models, we built classification models that predict the outcome of a stroke. To evaluate how good the predictions made by the models are, we used Accuracy and F1 score as performance indicators.
The following output results in Accuracy and F1-score for each of the respective models.
(Code available in Technical Appendix)

```
Model 2 Performance (After Oversampling Method to handle Class Imbalance)
-----------------------------------------------------------------------

                    Logistic Regression Accuracy: 73.190%
```

```
F1-Score: 0.25678

K-Nearest Neighbors Accuracy: 85.845%
F1-Score: 0.11429

Decision Tree Accuracy: 92.433%
F1-Score: 0.15942

Random Forest Accuracy: 93.673%
F1-Score: 0.02020

Gradient Boosting Accuracy: 79.256%
F1-Score: 0.22439
```

Here, we can see Random Forest has the highest accuracy of 93.673% followed by Decision Tree with 92.433% accuracy. Here we refer to accuracy to measure performance because we solved the class imbalance problem.

5. Summary

We referred to Accuracy because it is the measure of all the correctly identified cases, it is the most intuitive performance measure and it is simply a ratio of correctly predicted observations to the total observations. One may think that, if we have high accuracy then our model is best. accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost the same. Therefore, we looked at other parameters to evaluate the performance of our model.
F1 Score is the weighted average of Precision and Recall. Therefore, F1 Score takes both false positives and false negatives into account. F1 is usually more useful than accuracy, especially in cases as such, when we have an uneven class distribution.

6. Conclusion

In this Stroke prediction dataset, we performed exploratory data analysis and found our data contains two problems, NA's for bmi and a much larger problem of class imbalance. We solve the missing value problem by using K-Nearest neighbor imputation; using the nearest neighbor of a given example to impute the values. For the class imbalance, we had very few positive labels compared to the negative labels.In most real-life classification problems, imbalanced class distribution exists and thus F1-score is a better metric to evaluate our model on.
All the columns had different ranges of values, we used a standard scaler from sklearn to make the columns all on the same ranges of values (mean of 0, and a var of 1).We then split 70% of data into the training set, and the other 30% into the test set. We ran the models prior to fixing the class imbalance, the output resulted in very high accuracies for each model and 0 for F1-score. The F1 score looks at both the positive class and the negative class, it's a balance of precision and recall , how well we are doing between classes and how well we are doing within classes. Model 1 performance was not great, so we continued with Model 2.

```
Model 1 Performance  ( prior fixing to class-imbalance)
-----------------------------------------------------

                    Logistic Regression Accuracy: 94.586%
                              F1-Score: 0.00000

                    K-Nearest Neighbors Accuracy: 94.260%
                              F1-Score: 0.00000

                          Decision Tree Accuracy: 90.868%
                              F1-Score: 0.19540

                          Random Forest Accuracy: 94.521%
                              F1-Score: 0.02326

                      Gradient Boosting Accuracy: 94.586%
                              F1-Score: 0.00000
```

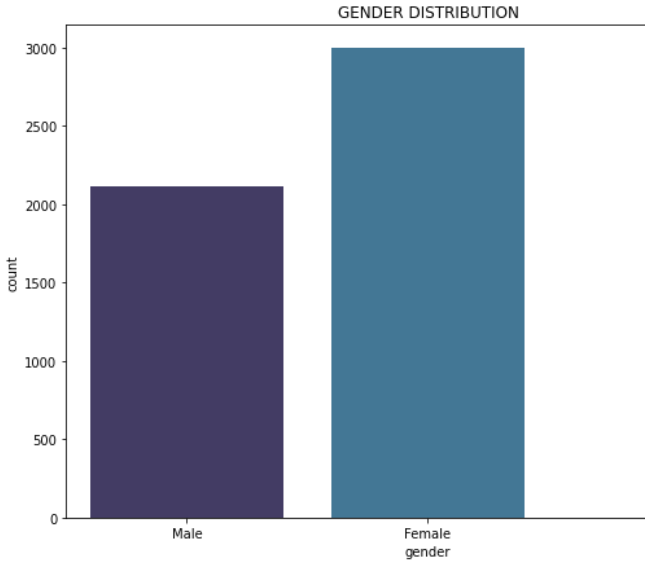
Because of the class imbalance, all of the predictions were made in the majority class i.e the negative class and no single prediction was made in the positive class. To get the F1 score higher, and fix the class-imbalance problem we used oversampling method. For oversampling, we took the positive class (minority class) and sampled with replacement as many times as we needed until we reached an equal number of negative and positive examples. The new positive class contained duplicates, without them we are back to the original imbalanced data. Throughout this process we learned that much of the work went into cleaning the data and preprocessing, as the data was not in perfect condition and rendered a poor model 1 performance. After we employed the oversampling method, we split again, but this time on the oversampled data. We trained the five classifiers on the new oversampled data to see which model performed the best. We analyzed the results with the Accuracy score, and F1- score from sklearn.metrics. Below are the results. As you can see, after we corrected the class imbalance via oversampling all the F1 - scores went up across the models, however some models performed worse than others. For instance Random Forest performed the worst with a F1 score of 0.02020. Although not high, the logistic regression had the highest F1 score of 0.25678. Interpreting our results in terms of the percentage of the data predicted correctly, Random Forest had the best accuracy of 93.673% followed by Decision Tree with 92.433% accuracy in stroke prediction.

```
Model 2 Performance (After Oversampling Method to handle Class Imbalance)
-------------------------------------------------------------------------

                    Logistic Regression Accuracy: 73.190%
                              F1-Score: 0.25678

                    K-Nearest Neighbors Accuracy: 85.845%
                              F1-Score: 0.11429

                          Decision Tree Accuracy: 92.433%
                              F1-Score: 0.15942
```

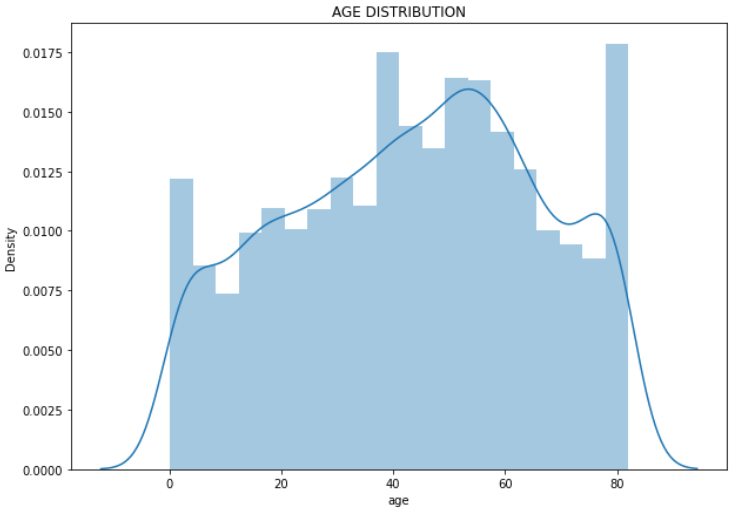
```
Random Forest Accuracy: 93.673%
F1-Score: 0.02020

Gradient Boosting Accuracy: 79.256%
F1-Score: 0.22439
```

Technical Appendix

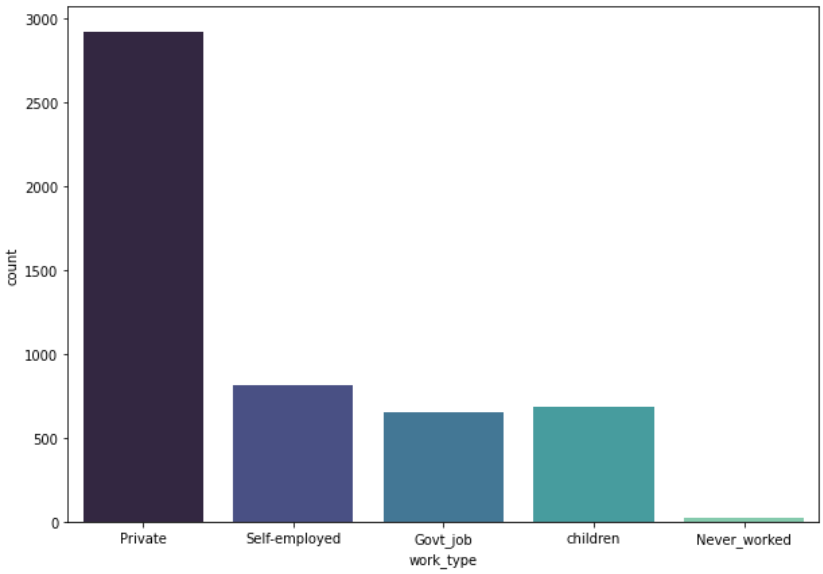
Exploratory Data Analysis Results



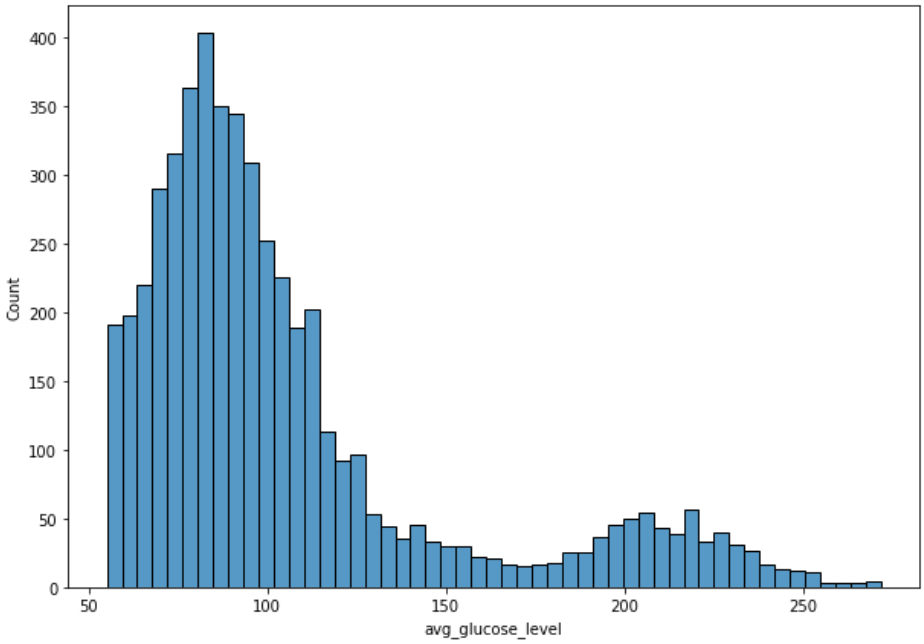
Gender Distribution in our dataset.



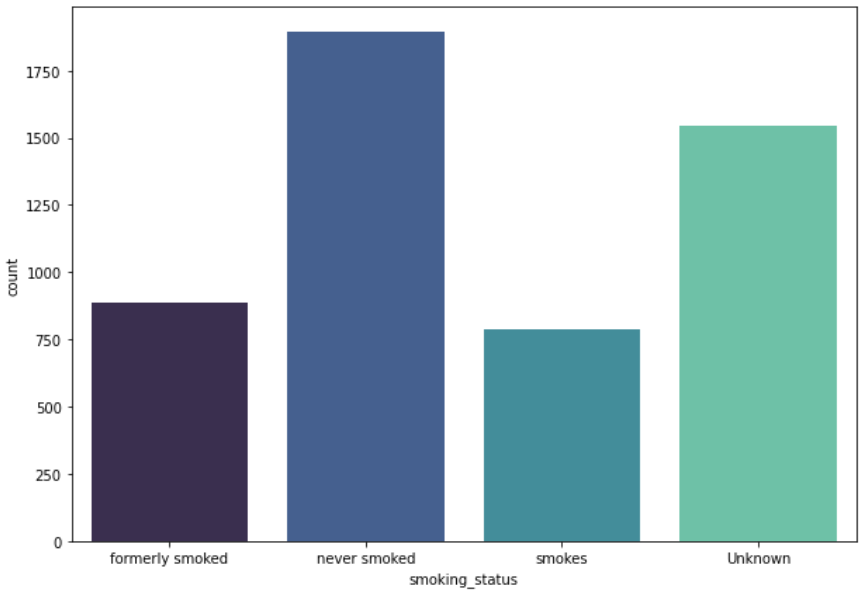
Distribution of Age across the sampled dataset. The distribution of age is approximately normal.



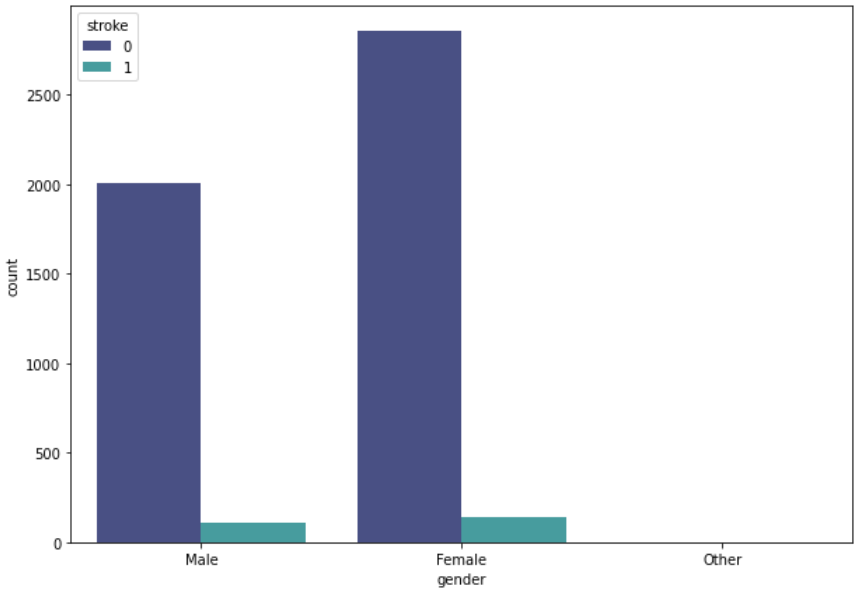
Count of the Work_type variable, as you can see, most of the sample works in the private sector.



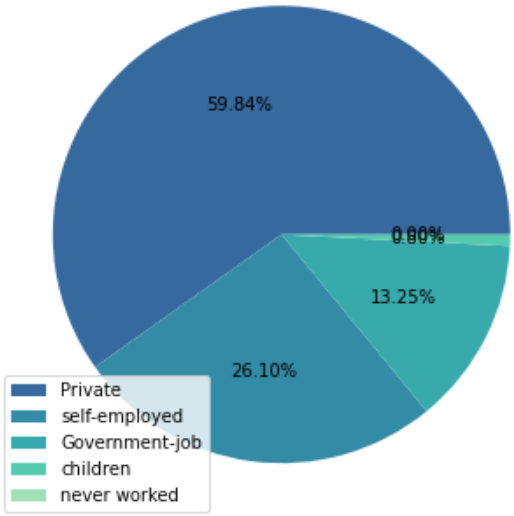
Histogram of the Average Glucose level
The majority of the participants have an average glucose level from 50 to 100.



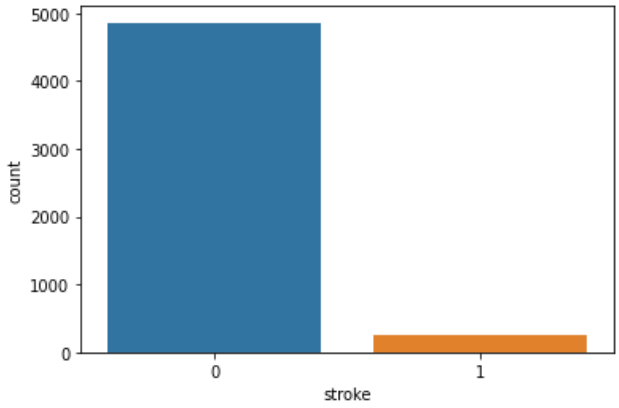
Countplot of the sampled participants smoking status. Most of the participants never smoked.



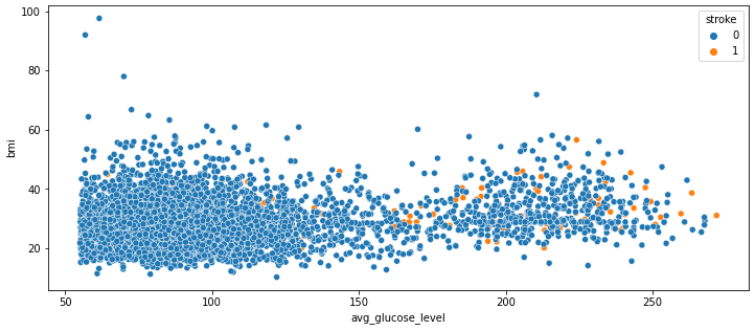
Breakdown of Stroke status based on gender.



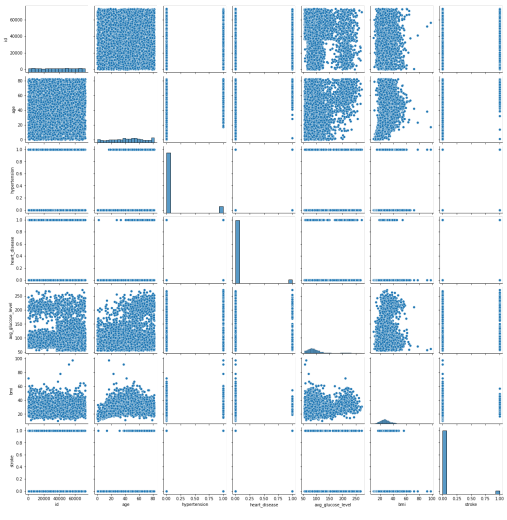
Breakdown of Stroke cases based on work_type. Most stroke cases occur among those in private work.



Our original imbalanced data. Notice the positive case is the minority class. We corrected this by using the oversampling method.



Plot displays stroke cases concentrated largey among higher glucose levels.



Pairplot of bmi, avg glucose level, heart disease, hypertension, age and stroke

Python Script output:

Importing Dependencies

```
[55]  1 import numpy as np
      2 import pandas as pd
      3 import matplotlib.pyplot as plt
      4
      5 from sklearn.model_selection import train_test_split
      6 # For dealing with missing data
      7 from sklearn.impute import KNNImputer
      8 # For Preprocessing
      9 from sklearn.preprocessing import StandardScaler
     10 # Importing Models from Scikit-Learn
     11 from sklearn.linear_model import LogisticRegression
     12 from sklearn.neighbors import KNeighborsClassifier
     13 from sklearn.tree import DecisionTreeClassifier
     14 from sklearn.svm import LinearSVC, SVC
     15 from sklearn.neural_network import MLPClassifier
     16 from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
     17 from xgboost import XGBClassifier
     18 from sklearn.metrics import accuracy_score, f1_score
     19 from sklearn.metrics import confusion_matrix
     20 import seaborn as sn
     21
     22 import warnings
     23 warnings.filterwarnings(action='ignore')
```

Importing Data

```
[10]  1 strokeData
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...	...
5105	18234	Female	80.0	1	0	Yes	Private	Urban	83.75	NaN	never smoked	0
5106	44873	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
5107	19723	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
5108	37544	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
5109	44679	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

5110 rows × 12 columns

```
[11]  1 strokeData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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)
memory usage: 479.2+ KB
```

Here, we noticed the BMI column has missing i.e NA data

Preprocessing

- drop id column
- binary encoding
- one-hot encoding
- knn imputation; use nearest neighbor of a given example to impute the values

[26]
```python
def onehot_encode(df, column):
    df = df.copy()
    dummies = pd.get_dummies(df[column], prefix=column)
    df = pd.concat([df, dummies], axis=1)
    df = df.drop(column, axis=1)
    return df
```

```python
def preprocess_inputs(df):
  df = df.copy()
  # dropping id column
  df = df.drop('id',axis=1)

  #binary encoding for ever_married & Residence_type
  df['ever_married'] = df['ever_married'].replace({'No': 0, 'Yes': 1})
  df['Residence_type'] = df['Residence_type'].replace({'Rural': 0, 'Urban': 1})

  #One-hot encoding
  for column in ['gender', 'work_type', 'smoking_status']:
        df = onehot_encode(df, column=column)

  #splitting the data
  y = df['stroke']
  X = df.drop('stroke', axis=1)
  #train - test split
  X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True, random_state=1)

  # Now data is in numeric form & ready for knn imputation for the missing values in bmi
  imputer = KNNImputer()
  imputer.fit(X_train)
  X_train = pd.DataFrame(imputer.transform(X_train), index=X_train.index, columns=X_train.columns)
  X_test = pd.DataFrame(imputer.transform(X_test), index=X_test.index, columns=X_test.columns)

  # Scalig X
  scaler = StandardScaler()
  scaler.fit(X_train)
  X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=X_train.columns)
  X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=X_test.columns)
  return X_train, X_test, y_train, y_test
```

```
[43]  1 X_train, X_test, y_train, y_test = preprocess_inputs(strokeData)
```

```
[44]  1 X_train
```

	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	gender_Female	gender_Male	gender_Other	work_type_Govt_job	work_type_Never_worked	wor
4313	0.608843	-0.330374	-0.238161	0.719002	0.994146	0.638258	0.010074	-1.174021	1.174021	0.0	-0.383679	-0.069103	
376	-1.886989	-0.330374	-0.238161	-1.390817	-1.005888	-0.386663	-1.754554	0.851774	-0.851774	0.0	-0.383679	-0.069103	
4913	-1.036663	-0.330374	-0.238161	-1.390817	0.994146	-0.466196	-1.105794	-1.174021	1.174021	0.0	-0.383679	-0.069103	
1791	-1.347974	-0.330374	-0.238161	-1.390817	-1.005888	0.697358	-1.404223	-1.174021	1.174021	0.0	-0.383679	-0.069103	
2166	-0.680878	-0.330374	-0.238161	0.719002	-1.005888	1.401291	-0.236455	-1.174021	1.174021	0.0	-0.383679	-0.069103	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2895	0.119639	-0.330374	-0.238161	0.719002	0.994146	0.704389	0.036024	-1.174021	1.174021	0.0	-0.383679	-0.069103	
2763	0.075165	-0.330374	-0.238161	0.719002	-1.005888	-0.922303	0.386355	0.851774	-0.851774	0.0	-0.383679	-0.069103	
905	-0.547458	-0.330374	-0.238161	0.719002	-1.005888	-0.647013	0.853462	0.851774	-0.851774	0.0	-0.383679	-0.069103	
3980	0.075165	-0.330374	-0.238161	0.719002	-1.005888	2.469274	3.370652	0.851774	-0.851774	0.0	-0.383679	-0.069103	
235	1.098047	-0.330374	-0.238161	0.719002	-1.005888	2.314602	1.333545	0.851774	-0.851774	0.0	-0.383679	-0.069103	

3577 rows × 19 columns

```
[25]  1 {column: len(X[column].unique()) for column in X.select_dtypes('object').columns}
```

```
{'gender': 3, 'smoking_status': 4, 'work_type': 5}
```

```
1 X_train.isna().sum()
```

```
age                              0
hypertension                     0
heart_disease                    0
ever_married                     0
Residence_type                   0
avg_glucose_level                0
bmi                              0
gender_Female                    0
gender_Male                      0
gender_Other                     0
work_type_Govt_job               0
work_type_Never_worked           0
work_type_Private                0
work_type_Self-employed          0
work_type_children               0
smoking_status_Unknown           0
smoking_status_formerly smoked   0
smoking_status_never smoked      0
smoking_status_smokes            0
dtype: int64
```

```
[45]  1 X_train.mean()
```

```
age                              -2.991426e-16
hypertension                      3.580213e-16
heart_disease                     5.517439e-16
ever_married                      4.669951e-16
Residence_type                   -2.784714e-16
avg_glucose_level                -1.364423e-16
bmi                              -3.707158e-16
gender_Female                    -4.224248e-16
gender_Male                       4.224248e-16
gender_Other                      0.000000e+00
work_type_Govt_job               -1.158642e-16
work_type_Never_worked           -4.501105e-16
work_type_Private                -4.332260e-16
work_type_Self-employed           1.114258e-17
work_type_children               -1.655868e-16
smoking_status_Unknown           -3.681086e-17
smoking_status_formerly smoked    4.255286e-16
smoking_status_never smoked       5.366440e-16
smoking_status_smokes             5.943744e-17
dtype: float64
```

```
[46]  1 X_train.var()
```

```
age                               1.00028
hypertension                      1.00028
heart_disease                     1.00028
ever_married                      1.00028
Residence_type                    1.00028
avg_glucose_level                 1.00028
bmi                               1.00028
gender_Female                     1.00028
gender_Male                       1.00028
gender_Other                      0.00000
work_type_Govt_job                1.00028
work_type_Never_worked            1.00028
work_type_Private                 1.00028
work_type_Self-employed           1.00028
work_type_children                1.00028
smoking_status_Unknown            1.00028
smoking_status_formerly smoked    1.00028
smoking_status_never smoked       1.00028
smoking_status_smokes             1.00028
dtype: float64
```

Training The Models

- Logistic Regression
- K Nearest Neighbor
- Decision Tree
- Random Forest
- Gradient Boosting Classifier

```
[47]  1 models = {
      2     "                    Logistic Regression": LogisticRegression(),
      3     "                    K-Nearest Neighbors": KNeighborsClassifier(),
      4     "                          Decision Tree": DecisionTreeClassifier(),
      5     "                          Random Forest": RandomForestClassifier(),
      6     "                                XGBoost": XGBClassifier(eval_metric='mlogloss'),
      7 }
      8 for name, model in models.items():
      9     model.fit(X_train, y_train)
     10     print(name + " trained.")
```

```
                    Logistic Regression trained.
                    K-Nearest Neighbors trained.
                          Decision Tree trained.
                          Random Forest trained.
                                XGBoost trained.
```

Model Performance 1

```
print("Model Performance\n-----------------")
for name, model in models.items():
    y_pred = model.predict(X_test)
    print(
        "\n" + name + " Accuracy: {:.3f}%\n\t\t\t\t\t       F1-Score: {:.5f}"\
        .format(accuracy_score(y_test, y_pred) * 100, f1_score(y_test, y_pred))
    )
```

```
Model Performance
-----------------

                  Logistic Regression Accuracy: 94.586%
                                      F1-Score: 0.00000

                  K-Nearest Neighbors Accuracy: 94.260%
                                      F1-Score: 0.00000

                        Decision Tree Accuracy: 90.868%
                                      F1-Score: 0.15663

                        Random Forest Accuracy: 94.586%
                                      F1-Score: 0.02353

                              XGBoost Accuracy: 94.521%
                                      F1-Score: 0.00000

      Gradient Boosting Accuracy: 94.586%
                          F1-Score: 0.00000
```

```
[50]  y_train.value_counts()
```

```
0    3411
1     166
Name: stroke, dtype: int64
```

Problem- encountered: Class-Imbalance

Solution for Class-Imbalance: Oversampling

```
[ ]  oversampled_data.query("stroke == 1")
```

	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	gender_Female	gender_Male	gender_Other	work_type_Govt_job	work_type_Never_worked	w
232	1.320413	-0.330374	-0.238161	0.719002	0.994146	-0.763896	0.697760	0.851774	-0.851774	0.0	-0.383679	-0.069103	
125	1.275940	-0.330374	-0.238161	0.719002	0.994146	0.564657	-0.080752	0.851774	-0.851774	0.0	2.606344	-0.069103	
130	1.320413	3.026868	-0.238161	0.719002	-1.005888	1.854980	0.970239	0.851774	-0.851774	0.0	-0.383679	-0.069103	
186	0.875682	-0.330374	4.198834	0.719002	0.994146	0.241252	0.710735	-1.174021	1.174021	0.0	-0.383679	-0.069103	
65	1.542778	3.026868	-0.238161	-1.390817	0.994146	0.160401	-0.651661	-1.174021	1.174021	0.0	-0.383679	-0.069103	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
243	1.098047	3.026868	4.198834	0.719002	0.994146	3.115424	1.489247	0.851774	-0.851774	0.0	-0.383679	-0.069103	
155	1.498305	-0.330374	4.198834	0.719002	-1.005888	2.764996	1.022140	0.851774	-0.851774	0.0	-0.383679	-0.069103	
15	0.297531	3.026868	-0.238161	0.719002	-1.005888	1.355592	0.243628	0.851774	-0.851774	0.0	-0.383679	-0.069103	
144	1.275940	3.026868	-0.238161	0.719002	0.994146	2.550345	0.126851	0.851774	-0.851774	0.0	-0.383679	-0.069103	
235	1.098047	-0.330374	-0.238161	0.719002	-1.005888	2.314602	1.333545	0.851774	-0.851774	0.0	-0.383679	-0.069103	

166 rows × 20 columns

```
oversampled_data = pd.concat([X_train, y_train], axis=1).copy()
num_samples = y_train.value_counts()[0] - y_train.value_counts()[1]
new_samples = oversampled_data.query("stroke == 1").sample(num_samples, replace=True, random_state=1)

oversampled_data = pd.concat([oversampled_data, new_samples], axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)

y_train_oversampled = oversampled_data['stroke']
X_train_oversampled = oversampled_data.drop('stroke', axis=1)


```

```
oversampled_data
# contains duplicates, without them we are back to the original imbalanced data
```

	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	gender_Female	gender_Male	gender_Other	work_type_Govt_job	work_type_Never_worked	w
0	0.030692	-0.330374	-0.238161	0.719002	-1.005888	0.604862	2.073132	0.851774	-0.851774	0.0	2.606344	-0.069103	
1	-1.481394	-0.330374	-0.238161	-1.390817	0.994146	-0.769169	-0.470009	-1.174021	1.174021	0.0	-0.383679	-0.069103	
2	-0.236146	-0.330374	-0.238161	0.719002	-1.005888	-0.536721	0.542057	0.851774	-0.851774	0.0	-0.383679	-0.069103	
3	-1.703759	-0.330374	-0.238161	-1.390817	-1.005888	-0.177285	-1.559926	-1.174021	1.174021	0.0	-0.383679	-0.069103	
4	1.098047	-0.330374	4.198834	0.719002	0.994146	2.595164	0.373380	-1.174021	1.174021	0.0	-0.383679	-0.069103	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6817	1.142520	3.026868	-0.238161	-1.390817	0.994146	-0.653824	-0.054802	-1.174021	1.174021	0.0	-0.383679	-0.069103	
6818	-0.058254	-0.330374	-0.238161	0.719002	-1.005888	-0.489924	-0.470009	-1.174021	1.174021	0.0	-0.383679	-0.069103	
6819	0.742262	-0.330374	4.198834	0.719002	0.994146	-0.302956	0.892388	-1.174021	1.174021	0.0	-0.383679	-0.069103	
6820	-0.547458	-0.330374	-0.238161	0.719002	0.994146	0.063950	0.526487	-1.174021	1.174021	0.0	-0.383679	-0.069103	
6821	0.519897	3.026868	4.198834	0.719002	0.994146	2.300101	1.424371	0.851774	-0.851774	0.0	-0.383679	-0.069103	

6822 rows × 20 columns

Checking the class distribution after handling class imbalance data

```
[75]  oversampled_data['stroke'].value_counts()
```

```
1    3411
0    3411
Name: stroke, dtype: int64
```

```
[76]  y_train_oversampled.value_counts()
```

```
1    3411
0    3411
Name: stroke, dtype: int64
```

Re-Training the models after accounting for class-imbalance

```
[77]  1 models = {
      2     "                  Logistic Regression": LogisticRegression(),
      3     "                  K-Nearest Neighbors": KNeighborsClassifier(),
      4     "                        Decision Tree": DecisionTreeClassifier(),
      5     "                        Random Forest": RandomForestClassifier(),
      6     "                              XGBoost": XGBClassifier(eval_metric='mlogloss'),
      7 }
      8 for name, model in models.items():
      9     model.fit(X_train_oversampled, y_train_oversampled)
     10     print(name + " trained.")
```

```
                  Logistic Regression trained.
                  K-Nearest Neighbors trained.
                        Decision Tree trained.
                        Random Forest trained.
                              XGBoost trained.
```

Model Performance 2 after handling Class Imbalance

```
[80]  1 print("Model Performance\n-----------------")
      2 for name, model in models.items():
      3     y_pred = model.predict(X_test)
      4     print(
      5         "\n" + name + " Accuracy: {:.3f}%\n\t\t\t\t       F1-Score: {:.5f}"\
      6         .format(accuracy_score(y_test, y_pred) * 100, f1_score(y_test, y_pred))
      7     )
```

```
Model Performance
-----------------

                  Logistic Regression Accuracy: 73.190%
                                      F1-Score: 0.25678

                  K-Nearest Neighbors Accuracy: 85.845%
                                      F1-Score: 0.11429

                        Decision Tree Accuracy: 92.433%
                                      F1-Score: 0.15942

                        Random Forest Accuracy: 93.673%
                                      F1-Score: 0.02020

                              XGBoost Accuracy: 77.299%
                                      F1-Score: 0.24675
```

```
Gradient Boosting Accuracy: 79.256%
                    F1-Score: 0.22439
```