Dataset - Brain Stroke Risk Prediction

J.A.P.G. Marthris - COMScDS221P-015

Machine Learning Assignment - Section 02

Introduction th the dataset & familiarization

I have selected the Brain Stroke prediction dataset from Kaggle.com. Brain strokes, also known as cerebrovascular accidents (CVAs), are a significant global health concern characterized by a sudden disruption in blood supply to the brain. They are a leading cause of disability and death, with millions of new cases each year, affecting people of all ages. Strokes come in two main types: ischemic, caused by blood vessel blockages, and hemorrhagic, resulting from blood vessel ruptures. Immediate medical attention is crucial, as early treatment can minimize brain damage and improve recovery chances. Preventative measures, including lifestyle changes and risk factor management, are vital in reducing the global burden of strokes.

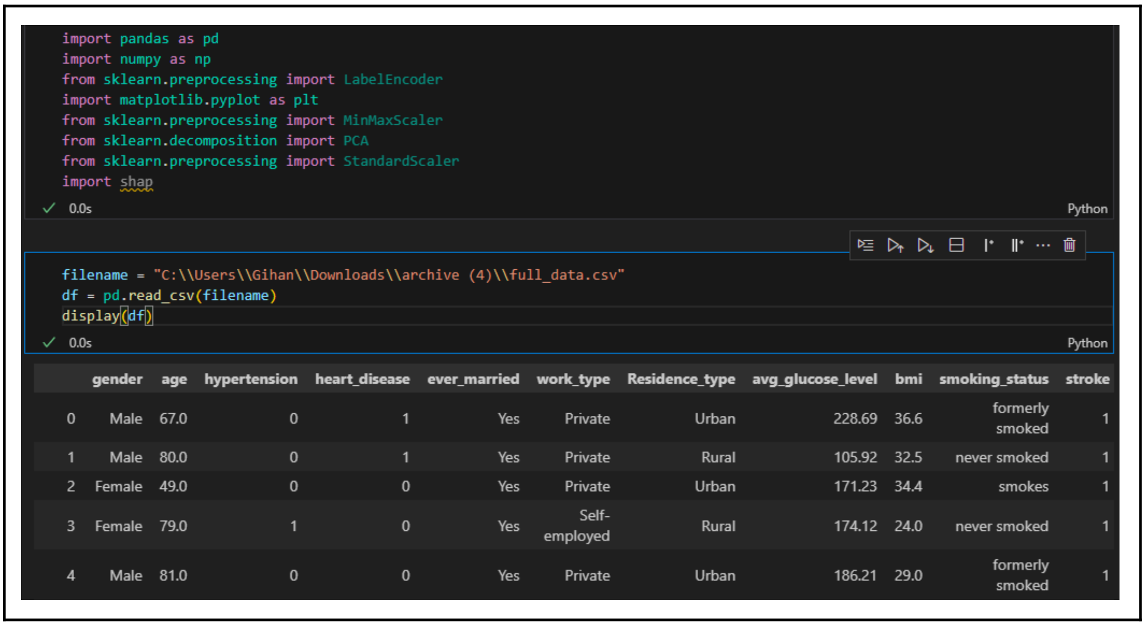
This dataset contains 11 attributes which included patients data of age, medical history, living behaviour etc.

Attribute	**Description**
Gender	"Male", "Female"
Age	age of the patient
Hypertension	0 if the patient doesn't have hypertension, 1 if the patient has hypertension
Heart Disease	0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
Ever Married	"No" or "Yes"
Work Type	"children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
Residence Type	"Rural" or "Urban"
Average Glucose Level	Average glucose level in blood
BMI	body mass index
Smoking Status	"formerly smoked", "never smoked", "smokes" or "Unknown"
Stroke	1 if the patient had a stroke or 0 if not *Note: "Unknown" in smoking_status means that the information is unavailable for this patient

Data Preprocessing

I have followed the most frequently used data preprocessing methods in order to achieve best outcome from the dataset.

1.Loading the necessary Python libraries & dataset to the Jupiter Notebook environment.

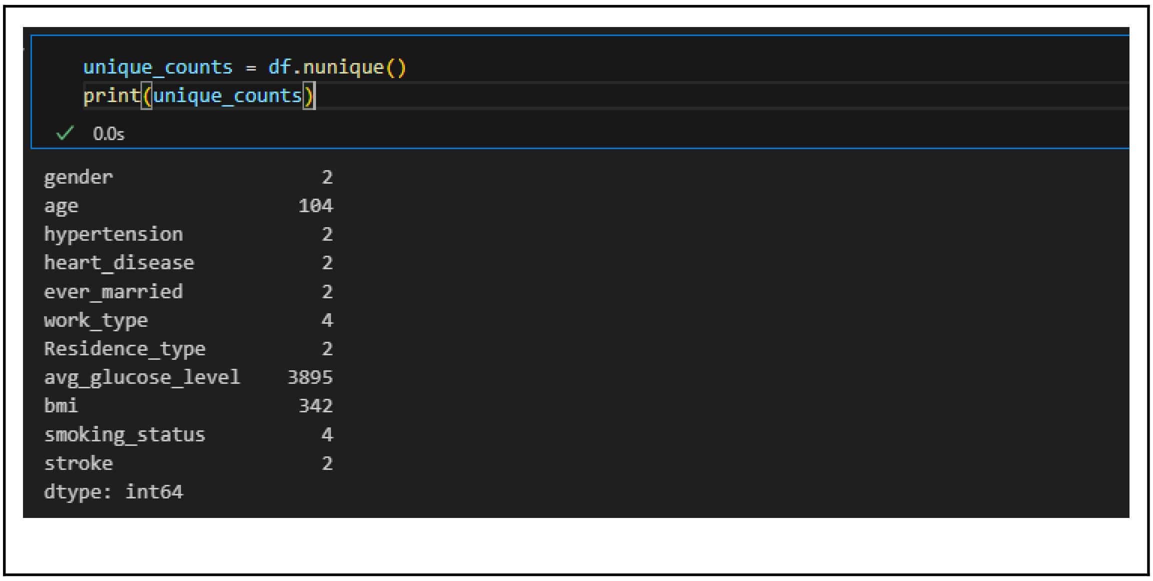


```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import shap
```

```
filename = "C:\\Users\\Gihan\\Downloads\\archive (4)\\full_data.csv"
df = pd.read_csv(filename)
display(df)
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
2	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
3	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
4	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1

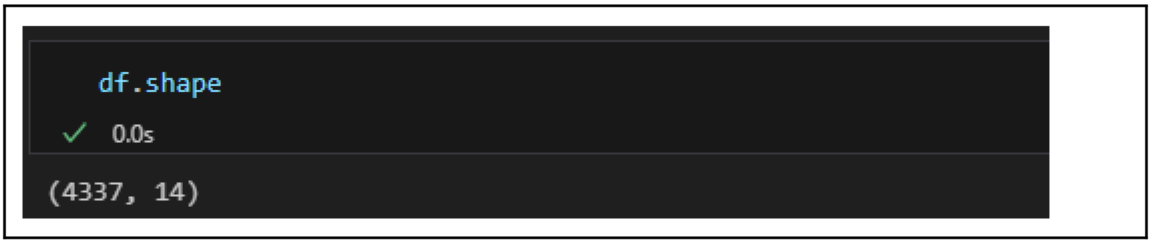
2. Checked the unique elements in the each variable



```
unique_counts = df.nunique()
print(unique_counts)
```

```
gender                  2
age                   104
hypertension            2
heart_disease           2
ever_married            2
work_type               4
Residence_type          2
avg_glucose_level    3895
bmi                   342
smoking_status          4
stroke                  2
dtype: int64
```

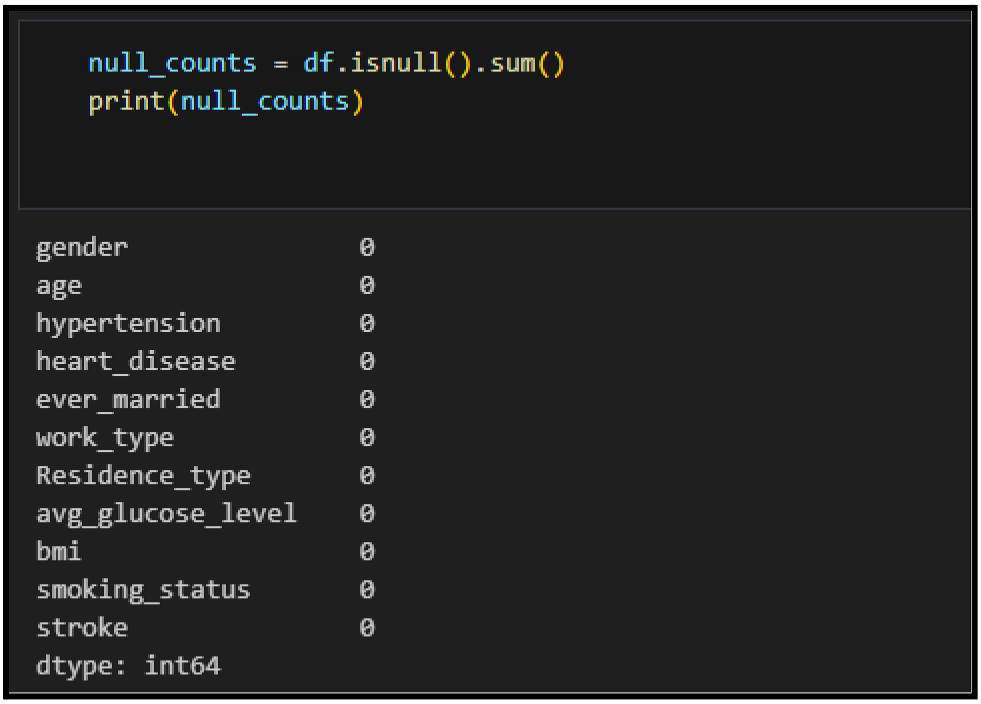
3. Checked the dimension of the dataset



```
df.shape
```

```
(4337, 14)
```

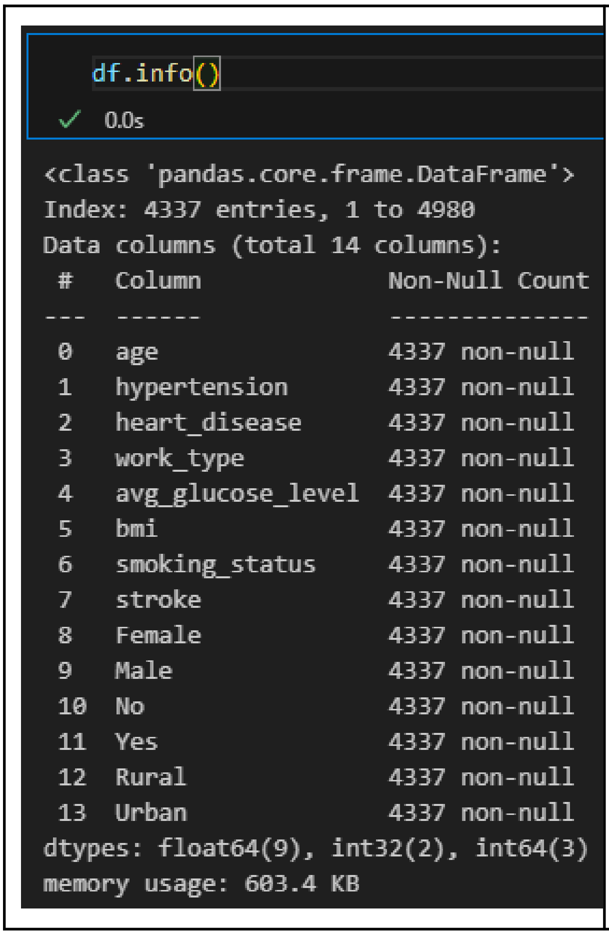
4.Check if there are any null values in the dataset



```
null_counts = df.isnull().sum()
print(null_counts)
```

```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

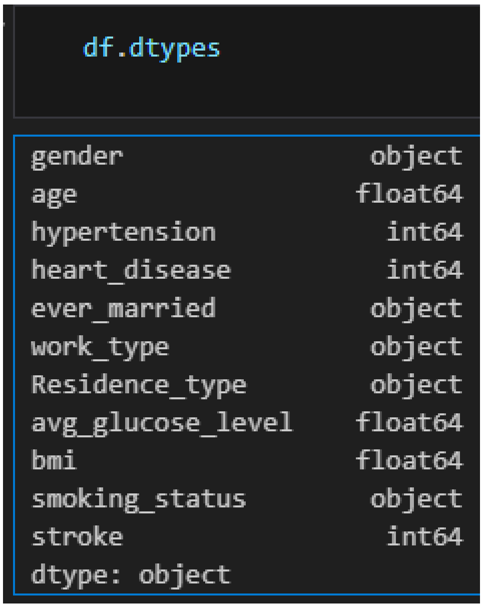
5. Checking the available data types in the dataset



```
df.info()
```

✓ 0.0s

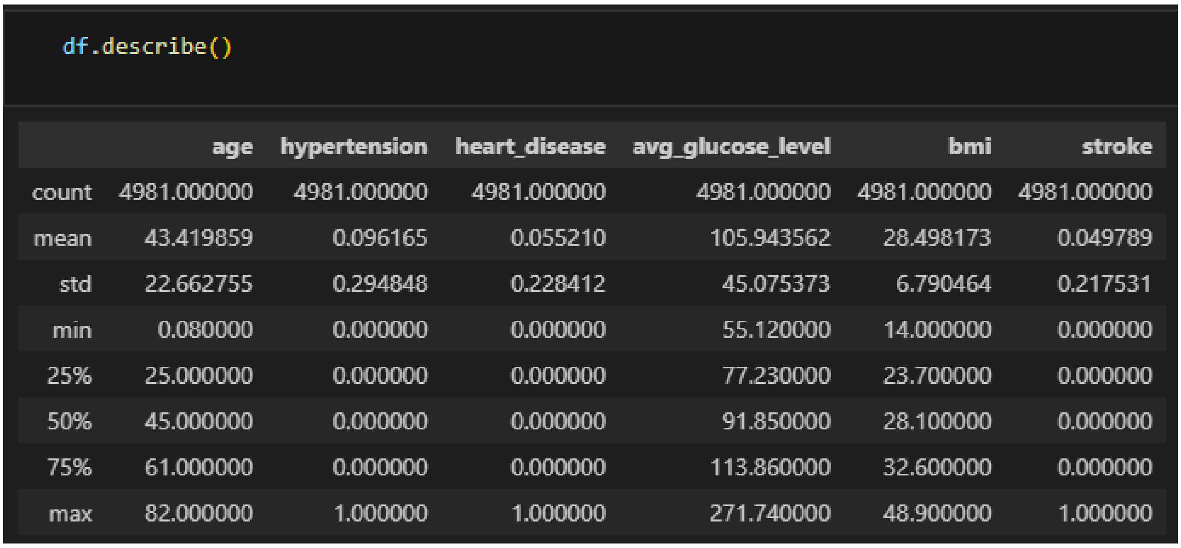
```
<class 'pandas.core.frame.DataFrame'>
Index: 4337 entries, 1 to 4980
Data columns (total 14 columns):
 #   Column             Non-Null Count
---  ------             --------------
 0   age                4337 non-null
 1   hypertension       4337 non-null
 2   heart_disease      4337 non-null
 3   work_type          4337 non-null
 4   avg_glucose_level  4337 non-null
 5   bmi                4337 non-null
 6   smoking_status     4337 non-null
 7   stroke             4337 non-null
 8   Female             4337 non-null
 9   Male               4337 non-null
 10  No                 4337 non-null
 11  Yes                4337 non-null
 12  Rural              4337 non-null
 13  Urban              4337 non-null
dtypes: float64(9), int32(2), int64(3)
memory usage: 603.4 KB
```



```
df.dtypes
```

```
gender                object
age                  float64
hypertension           int64
heart_disease          int64
ever_married          object
work_type             object
Residence_type        object
avg_glucose_level    float64
bmi                  float64
smoking_status        object
stroke                 int64
dtype: object
```

6. Checked the summery of descriptive statistics od the dataset

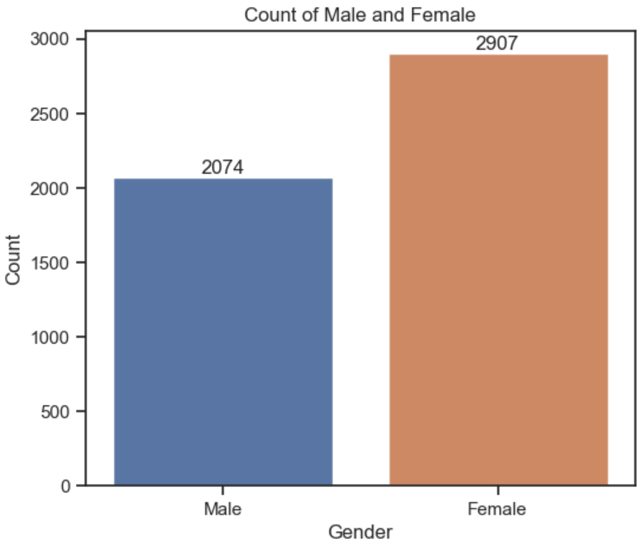


```
df.describe()
```

```
               age  hypertension  heart_disease  avg_glucose_level          bmi       stroke
count  4981.000000   4981.000000    4981.000000        4981.000000  4981.000000  4981.000000
mean     43.419859      0.096165       0.055210         105.943562    28.498173     0.049789
std      22.662755      0.294848       0.228412          45.075373     6.790464     0.217531
min       0.080000      0.000000       0.000000          55.120000    14.000000     0.000000
25%      25.000000      0.000000       0.000000          77.230000    23.700000     0.000000
50%      45.000000      0.000000       0.000000          91.850000    28.100000     0.000000
75%      61.000000      0.000000       0.000000         113.860000    32.600000     0.000000
max      82.000000      1.000000       1.000000         271.740000    48.900000     1.000000
```

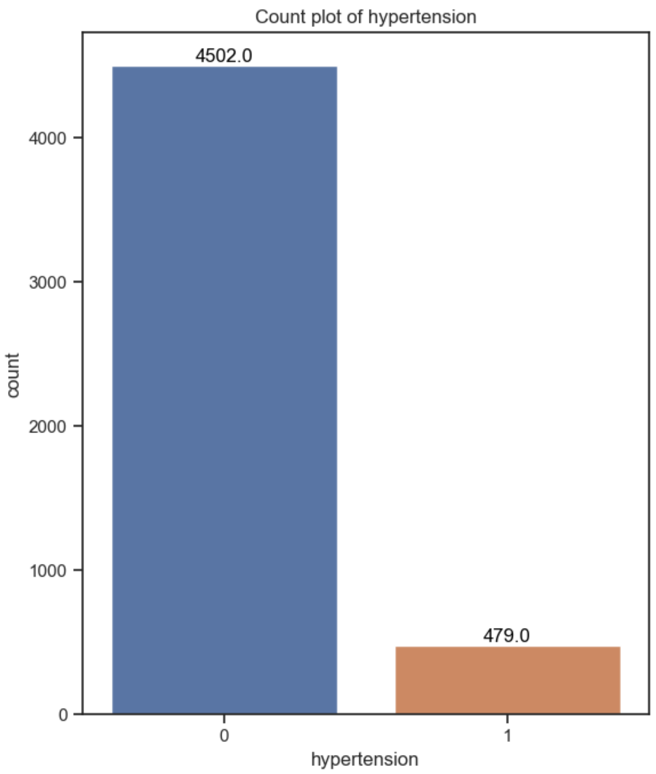
Based on the generated output mean age is 43 years and mean average glucose level is 105.9. Furthermore mean BMI index is 28.4.

Using count plot hereby we analysed the count of each variables visually and presented. These are categorical variables.

Gender ● Number of Female patients are higher particupated than Male patients.	

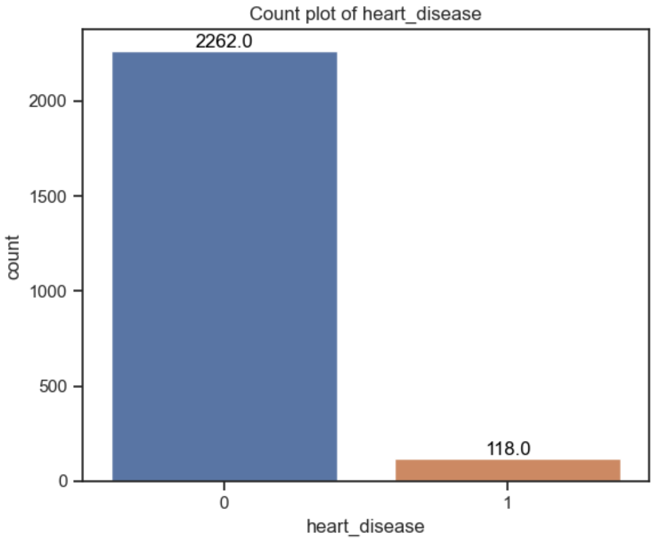
Hypertension

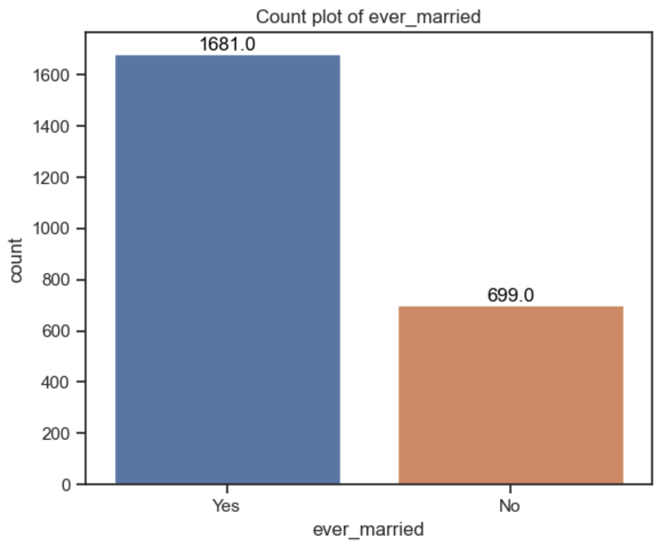
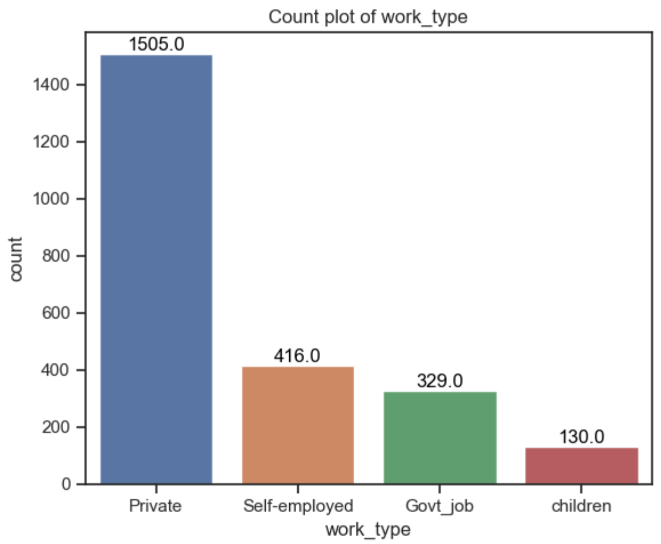
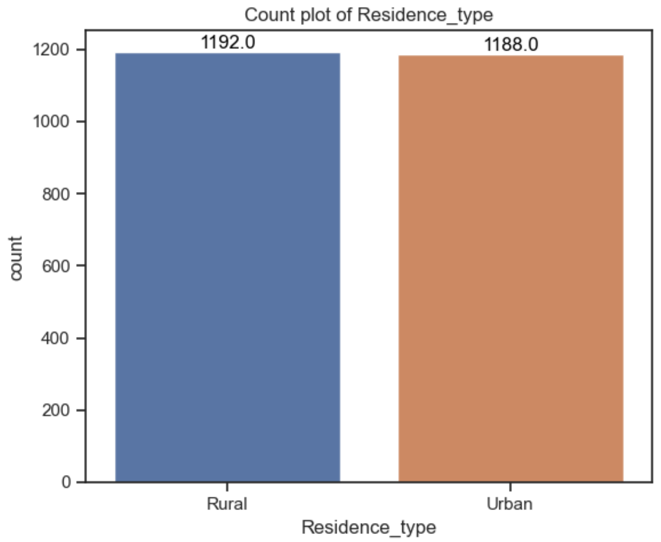
- Majority patients does not have hypertension



Heart Disease

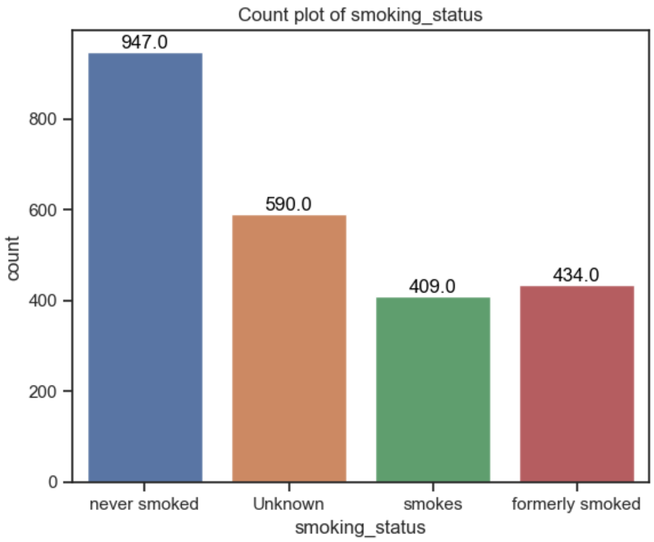
- There are 2262 patients does not have heart disease and 0nly 118 number of patients affected.



Ever married ● Major number of patients have married and onlly 699 of patients are not married.	
Work Type ● There are four types of work types in the dataset. Major class are binned in the Private work type which is 1505. ● Self employed patients have 416 and patients who are doing a government job is 329. However there are only 130 children's data available in this dataset.	
Residence Type ● There is a slight difference between rural and urbanly living area of the patients. However rural area data is higher than urbanly living patients.	

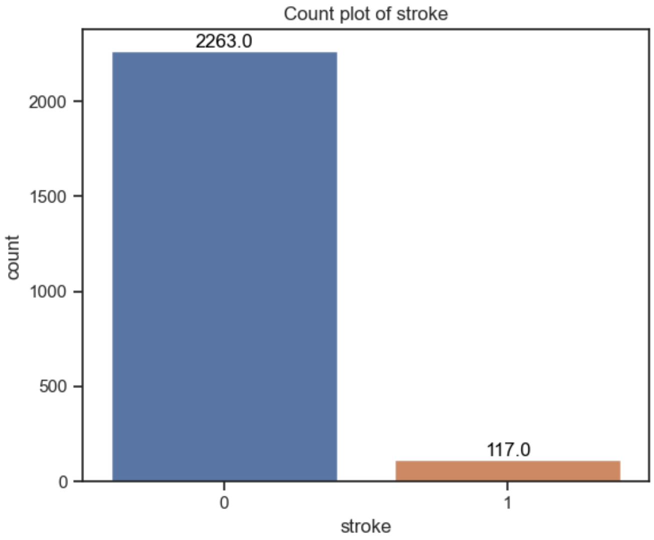
Smoking Status

- There are four categories of patients data available . Majority patients have not smoked and 590 of patient’s smoking data not available.
- However other rest of two categories have smoked and smoking.



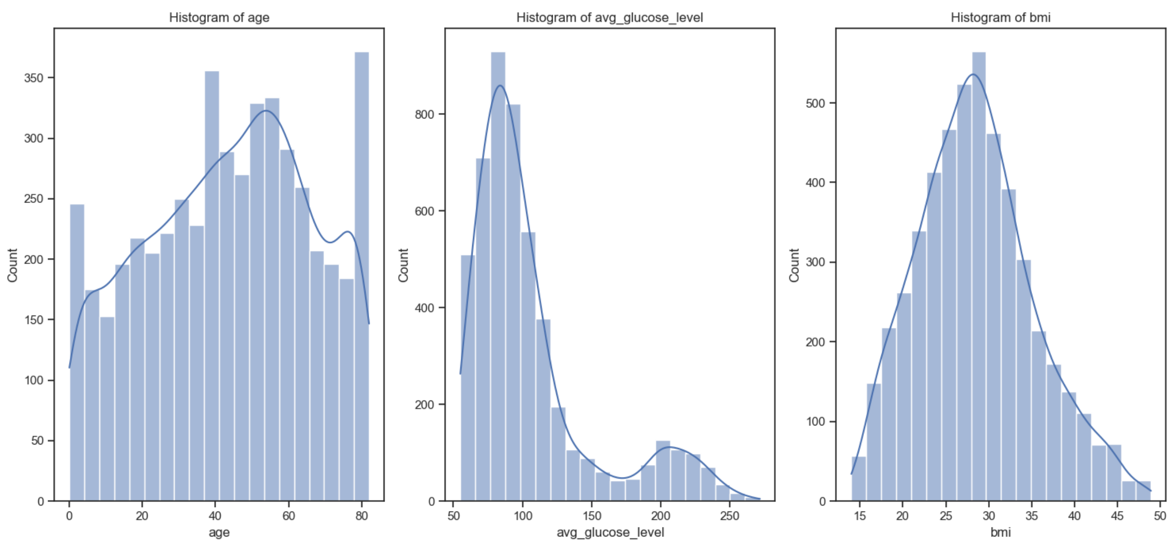
Stroke

- There are only 117 patients have accounted for had a stroke.



Continuous Parameters

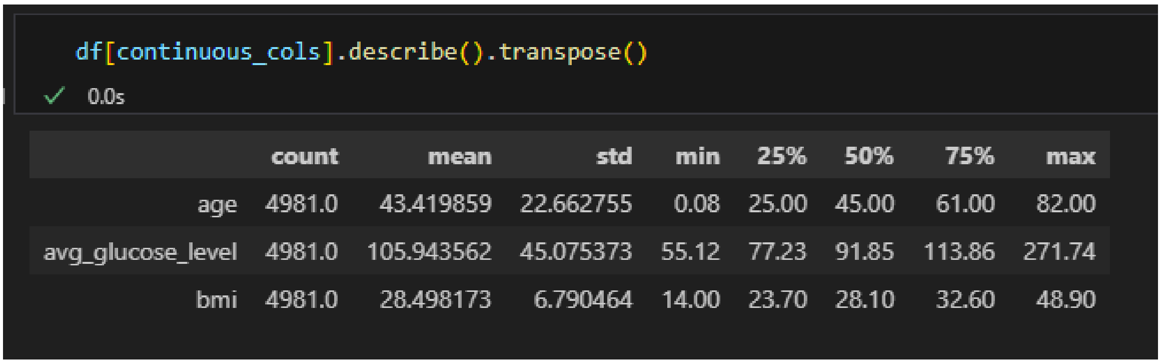
In this dataset there are three continuous variables available such as Age, Average glucose level and BMI. Using histogram plots I have analysed the data distribution and skewness of each variables.



Histogram of age

Histogram of avg_glucose_level

Histogram of bmi



```
df[continuous_cols].describe().transpose()
```

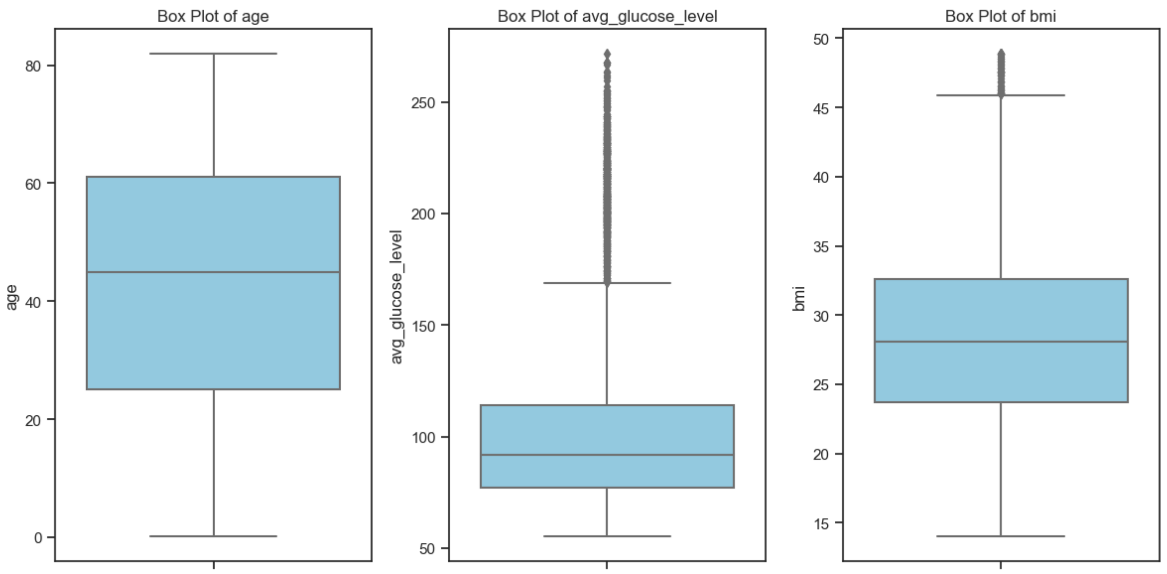
✓ 0.0s

```
                   count        mean        std    min    25%    50%     75%     max
              age  4981.0   43.419859  22.662755   0.08  25.00  45.00   61.00   82.00
avg_glucose_level  4981.0  105.943562  45.075373  55.12  77.23  91.85  113.86  271.74
              bmi  4981.0   28.498173   6.790464  14.00  23.70  28.10   32.60   48.90
```

- We can analyse the average glucose level has mostly left skewed in the plot which interprets there are outliers to be filtered.
- Age parameter varies from 0.08 to 82 years.

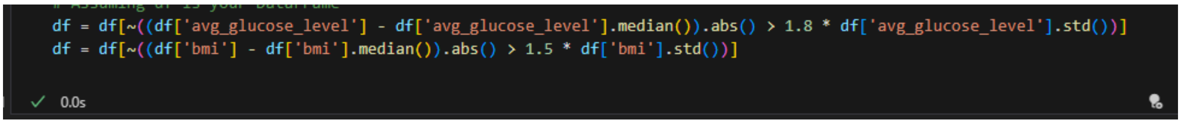
Handling Outliers

I have box plotted the three continuous variables in order to identify the outliers.



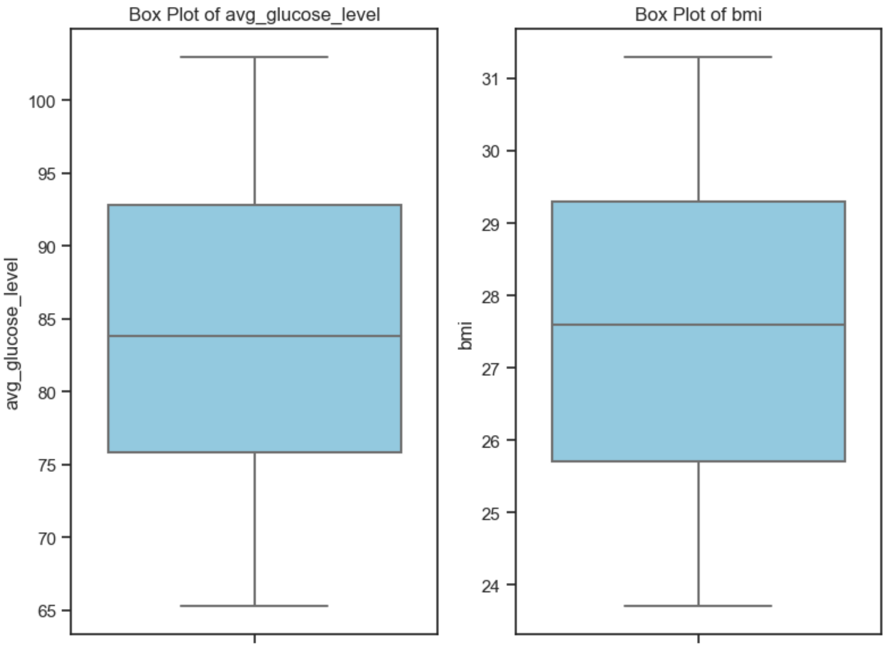
- We can clearly identify the average glucose level and BMI index have outlier data which can be filtered.

Using IQR Method outliers will be removed as follows.



```
# Assuming df is your DataFrame
df = df[~((df['avg_glucose_level'] - df['avg_glucose_level'].median()).abs() > 1.8 * df['avg_glucose_level'].std())]
df = df[~((df['bmi'] - df['bmi'].median()).abs() > 1.5 * df['bmi'].std())]
```

Results will be as follows…

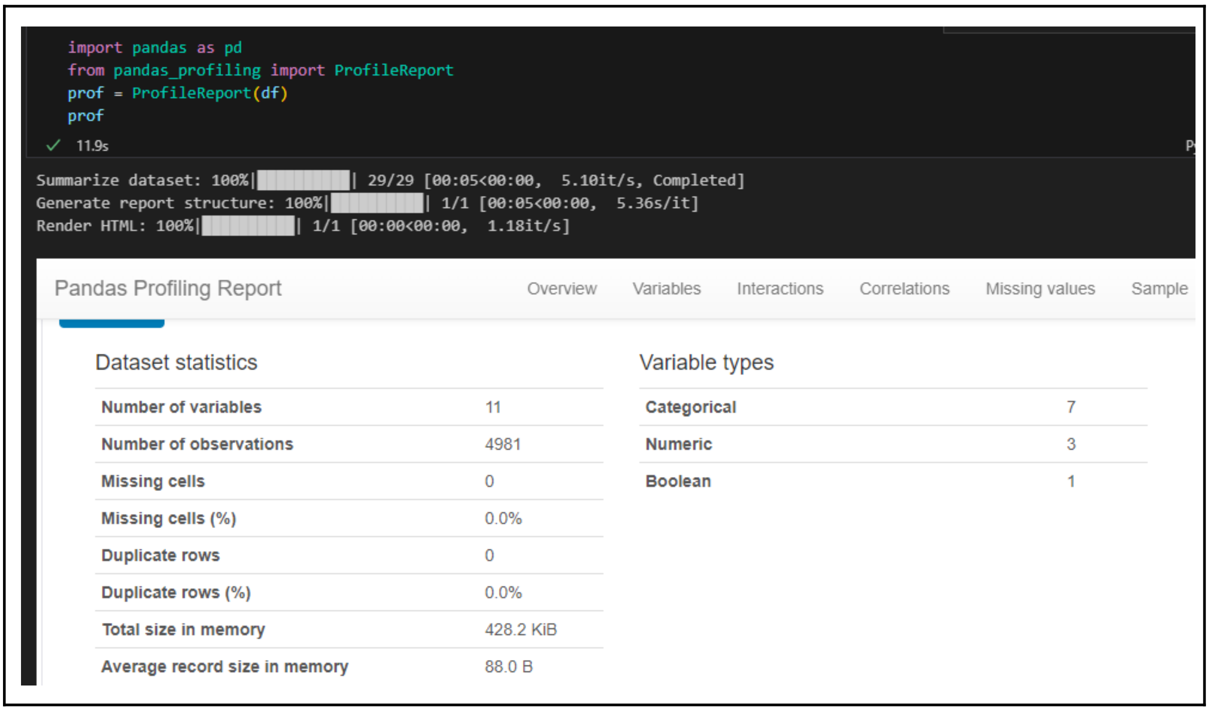


- Dataset dimension reduced to 3736 rows × 11 columns.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the process of investigating data to understand its characteristics and patterns. It can be performed using a variety of tools and techniques, such as visualization, statistical analysis. Also, it helps to identify potential problems with the data and to develop hypotheses about the data.

As a first step, I have created a Pandas Profiling Report.



```
import pandas as pd
from pandas_profiling import ProfileReport
prof = ProfileReport(df)
prof
```

✓ 11.9s

```
Summarize dataset: 100%|██████████| 29/29 [00:05<00:00,  5.10it/s, Completed]
Generate report structure: 100%|██████████| 1/1 [00:05<00:00,  5.36s/it]
Render HTML: 100%|██████████| 1/1 [00:00<00:00,  1.18it/s]
```

Dataset statistics

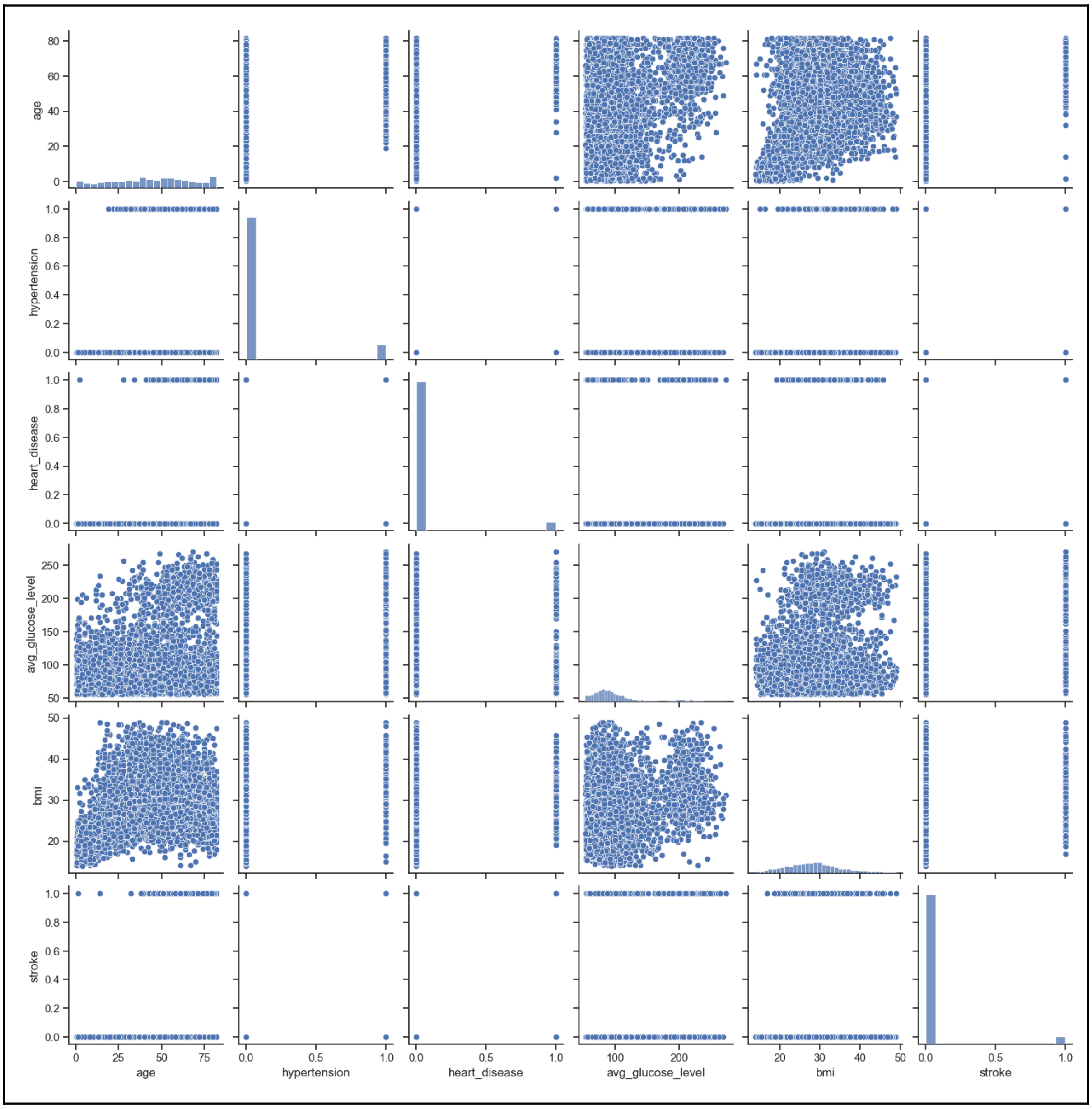
Number of variables	11
Number of observations	4981
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	428.2 KiB
Average record size in memory	88.0 B

Variable types

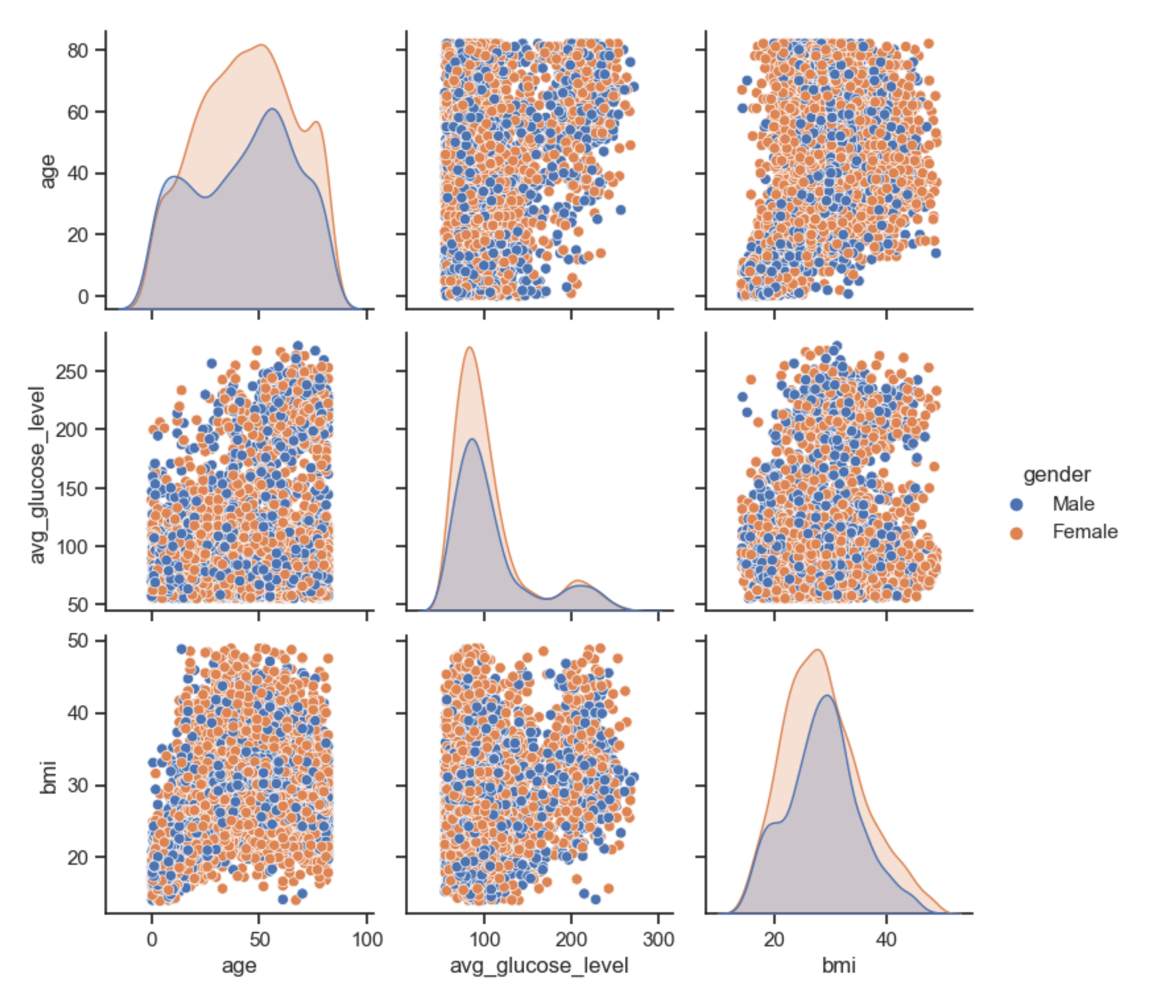
Categorical	7
Numeric	3
Boolean	1

Pandas profiling report provides all the missing values, value counts, corelation and overview of the dataset. These analysis already performed beginnig of the report.

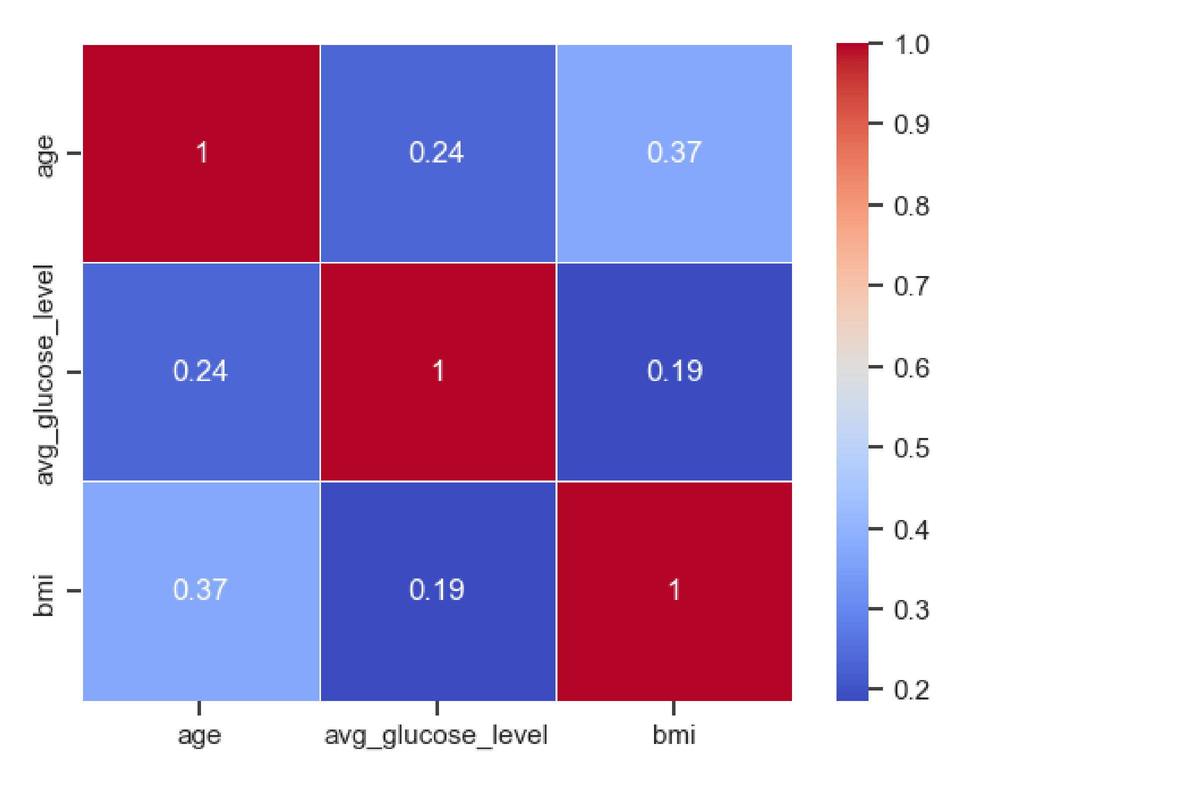
Pair Plot used to identify the relation with other variables within the dataset.

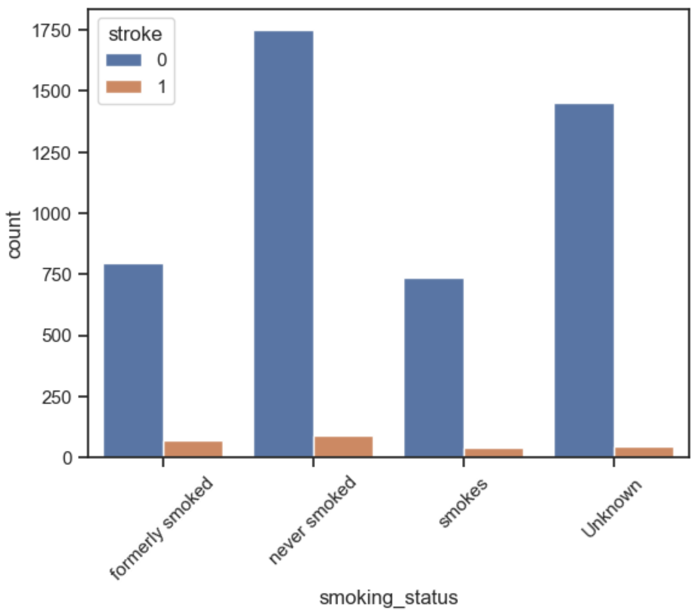
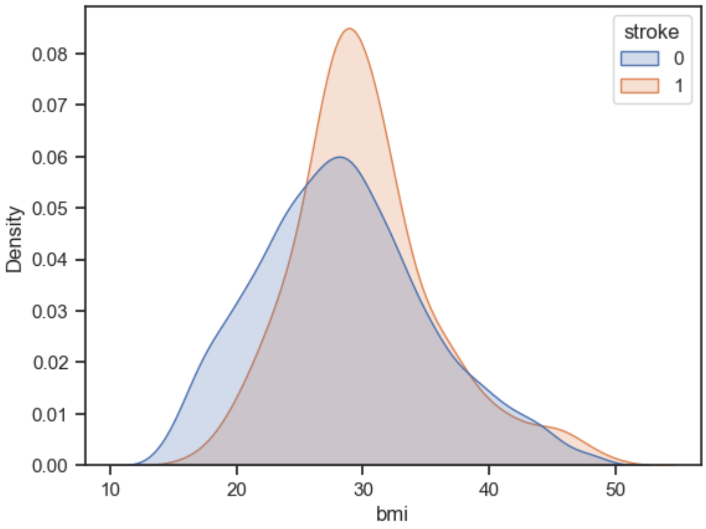
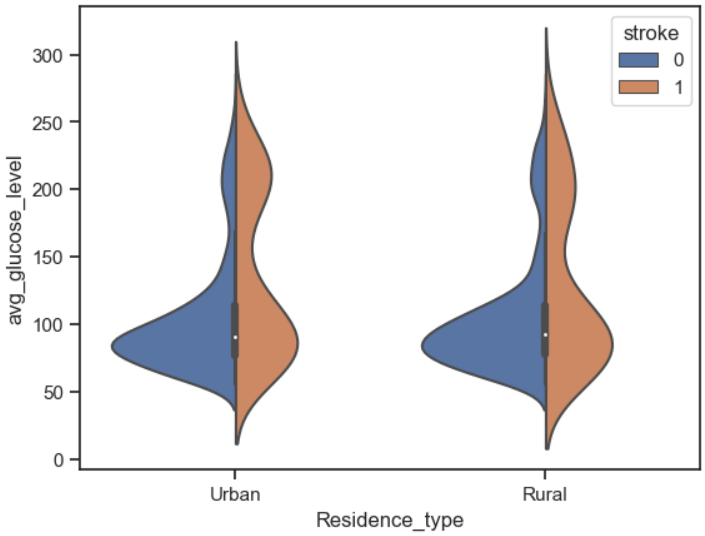


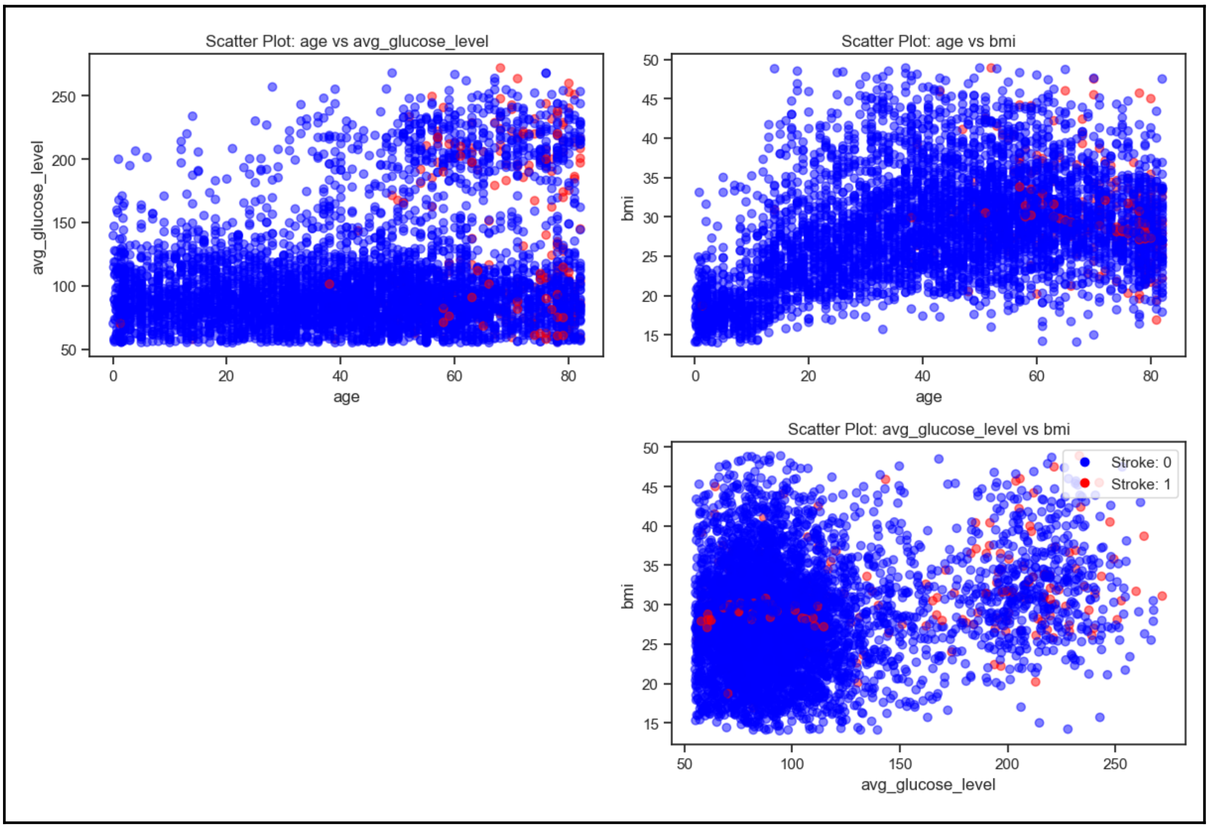
Analysing the corelation and pattern between numerical variables with pair plot and Heatmap



Heat Map



Relationship with Smoking and Stroke ● This chart interprets within the patients who have never smoked had a stroke compared to other three classes.	
Relationship with BMI and Stroke	
Relationship with BMI and Stroke ● Both variables behavevoir is similar when compared to each other	



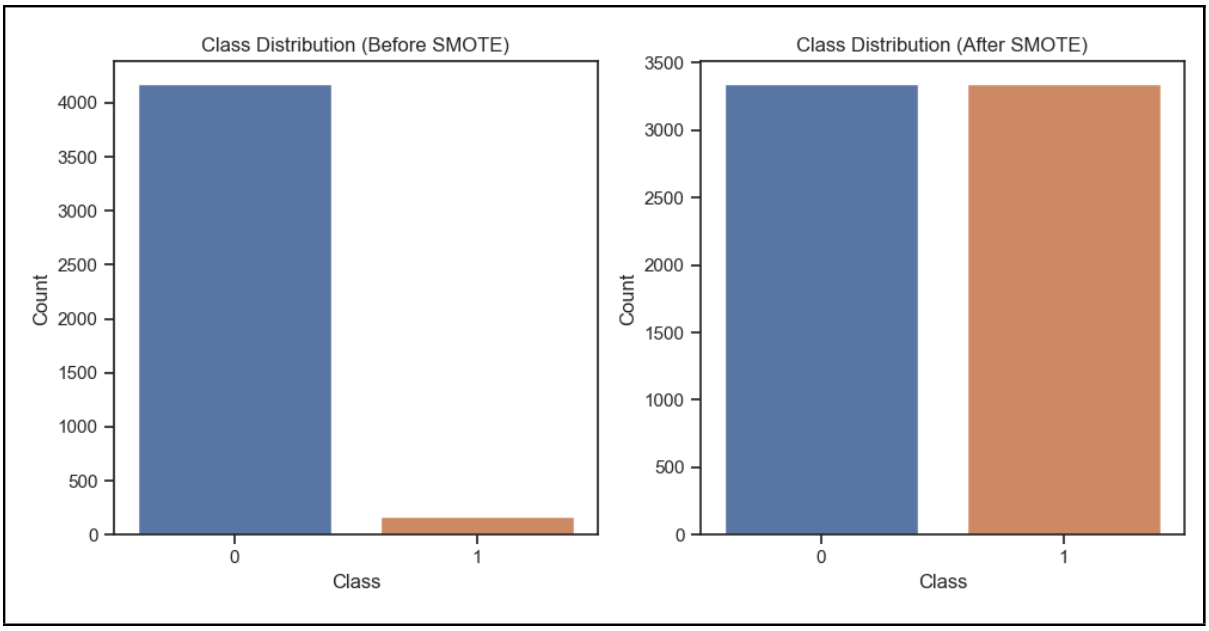
Summery of the EDA

- We have analysed missing values and this dataset does not contain any missing values.
- We analysed each variable counts and its corelation between other variables within the dataset.
- Based on the classification machine learning algorithm we apply to predict, we identified which variables need to be preprocessed and encoded.
- We identified that target variable has a class imbalance and it needed to be adjusted.
- By the insights generated from above analysis, myself have thorough knowledge about this dataset in ordert ot select the target variable and handle further predictions.

Feature Scaling and Encoding

Handling Data Imbalance

For our requirement and goal we need to balance the target variable (Stroke). In order to achieve that I have used the SMOTE technique. SMOTE (Synthetic Minority Over-sampling Technique) is a method used to address data imbalances in classification problems, particularly when we have an unequal distribution of classes in our dataset. It's often applied in scenarios where one class is underrepresented compared to another. The goal of SMOTE is to generate synthetic examples for the minority class to balance the dataset.



Feature Encoding

Feature encoding doesinvolves converting categorical or non-numeric features into a numerical format that can be used by machine learning algorithms.This is crucial preprocessing method because many machine learning models, especially those based on mathematical equations, require numerical input.

For achieve our goal and based on the EDA findings I have used two encoding techniques such as One Hot Encoding and Label Encodingcfrom sklearn.preprocessing

Encoding Method	Variables Applied
One Hot Encoding	'gender', 'ever_married', 'Residence_type'
Label Encoding	'Smoking_status', 'work_type'

Feature Scaling

After encoding the above mentioned variables ‘age’, 'avg_glucose_level', 'bmi' remained in the dataset. In these variables have varied range of numbers which are larger to the remaining features of the dataset.

```
Column: age, Min: 0.08, Max: 82.0
Column: avg_glucose_level, Min: 55.12, Max: 271.74
Column: bmi, Min: 14.0, Max: 48.9
```

I have applied standard scaler to scale the those variables for optimum outcome.
I have selected random forest classifier, support vector classifier, gradient boosting classifier for the machine learning outcome testing.

Each algorithm has its own capabilities and handling data preprocessing techniques its own. But most of the algorithms need to have a scaleda and encoded dataset inorder to perform best results. Now dataset is ready to apply for machine learning algorithms.

Model Selection

Project Overview

Objective: The primary goal of this project is to develop a robust and accurate classification model to predict whether patients are at risk of experiencing a *brain stroke* or not.

Machine Learning Algorithms:

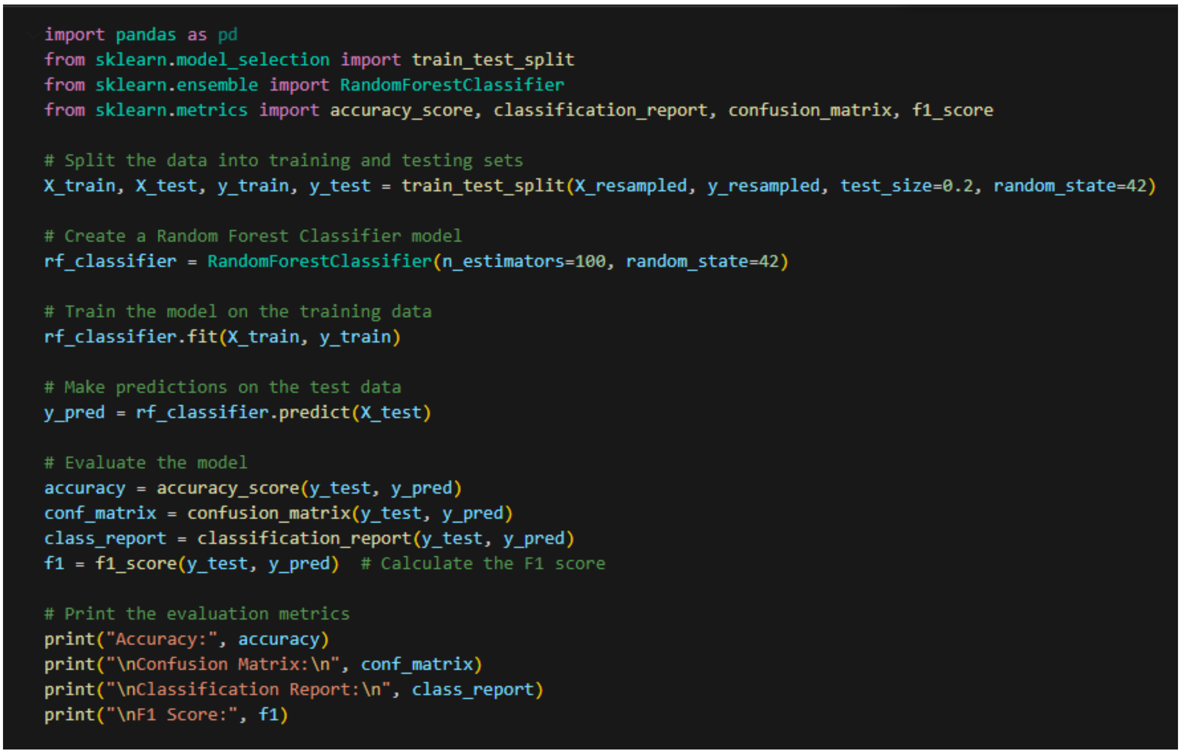
Random Forest Classifier: A versatile ensemble learning algorithm known for its ability to handle complex datasets and provide reliable predictions. It's selected for its strong generalization and feature importance analysis.

Support Vector Classifier (SVC): An efficient classifier that can effectively handle non-linear relationships in the data. SVC is chosen for its potential to separate stroke and non-stroke cases effectively.

Gradient Boosting Classifier: A powerful ensemble method that builds sequential decision trees to improve predictive performance. It's selected for its ability to capture complex patterns in the data and mitigate overfitting.

Our target variable will be ‘**Stroke’** and based on the above mentioned machine learning algorithms, python code implemented accordingly.

Random Forest Classifier



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Create a Random Forest Classifier model
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
f1 = f1_score(y_test, y_pred)  # Calculate the F1 score

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
print("\nF1 Score:", f1)
```

Classification Report				
	precision	**recall**	**f1-score**	**support**
0	0.96	0.92	0.95	832
1	0.93	0.97	0.95	838
accuracy			0.95	1670
macro avg	0.95	0.95	0.95	1670
weighted avg	0.95	0.95	0.95	1670
Accuracy:	0.9425149700598803			

- We got 94% accuracy from this model and True Positives (TP): 764 from the confusion matrix.

Hyper-parameter tuning (Grid Search)

For the Random Forest Classifier, the model evaluation and hyperparameter optimization process yielded the following results:

Utilizing a 5-fold cross-validation approach, a comprehensive search across 108 candidate parameter sets was conducted, culminating in a total of 540 individual model fits.

The optimal combination of hyperparameters identified for the Random Forest Classifier is as follows:

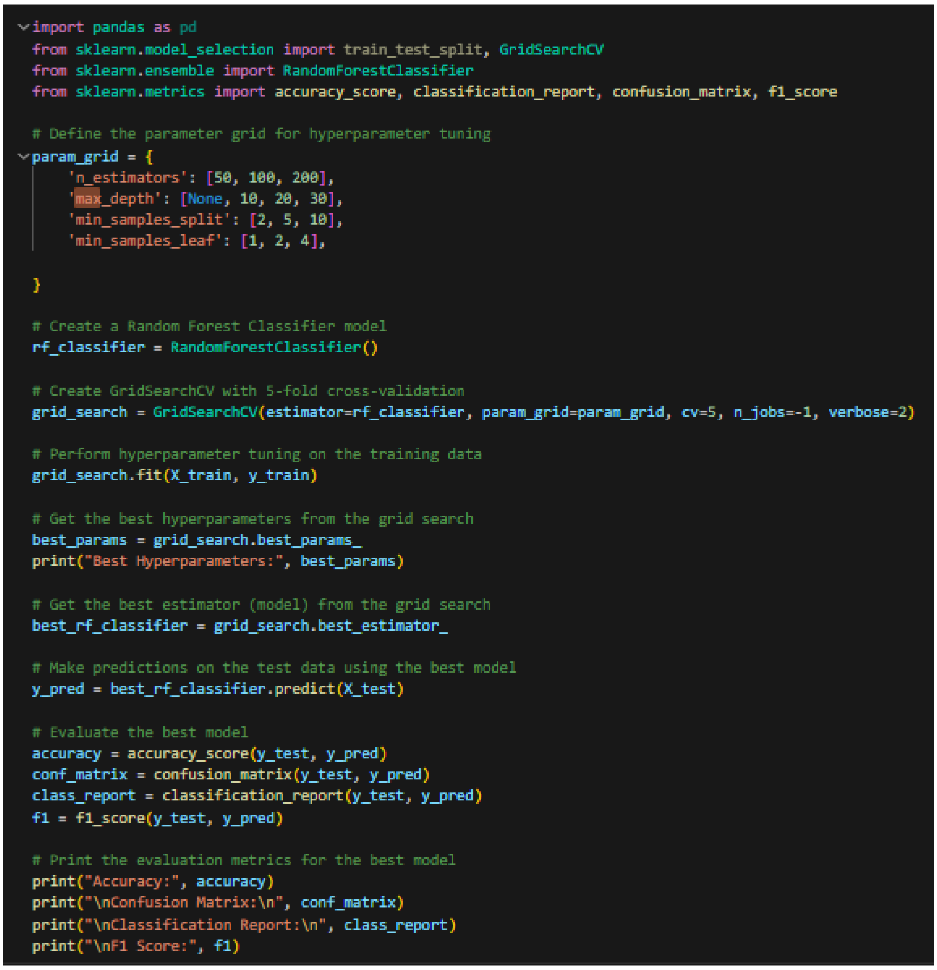
- 'max_depth': None
- 'min_samples_leaf': 1
- 'min_samples_split': 2
- 'n_estimators': 200

The model's performance in terms of accuracy reached an impressive value of approximately 95.09%. Additionally, the F1 Score, a metric that balances precision and recall, exhibited a high value of approximately 95.23%.

In further evaluating the model's performance, a confusion matrix was employed, revealing the following outcomes:

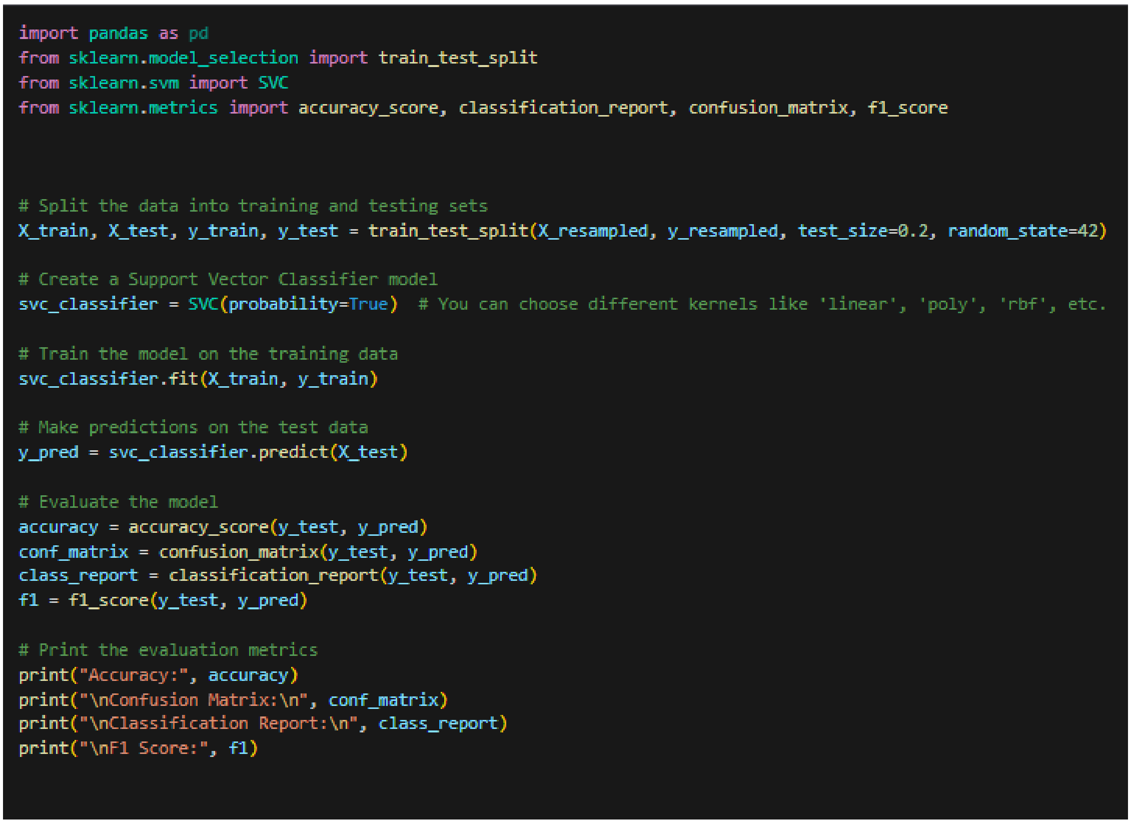
- True Positives (TP): 819
- True Negatives (TN): 769
- False Positives (FP): 63
- False Negatives (FN): 19

These results collectively attest to the Random Forest Classifier's strong predictive capabilities, accurately identifying relevant cases while minimizing both false positives and false negatives.



```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],

}

# Create a Random Forest Classifier model
rf_classifier = RandomForestClassifier()

# Create GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Perform hyperparameter tuning on the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters from the grid search
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best estimator (model) from the grid search
best_rf_classifier = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_rf_classifier.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics for the best model
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
print("\nF1 Score:", f1)
```

Support Vector Classifier (SVC)



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score



# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Create a Support Vector Classifier model
svc_classifier = SVC(probability=True)  # You can choose different kernels like 'linear', 'poly', 'rbf', etc.

# Train the model on the training data
svc_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svc_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
print("\nF1 Score:", f1)
```

Classification Report				
	precision	**recall**	**f1-score**	**support**
0	0.93	0.79	0.86	832
1	0.82	0.94	0.88	838
accuracy			0.87	1670
macro avg	0.87	0.87	0.87	1670
weighted avg	0.87	0.87	0.87	1670
Accuracy:	.86646706586802635			

- We got 86% accuracy from this model and True Positives (TP): 659 from confusion matrix.

Hyper parameter tuning (Grid Search)

In the context of model evaluation and hyperparameter optimization, the following outcomes were observed for the Support Vector Classifier:

Employing a 5-fold cross-validation strategy, an extensive exploration of 16 distinct candidate parameter configurations was conducted, resulting in a total of 80 individual model fits.

The most effective combination of hyperparameters, as identified through this process, is characterized by the following settings:

- 'C': 10
- 'gamma': 'scale'
- 'kernel': 'rbf'
- The model exhibited a commendable level of accuracy, achieving a score of approximately 89.58%.

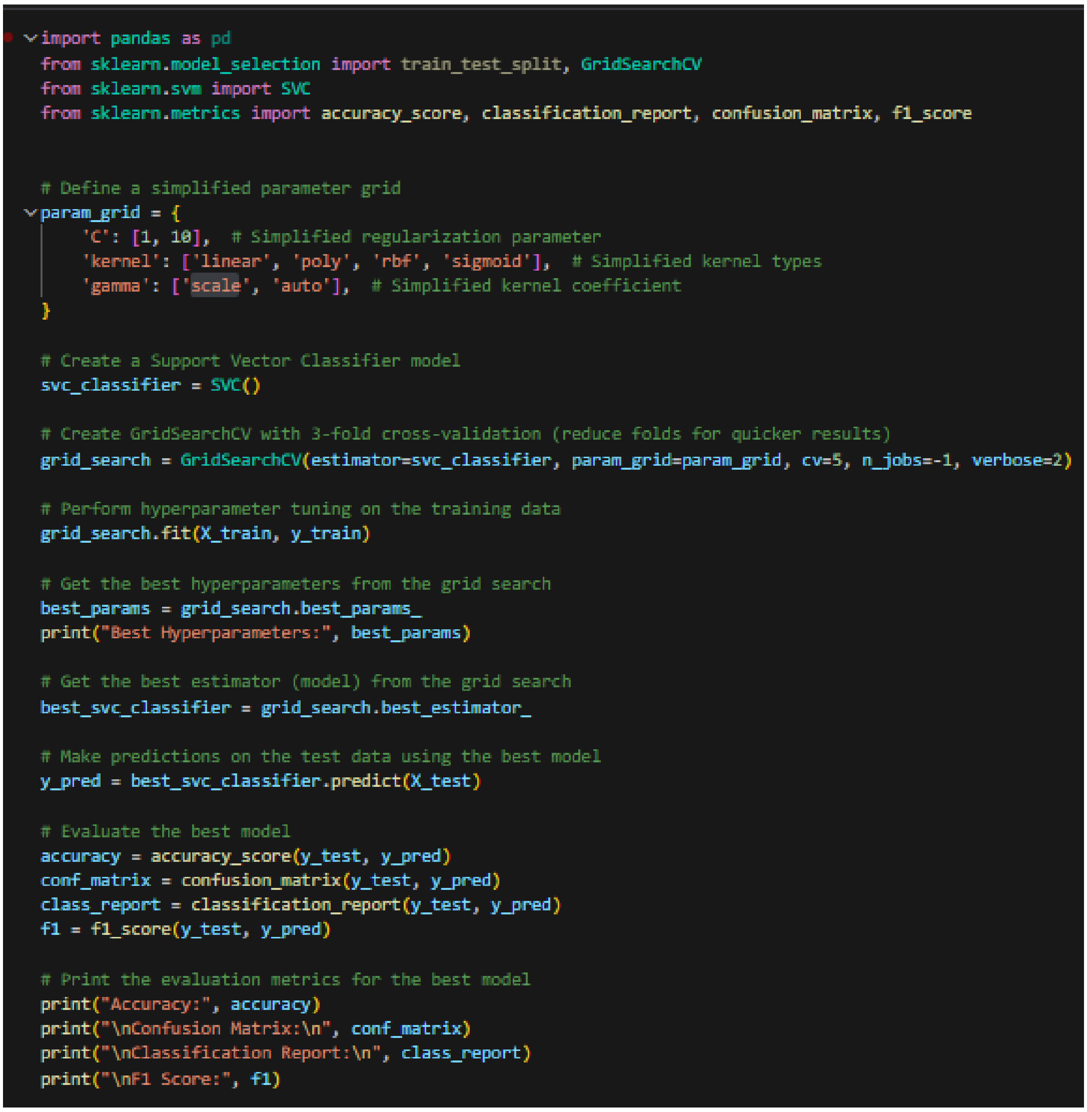
For a comprehensive assessment of the model's performance, a confusion matrix was employed, unveiling the following results:

- True Positives (TP): 805
- True Negatives (TN): 691
- False Positives (FP): 141
- False Negatives (FN): 33

Furthermore, a detailed classification report provides an in-depth breakdown of precision, recall, and F1-score for both classes (0 and 1), along with support values.

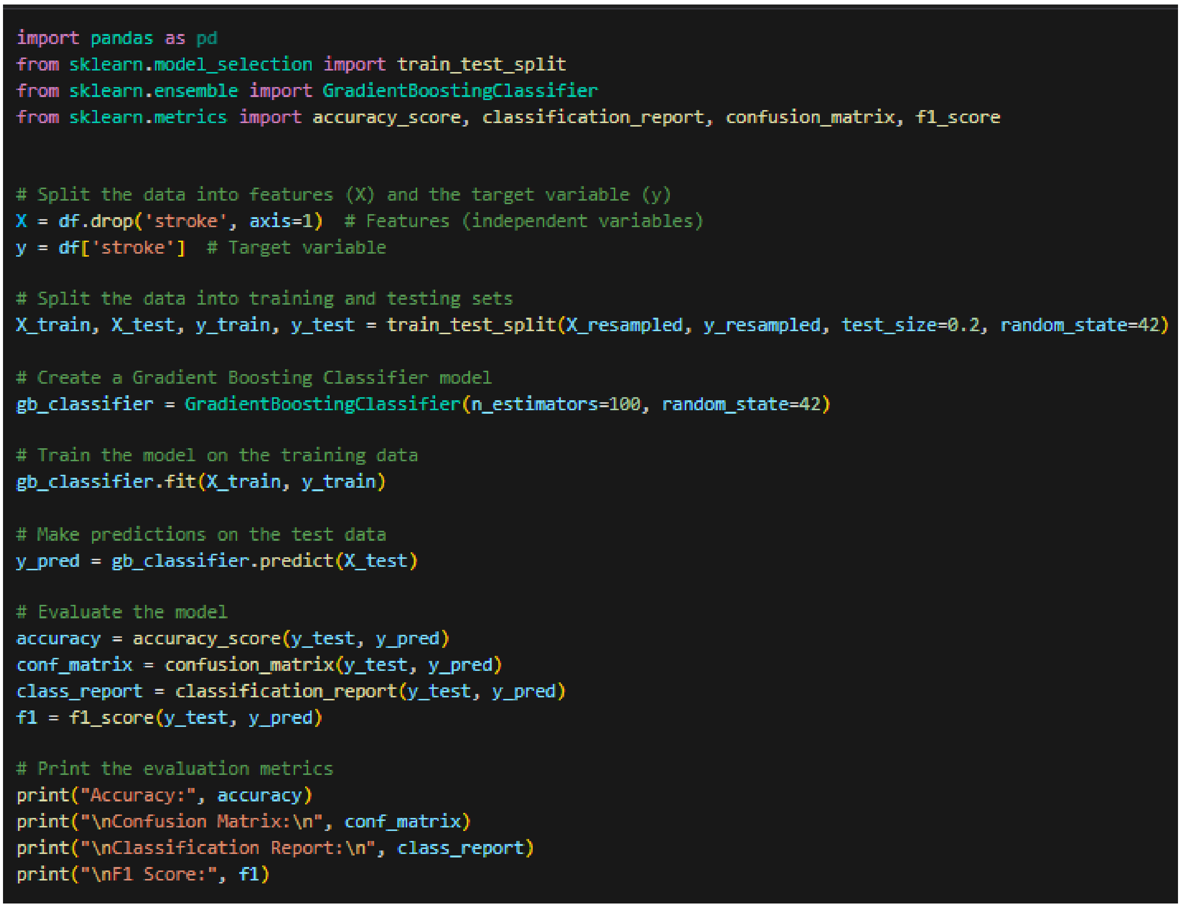
F1 Score: 0.9025

The weighted average F1 score, which balances precision and recall across classes, is approximately 0.9025. These metrics collectively attest to the Support Vector Classifier's strong predictive performance, effectively distinguishing between classes while maintaining a balance between precision and recall.



```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score


# Define a simplified parameter grid
param_grid = {
    'C': [1, 10],  # Simplified regularization parameter
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Simplified kernel types
    'gamma': ['scale', 'auto'],  # Simplified kernel coefficient
}

# Create a Support Vector Classifier model
svc_classifier = SVC()

# Create GridSearchCV with 3-fold cross-validation (reduce folds for quicker results)
grid_search = GridSearchCV(estimator=svc_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Perform hyperparameter tuning on the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters from the grid search
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best estimator (model) from the grid search
best_svc_classifier = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_svc_classifier.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics for the best model
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
print("\nF1 Score:", f1)
```

Gradient Boosting Classifier



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score


# Split the data into features (X) and the target variable (y)
X = df.drop('stroke', axis=1)  # Features (independent variables)
y = df['stroke']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Create a Gradient Boosting Classifier model
gb_classifier = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
gb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = gb_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
print("\nF1 Score:", f1)
```

Classification Report				
	precision	**recall**	**f1-score**	**support**
0	0.91	0.84	0.88	832
1	0.85	0.92	0.89	838
accuracy			0.88	1670
macro avg	0.88	0.88	0.88	1670
weighted avg	0.88	0.88	0.88	1670
Accuracy:	0.8802395209580839			

- We got 88% accuracy from this model and True Positives (TP): 700 from confusion matrix.

Hyper-parameter tuning (Grid Search)

In the context of model evaluation and hyperparameter optimization, the Gradient Boosting Classifier yielded the following notable outcomes:

Leveraging a 5-fold cross-validation approach, an extensive exploration encompassing 243 distinct candidate parameter configurations was conducted, resulting in a total of 1215 individual model fits.

The hyperparameter tuning process revealed an optimal combination of settings that significantly enhance the model's performance. These settings are as follows:

- 'learning_rate': 0.2
- 'max_depth': 5
- 'min_samples_leaf': 2
- 'min_samples_split': 10
- 'n_estimators': 300
-

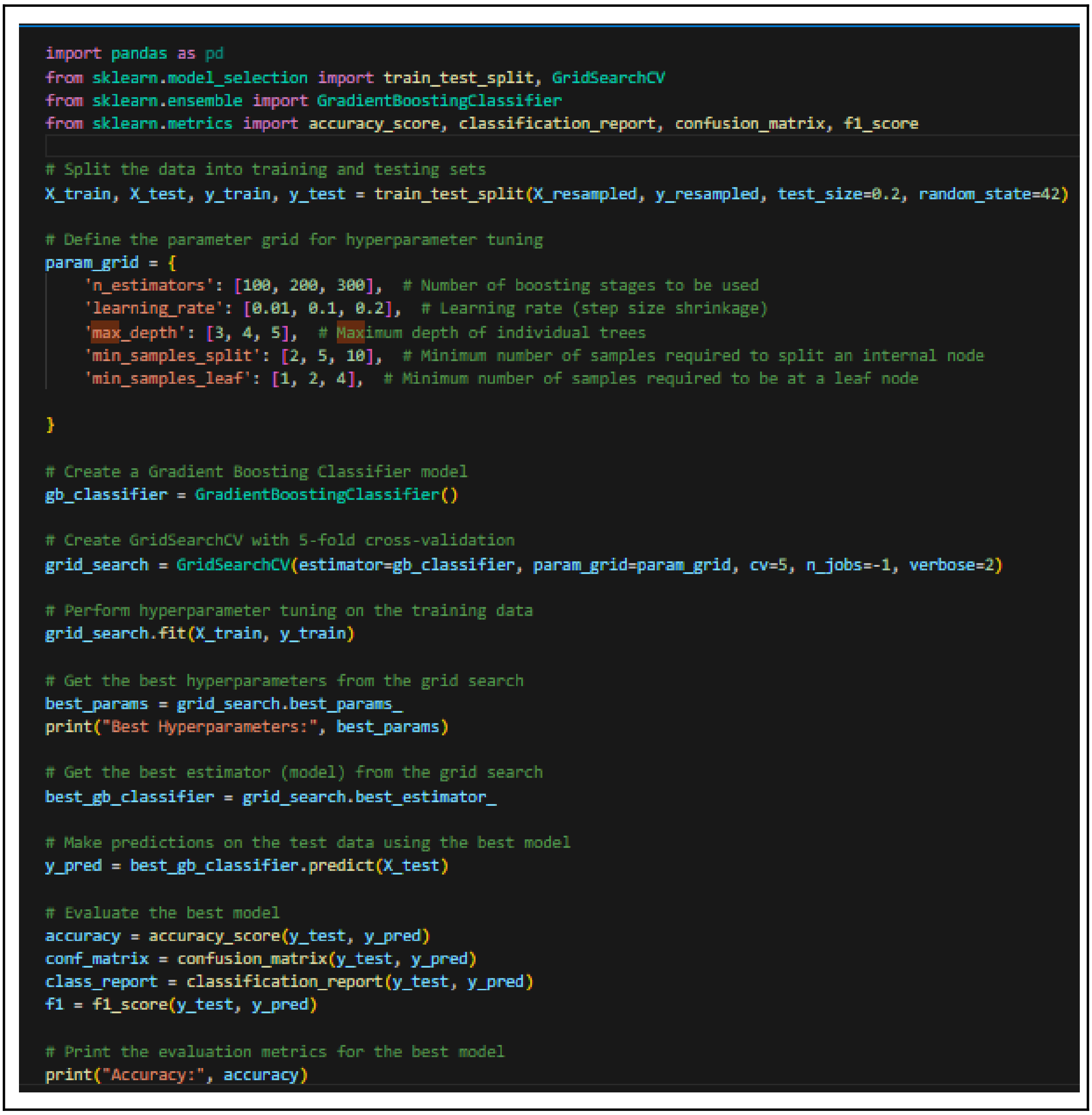
The model demonstrated a remarkable level of accuracy, achieving a score of approximately 96.77%.

For a comprehensive assessment of the model's performance, a confusion matrix was employed, presenting the following results:

- True Positives (TP): 804
- True Negatives (TN): 812
- False Positives (FP): 20
- False Negatives (FN): 34

Additionally, a detailed classification report offers an in-depth analysis of precision, recall, and F1-score for both classes (0 and 1), along with corresponding support values.

F1 Score: 0.9675
The weighted average F1 score, representing a harmonious balance between precision and recall across classes, is approximately 0.9675. These metrics collectively underscore the Gradient Boosting Classifier's exceptional predictive capabilities, effectively distinguishing between classes while upholding a high level of precision and recall.



```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of boosting stages to be used
    'learning_rate': [0.01, 0.1, 0.2],  # Learning rate (step size shrinkage)
    'max_depth': [3, 4, 5],  # Maximum depth of individual trees
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be at a leaf node

}

# Create a Gradient Boosting Classifier model
gb_classifier = GradientBoostingClassifier()

# Create GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(estimator=gb_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Perform hyperparameter tuning on the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters from the grid search
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best estimator (model) from the grid search
best_gb_classifier = grid_search.best_estimator_

# Make predictions on the test data using the best model
y_pred = best_gb_classifier.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics for the best model
print("Accuracy:", accuracy)
```

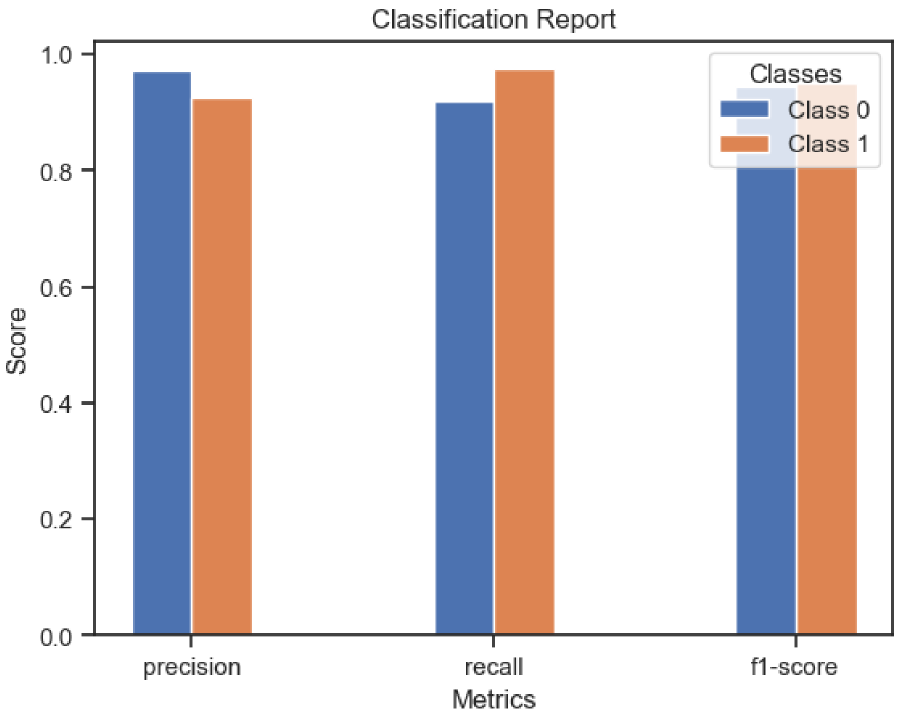
I have selected three machine learning algorithms and performed grid search for each algorithm in order to find the best hypermeter tuning. The best performing machine learning algorithm is Gradient Boosting Classifier algorithm achieving 96% accuracy level.

Model Evaluation

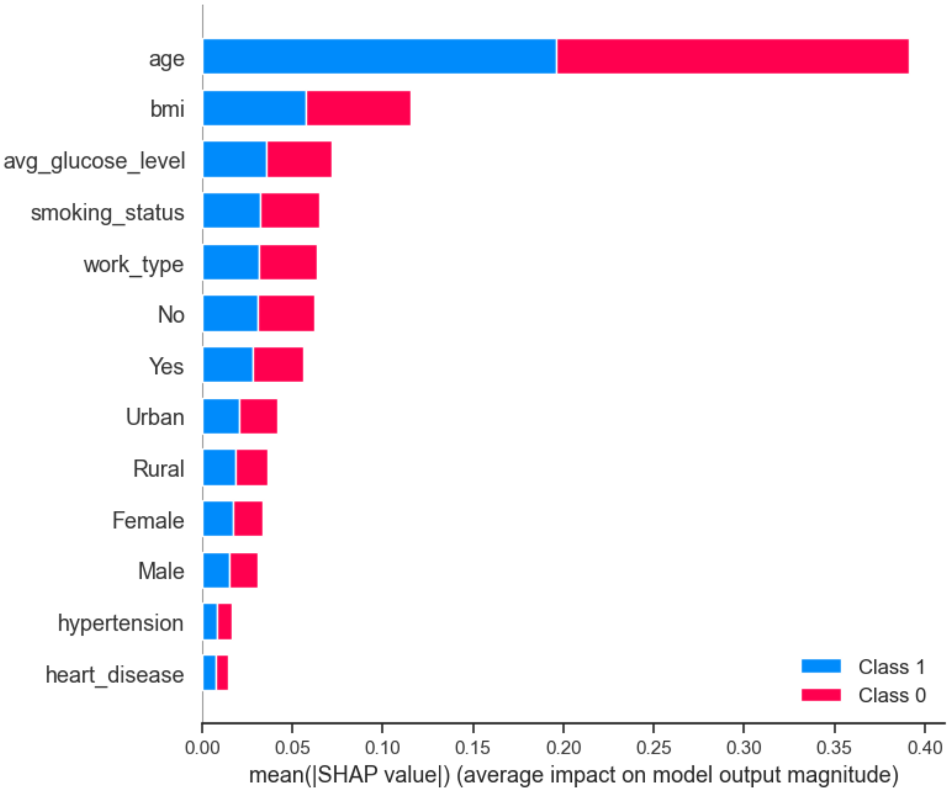
With the model evaluation performed by classification metrics such as F1 score, Confusion matrix and feature importance.

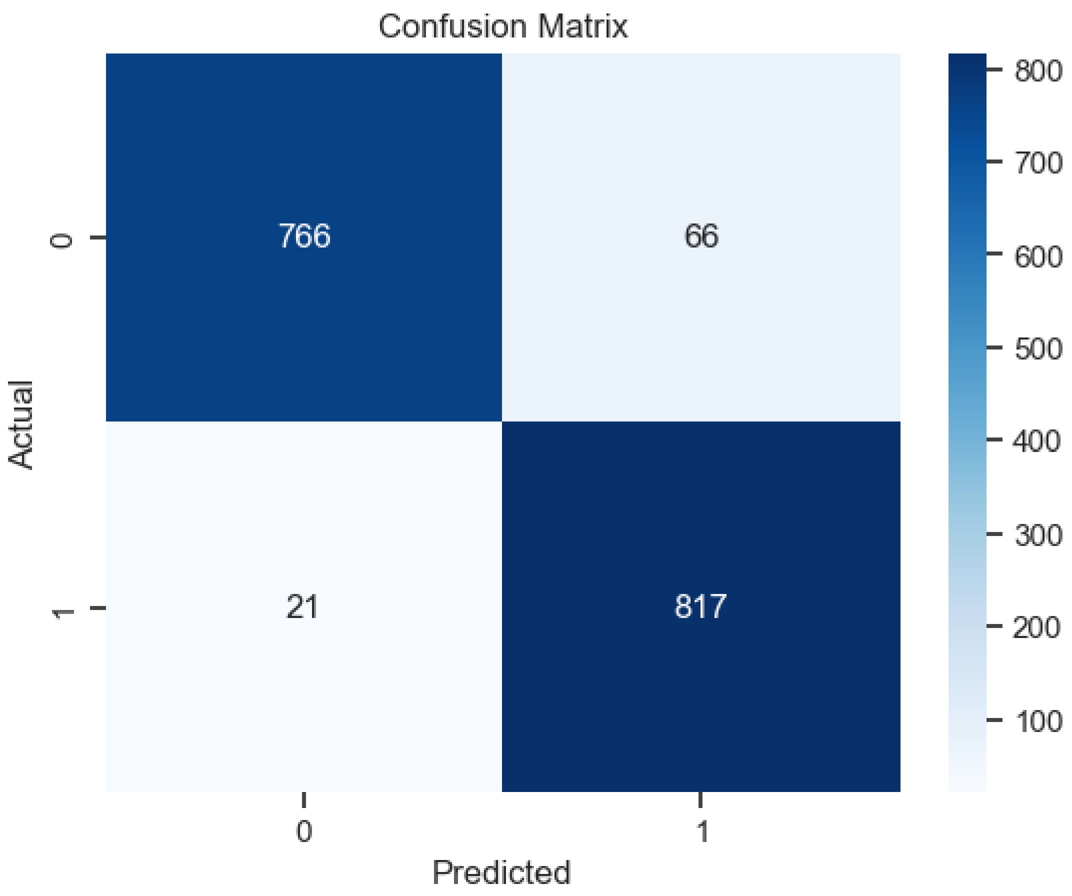
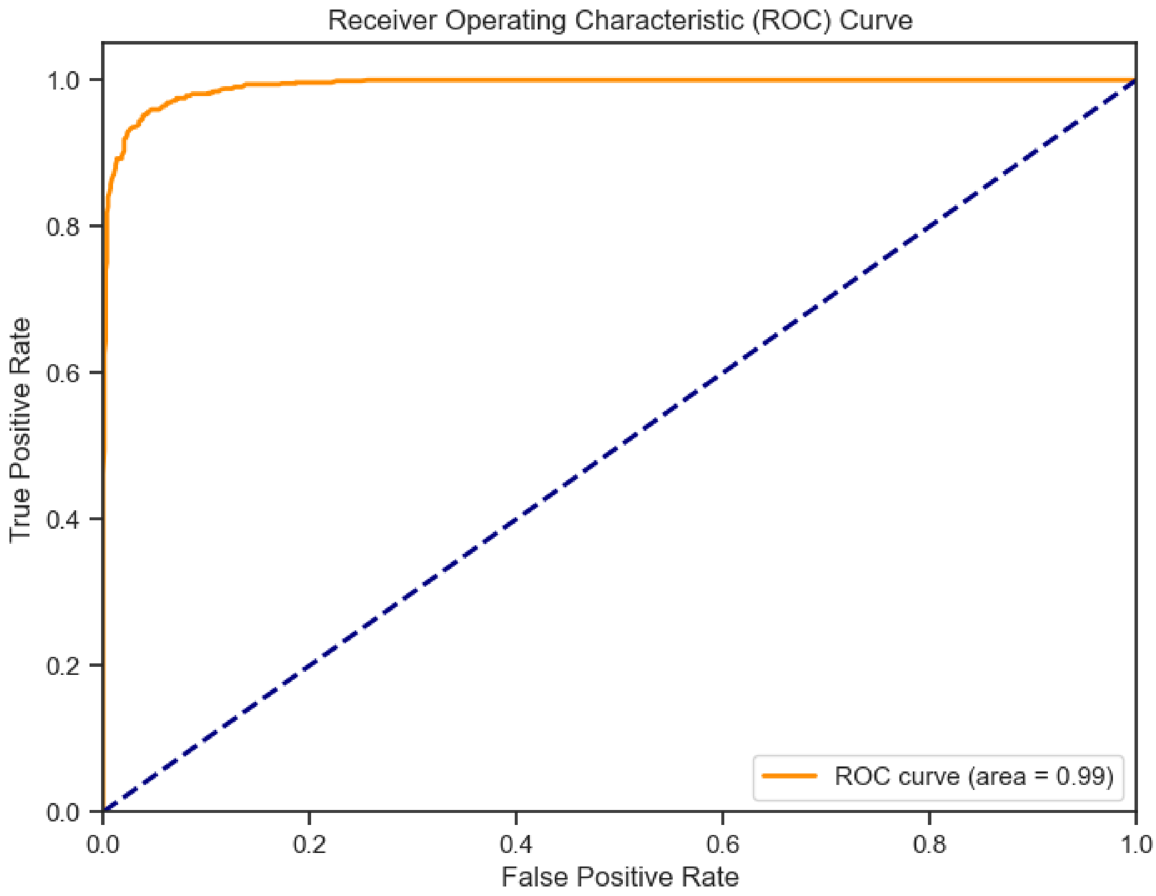
Random Forest Classifier

Classification Report



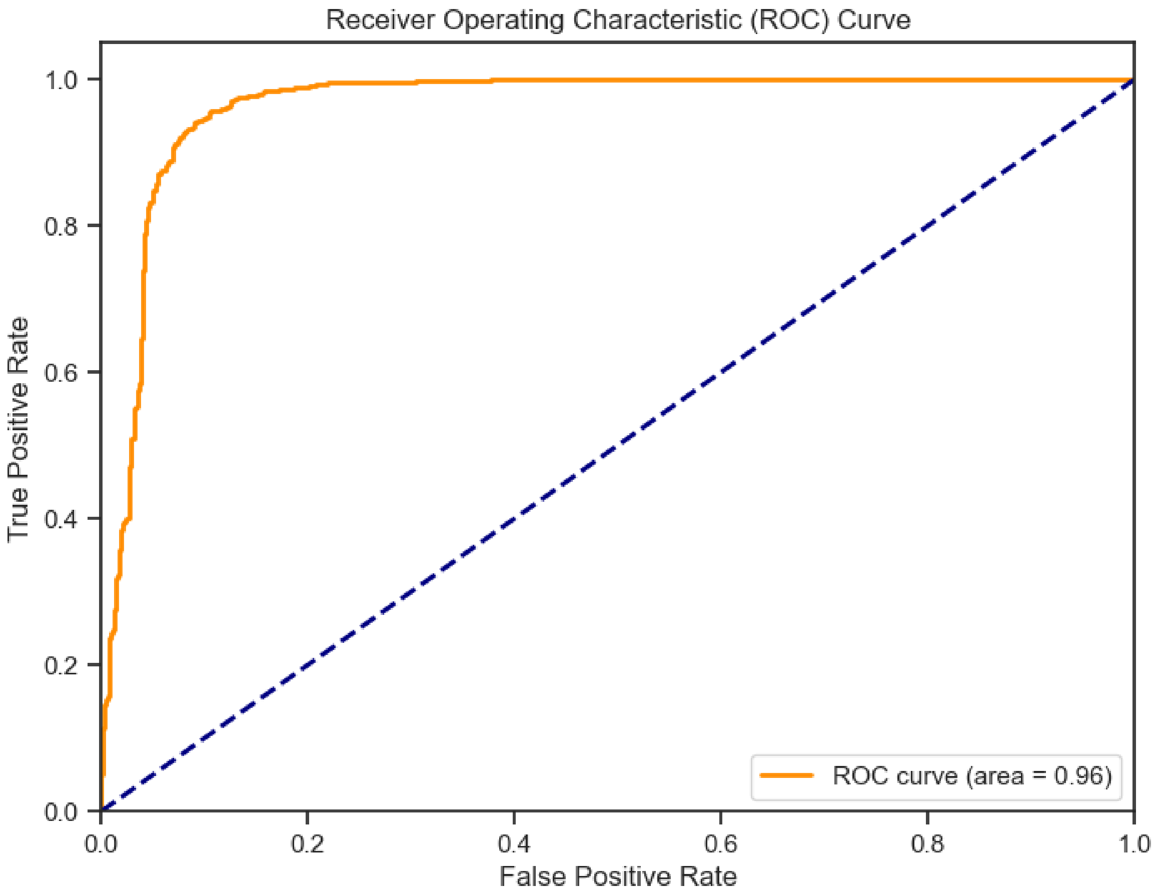
Feature importance



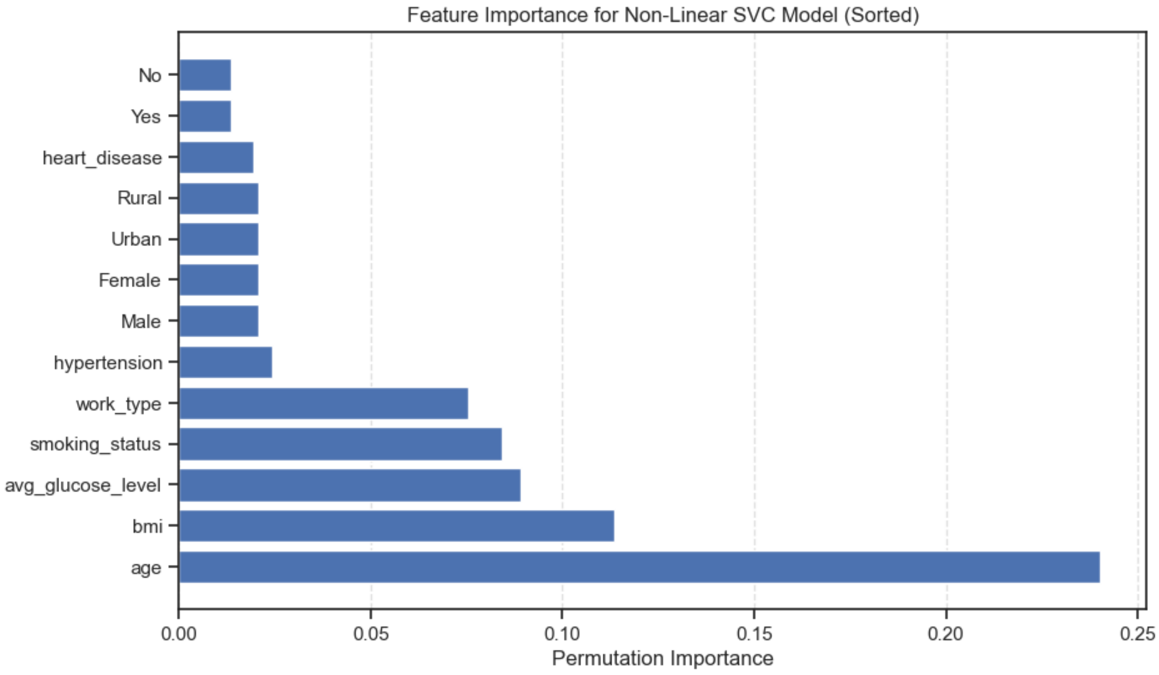
Confusion Matrix



Support Vector Classifier

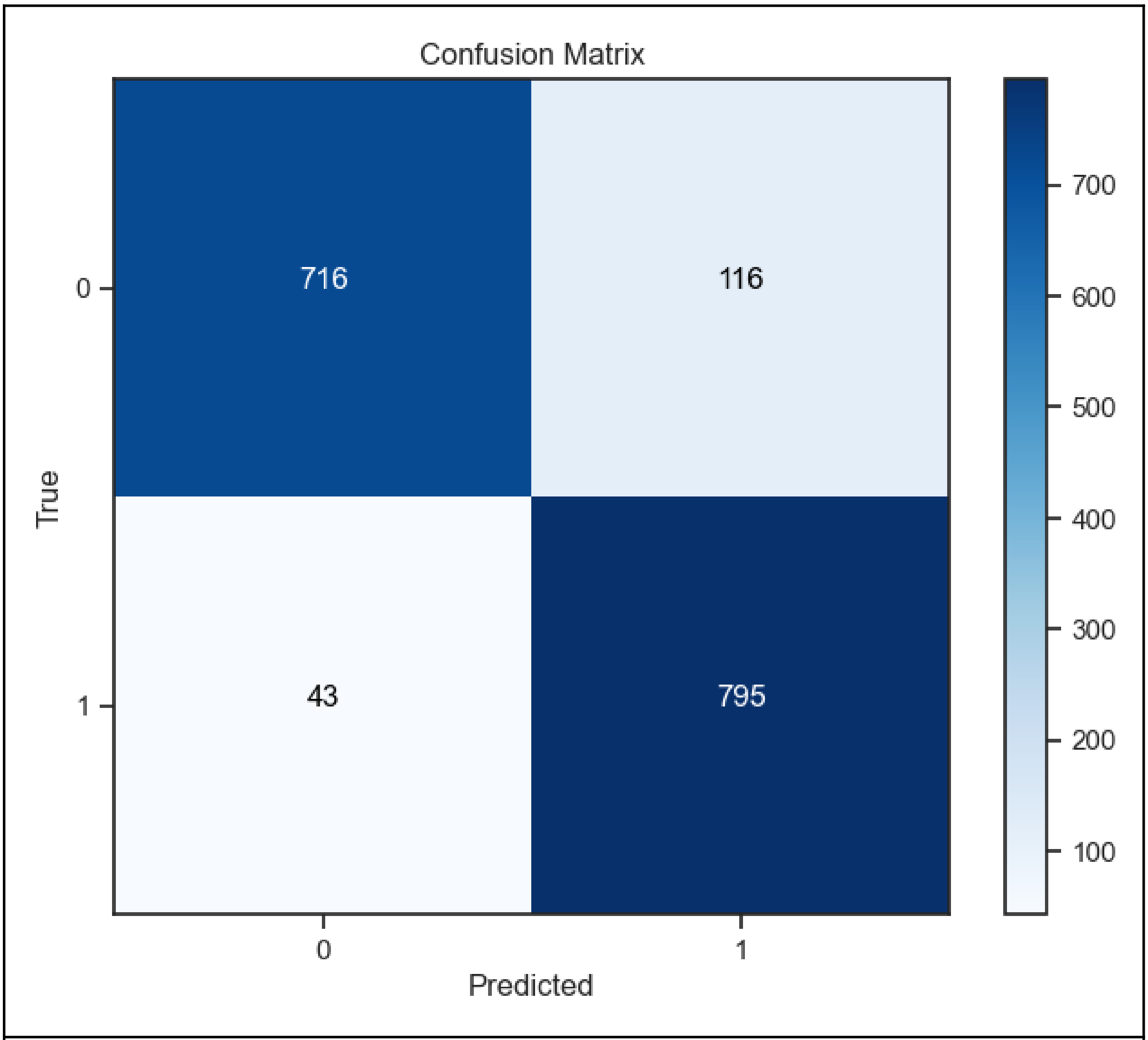
ROC Curve

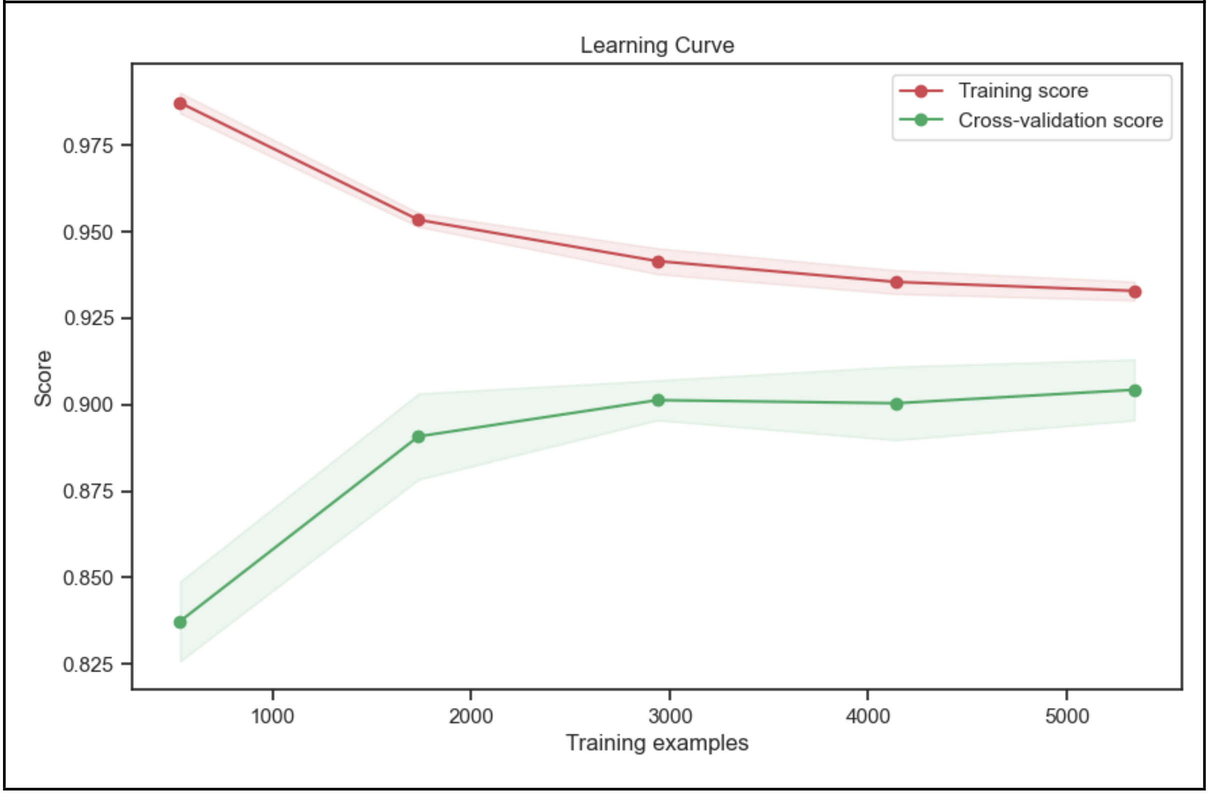


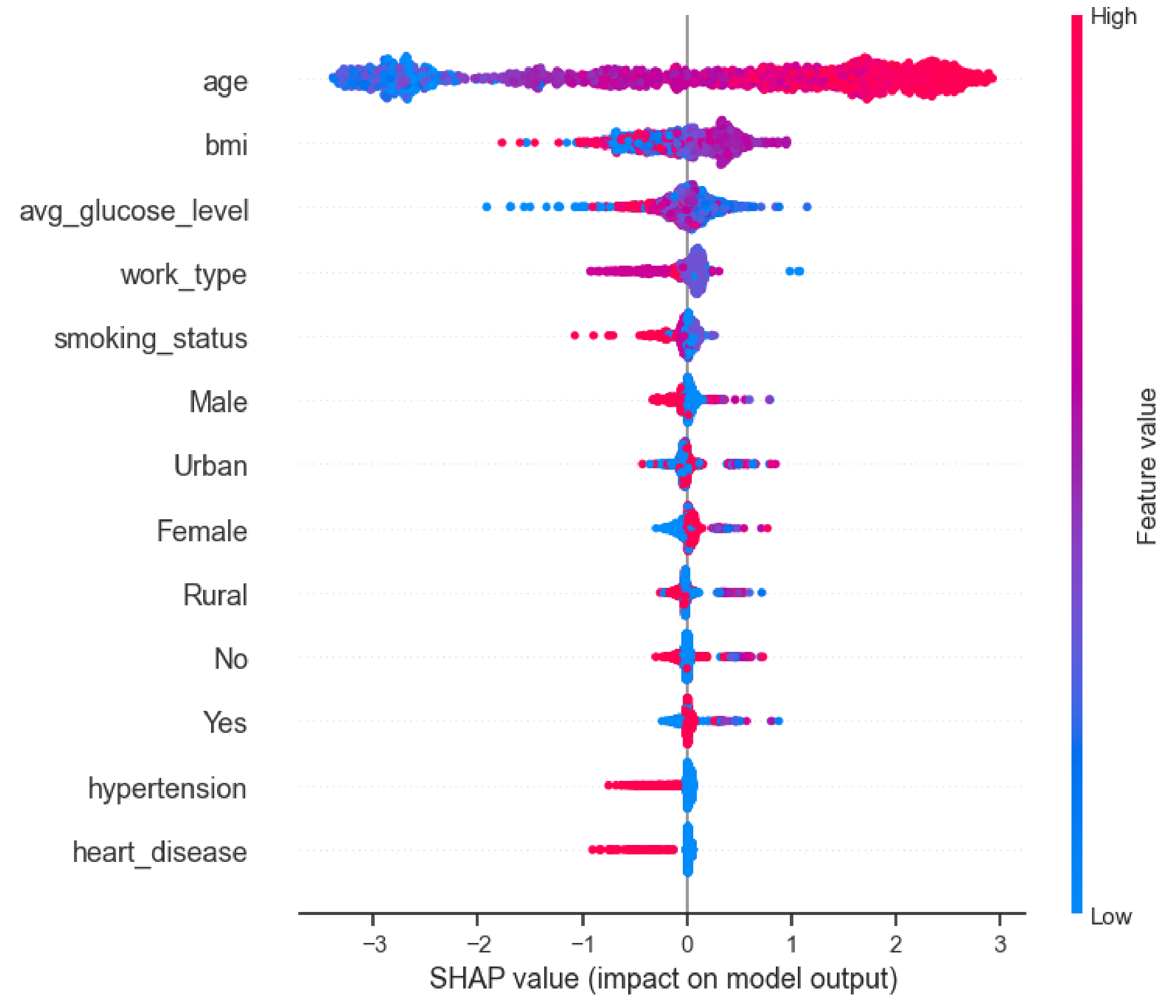
Feature Importance



Gradient Boosting Classifier



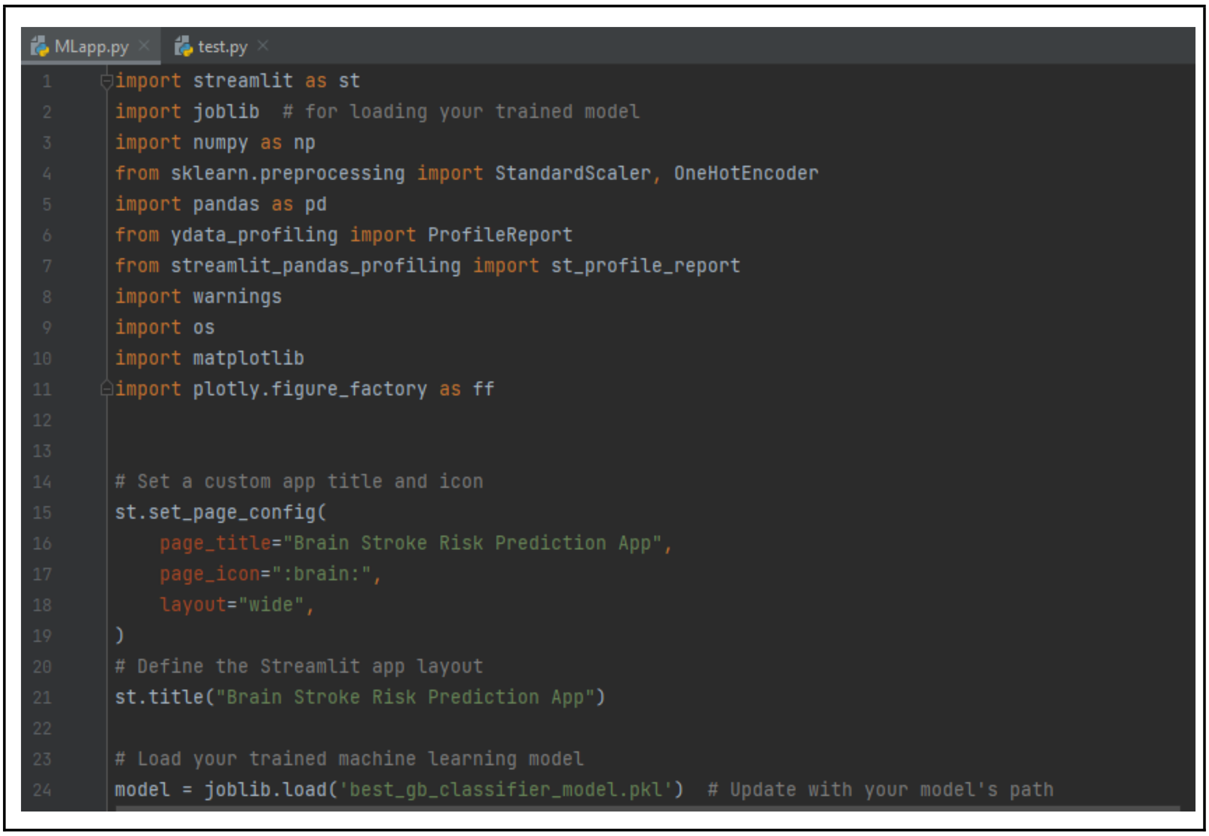


Feature Impotance


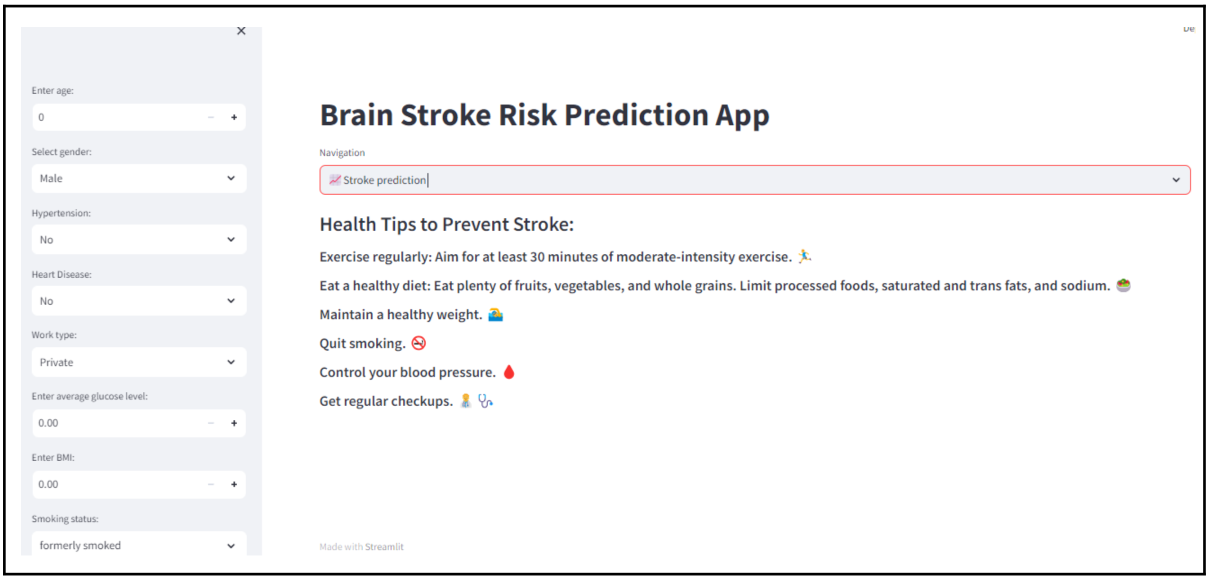
- **As a summery Age , BMI, Average Glucose level is main imporatnce and common feature for all three Machine learning algorithms.**

Model Interpretability and Charts Visualization

Fir this task i have taken a step to create a machine learning prediction app based on user inputs by using Streamlit Framework.
In order to complete that task I have installed the frame work to the python environment and coded a app with user friendly interface.



```
import streamlit as st
import joblib  # for loading your trained model
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd
from ydata_profiling import ProfileReport
from streamlit_pandas_profiling import st_profile_report
import warnings
import os
import matplotlib
import plotly.figure_factory as ff


# Set a custom app title and icon
st.set_page_config(
    page_title="Brain Stroke Risk Prediction App",
    page_icon=":brain:",
    layout="wide",
)
# Define the Streamlit app layout
st.title("Brain Stroke Risk Prediction App")

# Load your trained machine learning model
model = joblib.load('best_gb_classifier_model.pkl')  # Update with your model's path
```



Enter age: 0

Select gender: Male

Hypertension: No

Heart Disease: No

Work type: Private

Enter average glucose level: 0.00

Enter BMI: 0.00

Smoking status: formerly smoked

Brain Stroke Risk Prediction App

Navigation: 📈 Stroke prediction

Health Tips to Prevent Stroke:

Exercise regularly: Aim for at least 30 minutes of moderate-intensity exercise. 🏃

Eat a healthy diet: Eat plenty of fruits, vegetables, and whole grains. Limit processed foods, saturated and trans fats, and sodium. 🥗

Maintain a healthy weight. 🏊

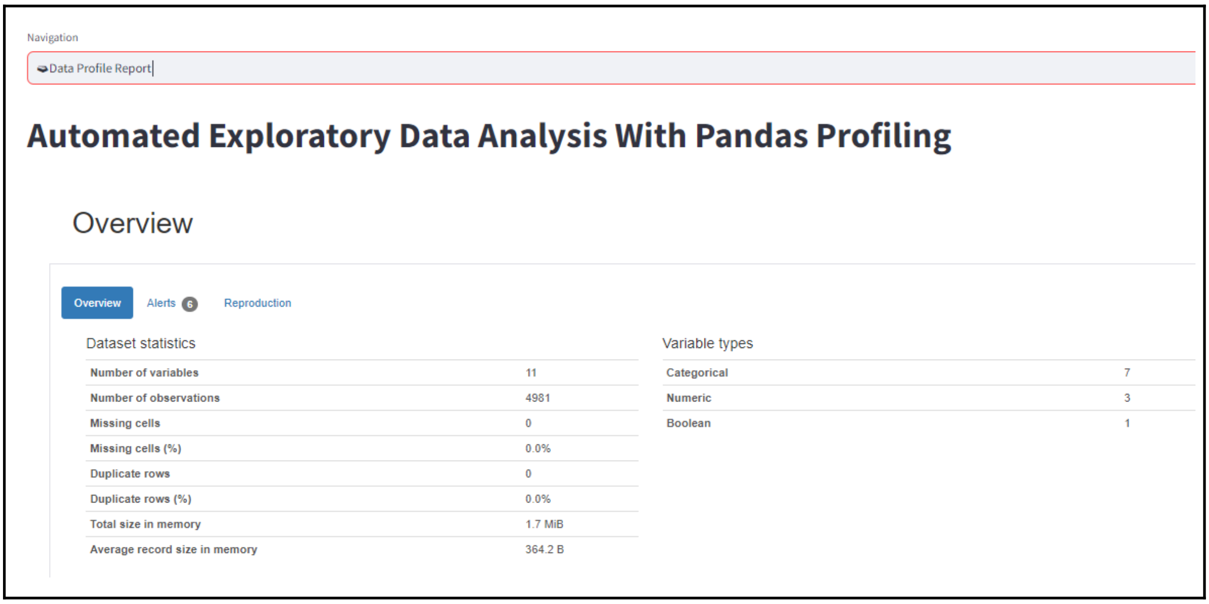
Quit smoking. 🚭

Control your blood pressure. 🩸

Get regular checkups. 🧑‍⚕️ 🩺

Anyone can enter their input and just with a prediction button can get a prediction if they are at a risk of brain stroke or not.

- Also to see the trained dataset’s analysis, I have added a Pandas profiling function which loads within seconds if you switch ti the second page.



Automated Exploratory Data Analysis With Pandas Profiling

Overview

Dataset statistics

Number of variables	11
Number of observations	4981
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.7 MiB
Average record size in memory	364.2 B

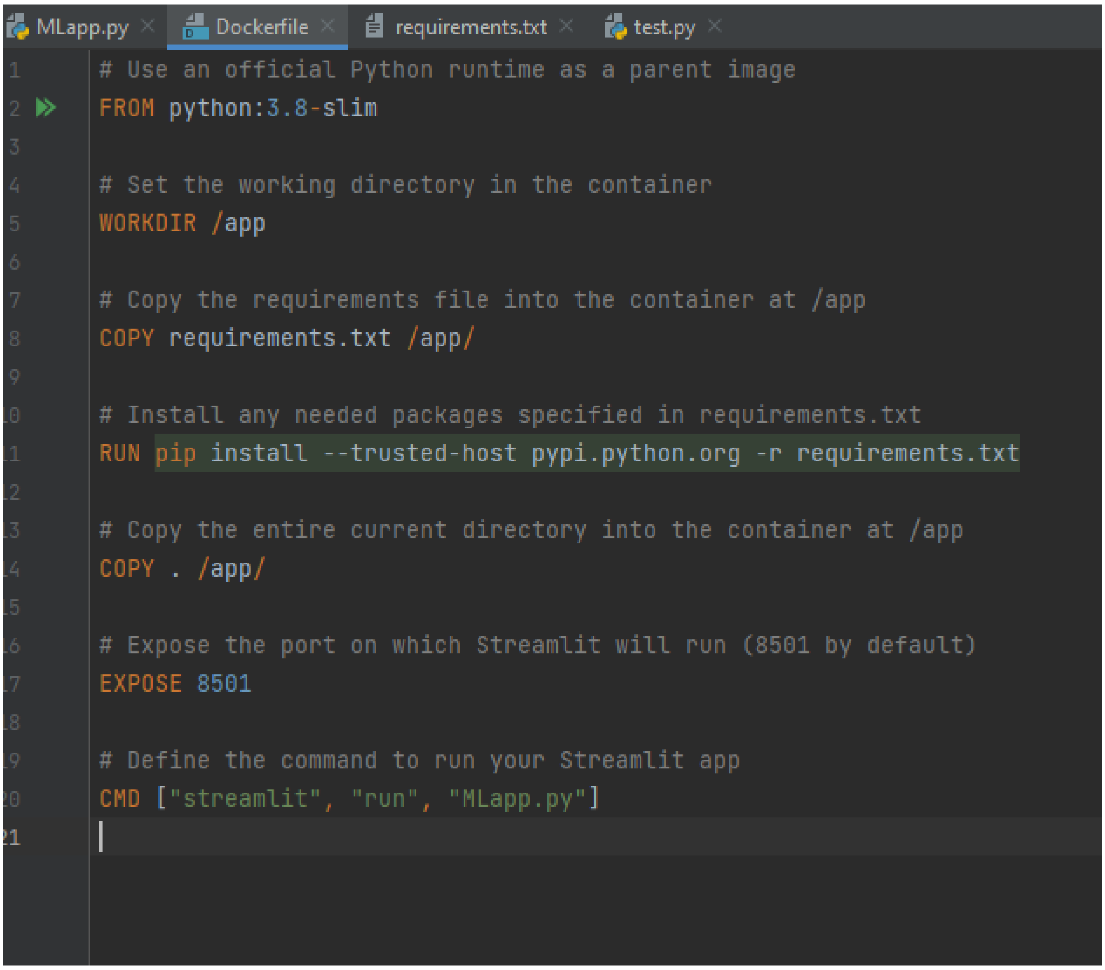
Variable types

Categorical	7
Numeric	3
Boolean	1

- In the 3rd page I have added a confusion matric plot of trained model of the brain stroke prediction. This trained model trained with Gradient Boosting classifier ML algorithm.

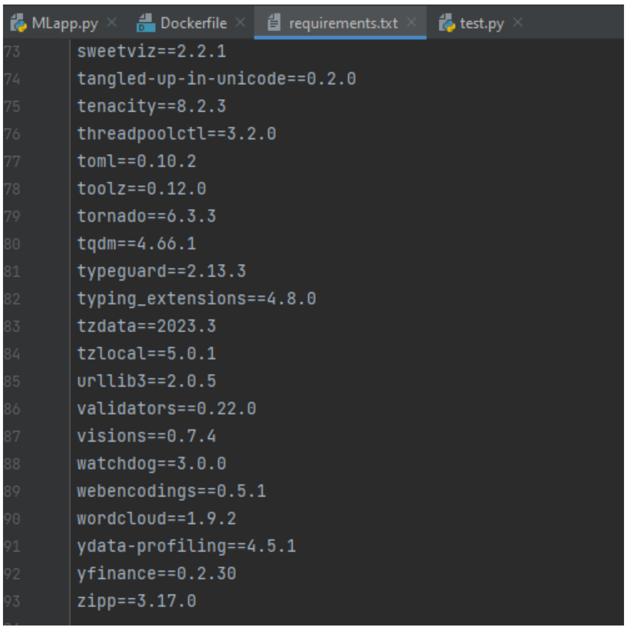
Deployement and Containerization

To deploy the python app we need to create a docker file using Docker.
In order complete this task I created a docker file in the same python workspace and entered the following instructions tot the same docker file.



```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt /app/

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Copy the entire current directory into the container at /app
COPY . /app/

# Expose the port on which Streamlit will run (8501 by default)
EXPOSE 8501

# Define the command to run your Streamlit app
CMD ["streamlit", "run", "MLapp.py"]
```

After that Created the requirements.txt file inorder to retrieve the python libraries used in the streamlit app.



```
sweetviz==2.2.1
tangled-up-in-unicode==0.2.0
tenacity==8.2.3
threadpoolctl==3.2.0
toml==0.10.2
toolz==0.12.0
tornado==6.3.3
tqdm==4.66.1
typeguard==2.13.3
typing_extensions==4.8.0
tzdata==2023.3
tzlocal==5.0.1
urllib3==2.0.5
validators==0.22.0
visions==0.7.4
watchdog==3.0.0
webencodings==0.5.1
wordcloud==1.9.2
ydata-profiling==4.5.1
yfinance==0.2.30
zipp==3.17.0
```

Work File Instructions

- App Pandas profiling function might disabled when data path changes.