ASSESSING THE RISK FACTORS ASSOCIATED WITH STROKE

TEAM MEMBERS

BHAVANA BETHI, KEERTHIKA SUNCHU, PALLAVI TELU, SRIJA DAMMANNAGARI, YAZNA PENMESTA

INDIANA UNIVERSITY-PURDUE UNIVERSITY INDIANAPOLIS, INDIANAPOLIS, INDIANA, USA

BBETHI@IU.EDU, KSUNCHU@IU.EDU, PTELU@IU.EDU, SRDAMMA@IU.EDU, YPENMETS@IU.EDU

INTRO I-501

PROFESSOR - ZEYANA HAMID

INTRODUCTION

- Using the 2015 BRFSS survey dataset, we employ machine learning to predict diabetes, hypertension, and stroke risks (Diabetes, Hypertension and Stroke Prediction, 2022).
- Modifiable factors like hypertension and diabetes significantly contribute to stroke morbidity and mortality (Volkow et al., 2017).
- The dataset covers crucial variables such as age, gender, BMI, smoking habits, physical activity, dietary choices, and medical histories (Diabetes, Hypertension and Stroke Prediction, 2022).
- Our goal is to distill insights for healthcare professionals, inform health policies, and empower individuals in managing these health risks effectively (Diabetes, Hypertension and Stroke Prediction, 2022).

STROKE

Definition

World Health Organization defined stroke as rapidly developed clinical signs of focal (or global) disturbance of cerebral function, lasting more than 24 hours or leading to death, with no apparent cause other than of vascular origin.

Symptoms

- Sudden confusion, trouble speaking, or trouble understanding speech
- Sudden numbness or weakness, especially on one side of the body
- Sudden severe headache with no known cause, trouble seeing from one or both eyes
- Sudden trouble walking, dizziness, or loss of balance or coordination

Risk Factors

High Blood pressure, Obesity, Physical inactivity, Poor diet, Smoking, Age and Sex

AIM

- Our aim is to investigate the impact of age, BMI, hypertension, smoking status, heart disease, and average glucose level on the risk of developing stroke in individuals.

PURPOSE

- We aim to investigate the influence of various factors on stroke occurrence and reveal insights into the relationships between these factors and health conditions. This knowledge will serve to inform healthcare providers, policymakers, and public health initiatives in developing tailored interventions for at-risk populations, ultimately reducing the burden of these diseases on individuals and society.

RESEARCH HYPOTHESIS

- **Null hypothesis (H0) -** The examined factors (sex, age, hypertension, heart disease, glucose level, BMI, smoking) are not associated with the risk of stroke.
- **Alternative hypothesis (H1) -** The examined factors (sex, age, hypertension, heart disease, glucose level, BMI, smoking) are associated with the risk of stroke.

METHODOLOGY

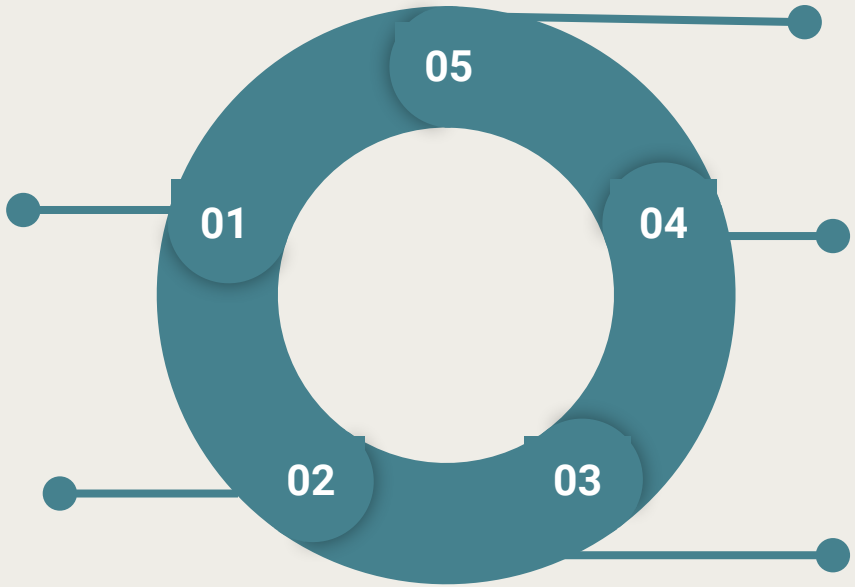
Data collection

Source - https://www.kaggle.com/datasets/prosperchuks/health-dataset/code

Data Cleaning

Data Preprocessing

Detecting the null values, outliers and replacing them.



Machine Learning

Classification Models

Model Building & Evaluation.

Exploratory Data Analysis & Visualization

Exploring data patterns & trends among Dependent and independent variables

Descriptive Statistics

Quantitative insights on data

DATA DESCRIPTION

The dataset comprises responses collected through the Behavioral Risk Factor Surveillance System (BRFSS) survey. We aim to determine the contributing factors to the occurrence of stroke. The variables encompass demographic information such as age and gender, heart disease, health-related indicators like BMI, average blood glucose levels, hypertension status, smoking habits. The data is available for download in CSV format.

CATEGORICAL DATA		NUMERICAL DATA
Binary	**Ordinal**	**Continuous**
Gender, Hypertension, Smoking status, heart disease, stroke, Ever married	Working Status	Average Glucose levels, BMI, Age

DATA CLEANING

- Loading Python Library

```
import pandas as pd
```

- Reading data using pandas

```
df = pd.read_csv('Stroke_data.csv')
df
```

	sex	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1.0	63	0	1	1	4	1	228.69	36.6	1	1
1	1.0	42	0	1	1	4	0	105.92	32.5	0	1
2	0.0	61	0	0	1	4	1	171.23	34.4	1	1
3	1.0	41	1	0	1	3	0	174.12	24.0	0	1
4	1.0	85	0	0	1	4	1	186.21	29.0	1	1
...	...	...	...	...	...	...	...	...	...	...	...
40905	1.0	38	0	0	0	4	1	120.94	29.7	1	0
40906	0.0	53	0	0	1	4	0	77.66	40.8	0	0
40907	1.0	32	0	0	1	2	0	231.95	33.2	0	0
40908	1.0	42	0	0	1	3	0	216.38	34.5	0	0
40909	1.0	35	0	0	0	4	0	95.01	28.0	0	0

40910 rows × 11 columns

- Checking for Null values
  We detected null values in the 'gender' column.

```
Null_values = df.isnull().sum()
Null_values
mode = df['sex'].mode().iloc[0]
mode
```

- Filling the Null values
  We imputed missing values in the 'gender' column with the mode.

```
df['sex'] = df['sex'].fillna(mode)
df
```

- Replaced the numerical data by categorical data

```
df['sex'] = df['sex'].replace({0: "female", 1: "male"})
df['hypertension'] = df['hypertension'].replace({0: "No", 1: "Yes"})
df['heart_disease'] = df['heart_disease'].replace({0: "No", 1: "Yes"})
df['ever_married'] = df['ever_married'].replace({0: "unmarried", 1: "married"})
df['work_type'] = df['work_type'].replace({0: "Never_worked", 1: "children", 2:"Govt_job", 3:"Self-employed", 4:"Private "})
df['Residence_type'] = df['Residence_type'].replace({0: "Rural", 1: "Urban"})
df['smoking_status'] = df['smoking_status'].replace({0: "Never smoked", 1: "smokes"})
df['stroke'] = df['stroke'].replace({0: "No", 1: "Yes"})

new_csv_file = 'STROKE_data.csv'
df.to_csv('STROKE_data.csv', encoding='utf-8')
```

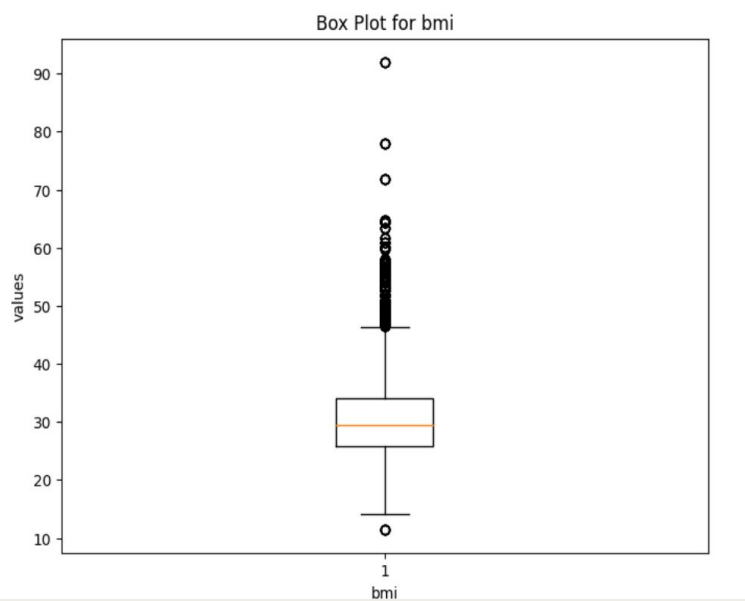
- Dropped the negative values

```
dataframe=df.loc[df['age'] >= 0]
dataframe
```

DETECTION OF OUTLIERS

- The BMI data was initially assessed for outliers using a box plot. Subsequently, using winsorization,we capped the data at 5th and 95th percentile. We have replaced the extreme values with adjusted values for a more accurate dataset.

```
import numpy as np
from scipy.stats.mstats import winsorize
import warnings
import pandas as pd

warnings.filterwarnings("ignore")
columns_of_interest = ['bmi']
# Set the winsorizing limits (capping at the 5th and 95th percentiles)
winsorizing_limits = [0.05, 0.05]

# Winsorize BMI column
df1['bmi'] = winsorize(df1['bmi'], limits=winsorizing_limits)

# Print or use the winsorized DataFrame
print(df1)
```



DESCRIPTIVE STATISTICS

- Identifying Dataset Dimensions
  Our dataset comprises 40,910 rows and 11 columns.

```
df.shape
```

```
(40910, 11)
```

- Counting Data Frame Elements
  The Data Frame contains a total of 450,010 elements.

```
df.size
```

```
450010
```

- Generating Descriptive Statistics
  The output from df1.describe() provides descriptive for each column within the DataFrame df1.

```
Statistics = df1.describe()
Statistics
```

- Verifying Data Types
  Within the Data Frame, there are 7 columns classified as 'object' type and 4 columns designated as 'float64'.

```
df1.dtypes
```

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

- Distribution based on Gender
  22,669 individuals are male, and 18,182 are female.

- Distribution based on Smoking Status
  50.44% of individuals have never smoked, and the remaining 49.56% are current smokers.

- Distribution based on Hypertension
  32,126 individuals are without hypertension, and 8,725 individuals have hypertension.

- Frequency of Heart Disease
  35,635 individuals do not have heart disease, while 5,216 individuals have a heart disease.

```
df1['sex'].value_counts()
```

```
sex
male      22713
female    18197
Name: count, dtype: int64
```

```
df1['smoking_status'].value_counts()
```

```
smoking_status
Never smoked    20921
smokes          19989
Name: count, dtype: int64
```

```
df1['hypertension'].value_counts()
```

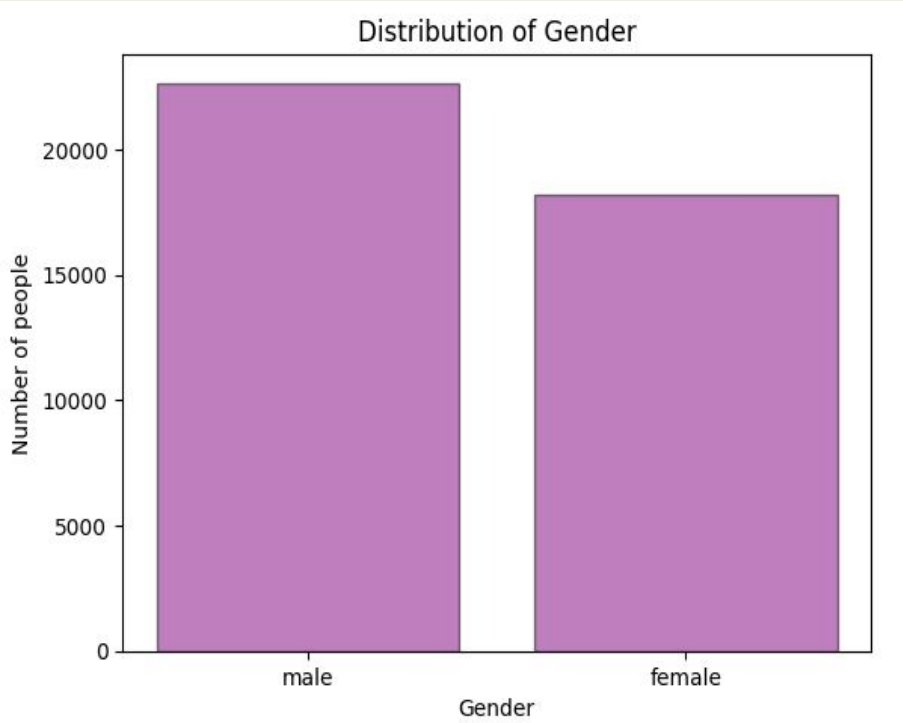
```
hypertension
No     32162
Yes     8748
Name: count, dtype: int64
```

```
df1['heart_disease'].value_counts()
```

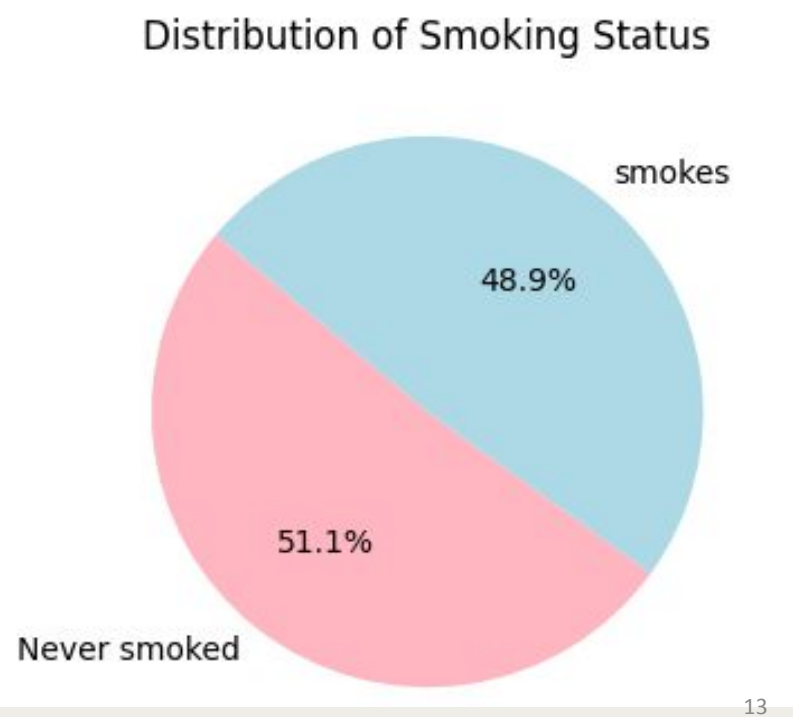
```
heart_disease
No     35685
Yes     5225
Name: count, dtype: int64
```

DESCRIPTIVE STATISTICS VISUALIZATION

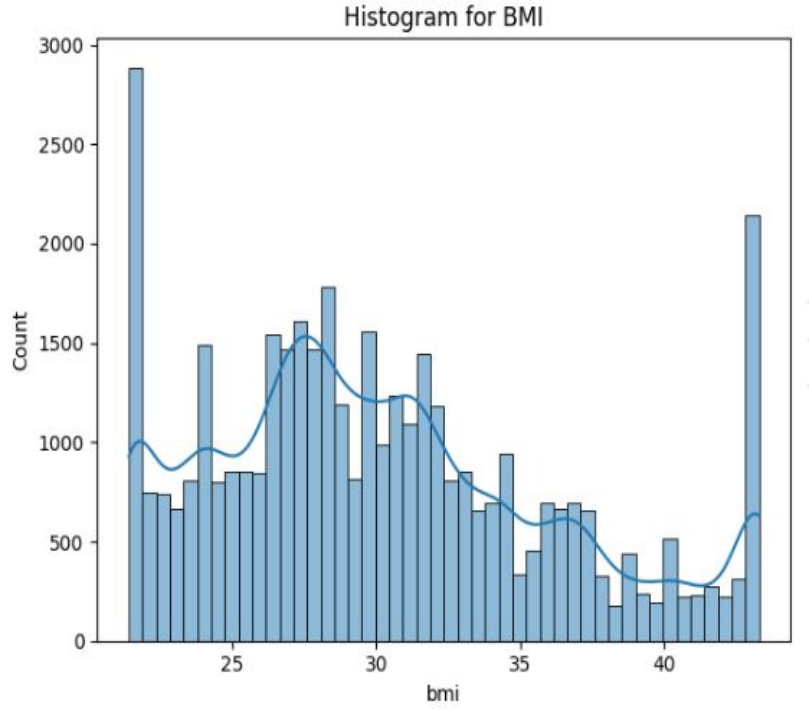
- Distribution of gender bar graph shows that 22,669 are male, and 18,182 are female.



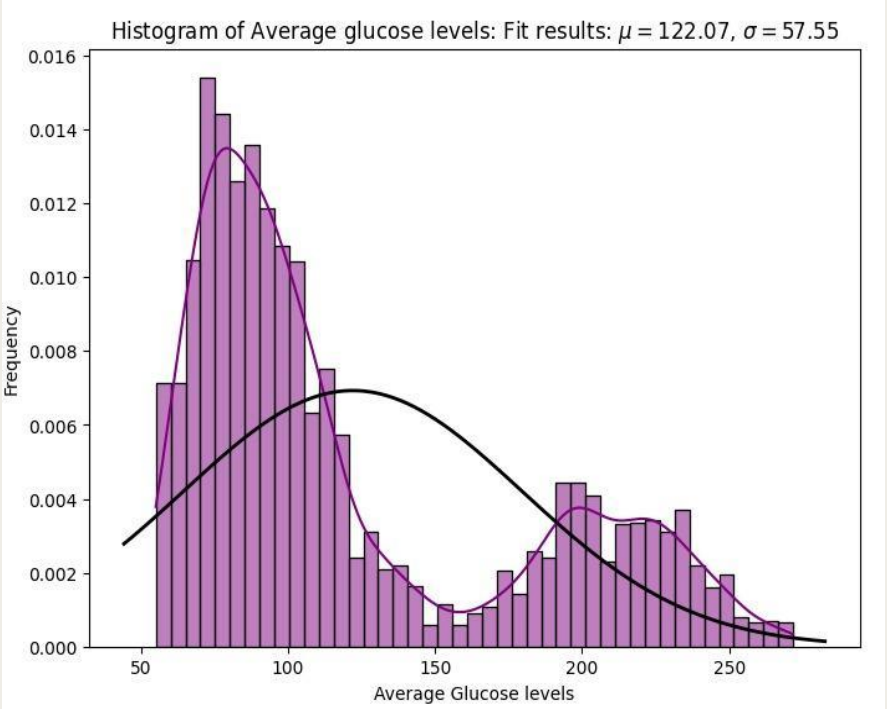
- Distribution of Smoking Status only 48.9% are current smokers



- The histogram shows that majority of people have bmi between 20 and 30

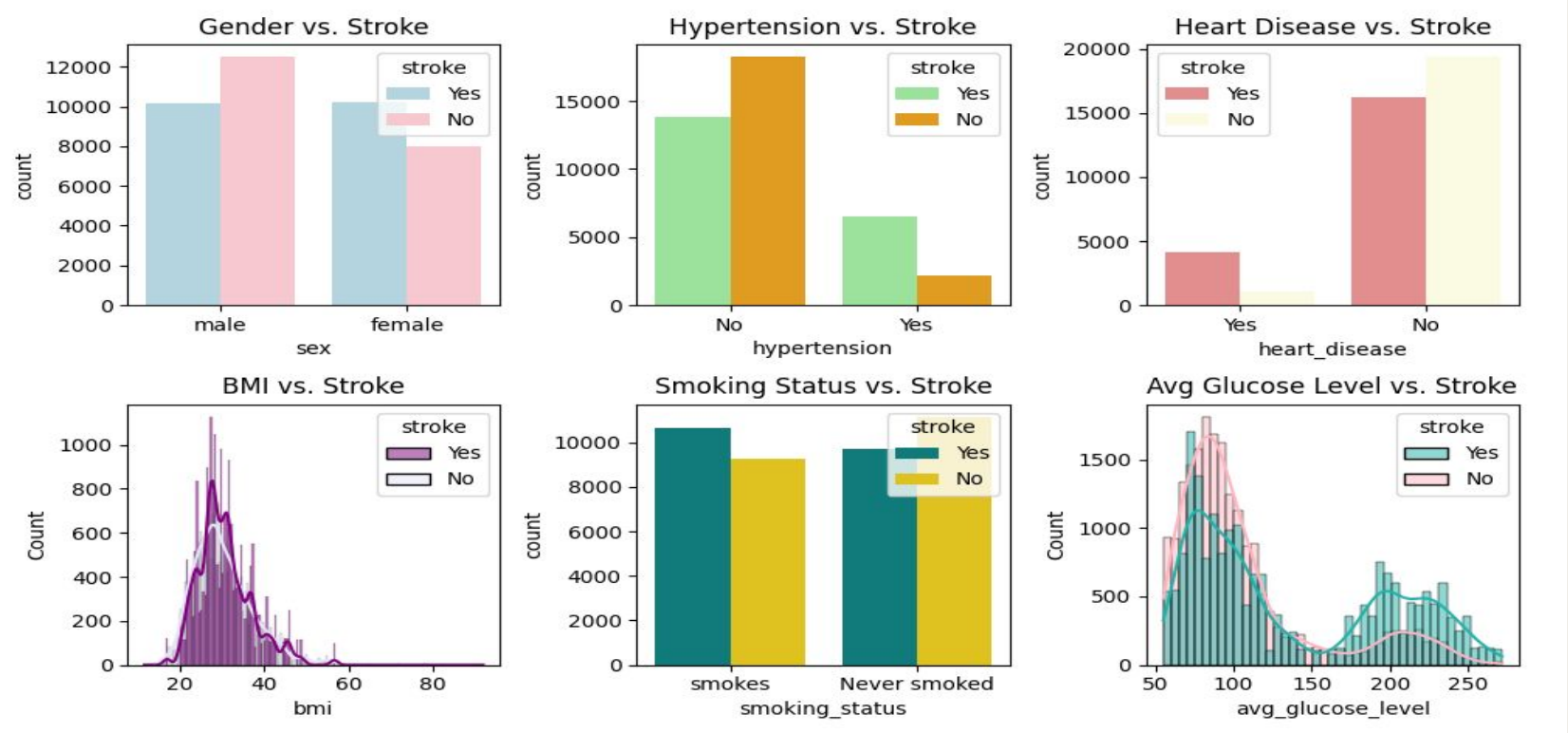


- Frequency distribution of Average glucose levels are 122mg/dl



EXPLORATORY DATA ANALYSIS

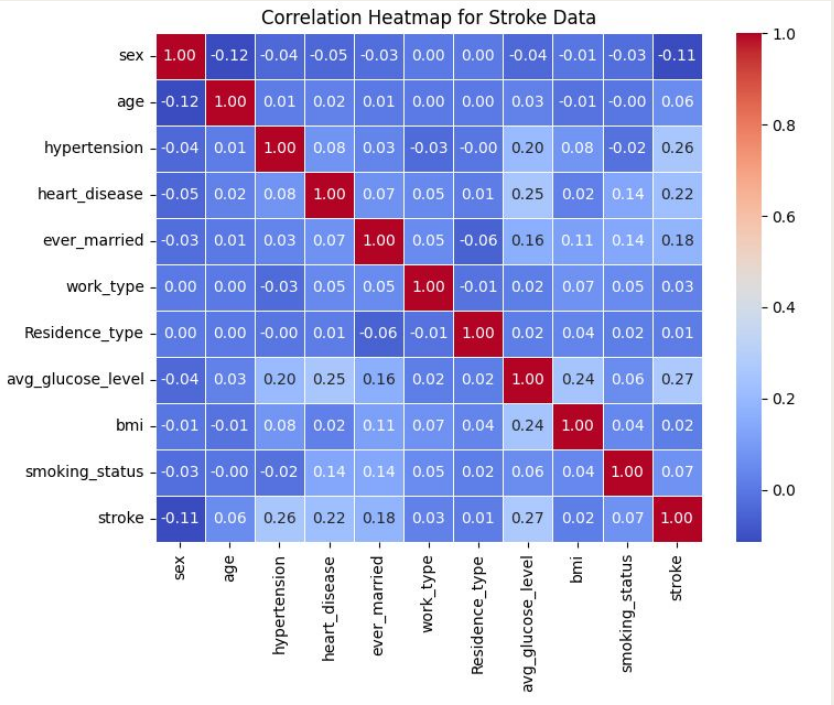
- There is a higher prevalence of smoking, hypertension among individuals who have experienced a stroke.



CORRELATION MATRIX

- We found strong positive correlations between stroke and hypertension, heart disease, average glucose level, and smoking status, and a weak negative correlation between stroke and age.

```
import seaborn as sns
import matplotlib.pyplot as plt
data = pd.read_csv('Stroke_data.csv')
correlation_matrix = data.corr()
plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap for Stroke Data')
plt.show()
print("Correlation Coefficients:")
print(correlation_matrix)
```

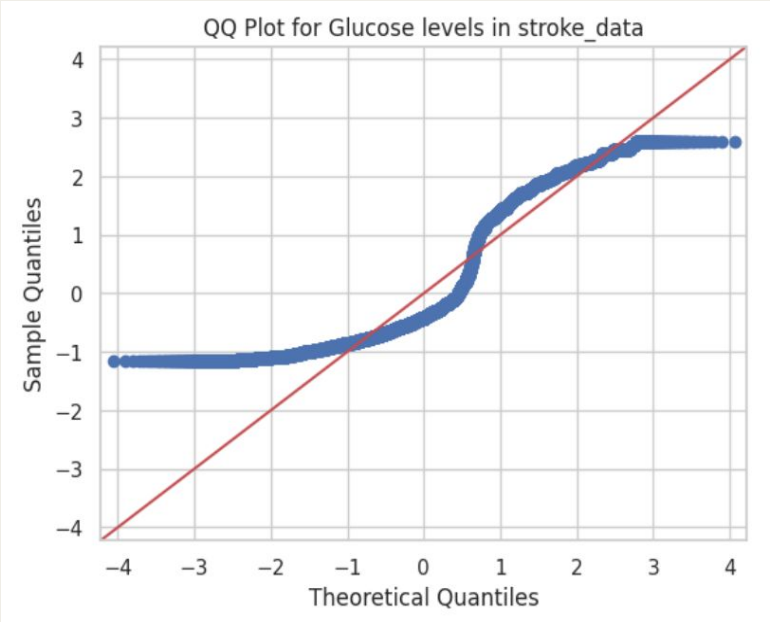


NORMALITY TEST

- The test yielded a statistic of 0.8441 and a p-value of 0.0.
- As p-value is less than the significance level, we reject the null hypothesis.
- This suggests that the BMI data does not follow a normal distribution.

```
statistic, p_value = shapiro(df1['avg_glucose_level'])
alpha = 0.05
print(f'Statistic: {statistic}, p-value: {p_value}')
if p_value > alpha:
    print("Fail to reject the null hypothesis. The data appears to be normally distributed.")
else:
    print("Reject the null hypothesis. The data does not appear to be normally distributed.")
```

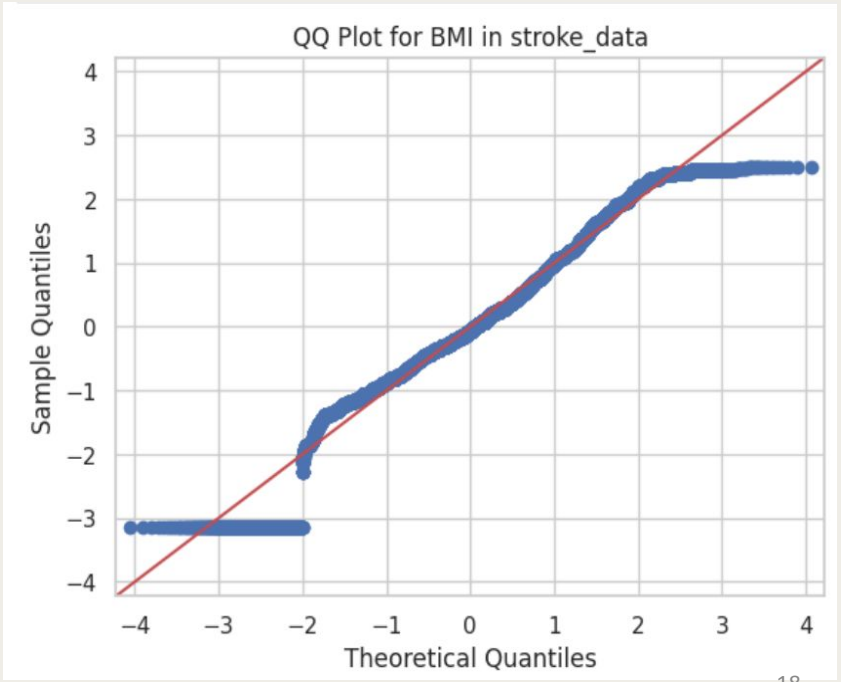
```
Statistic: 0.844083309173584, p-value: 0.0
Reject the null hypothesis. The data does not appear to be normally distributed.
```



- The test resulted in a statistic of 0.9715 and a p-value of 0.0, indicating rejection of the null hypothesis.
- The BMI data doesn't follow a normal distribution. The QQ plot illustrates right skewness, indicating more higher BMIs, aligning with obesity as a stroke risk factor.

```
import pandas as pd
from scipy.stats import shapiro

statistic, p_value = shapiro(df1['bmi'])

alpha = 0.05

print(f'Statistic: {statistic}, p-value: {p_value}')

if p_value > alpha:
    print("Fail to reject the null hypothesis. The data appears to be normally distributed.")
else:
    print("Reject the null hypothesis. The data does not appear to be normally distributed.")
```

```
Statistic: 0.971524715423584, p-value: 0.0
```



STATISTICAL TESTING

Chi square test

- This is performed using the `chi2_contingency` function.
- It assesses the association between categorical variables in the `Relation` contingency table.
- The test helps determine whether there is a significant relationship between the variables in our dataset.
- The test outputs the chi-square statistic (`Chi`), p-value (`P`), degrees of freedom (`DoF`), and the expected frequency of observations under the assumption of independence.
- A low p-value suggests that there is a significant association, indicating the presence of a relationship between the categorical variables in the dataset.

- The extremely small p-value (1.0152375803583455e-111) suggests that the observed association between gender and stroke is highly significant.
  Therefore, based on the data, we reject the null hypothesis.

```
Relation= pd.crosstab(df1.sex, df1.stroke)
Relation
```

stroke	No	Yes
sex		
female	7967	10215
male	12482	10187

```
from scipy.stats import chi2_contingency
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
Chi: 509.7817154145831
P: 7.074685852294051e-113
DoF: 1
Expected frequency:  [[ 9101.45940124  9080.54059876]
 [11347.54059876 11321.45940124]]
```

- P-value is less than 0.05.
  Therefore, we reject the null hypothesis and there is a significant association between hypertension and stroke.

```
Relation= pd.crosstab(df1.hypertension, df1.stroke)
Relation
```

stroke	No	Yes
hypertension		
No	18233	13893
Yes	2216	6509

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
Chi: 2697.302925290711
P: 0.0
DoF: 1
Expected frequency:  [[16081.48084502 16044.51915498]
 [ 4367.51915498  4357.48084502]]
```

- As P-value is less than 0.05, we reject the null hypothesis.

  There is a significant association between heart disease and stroke.

```
Relation= pd.crosstab(df1.smoking_status, df1.stroke)
Relation
```

stroke	No	Yes
smoking_status		
Never smoked	11157	9729
smokes	9292	10673

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
Chi: 192.8304265870189
P: 7.6653756901355598e-44
DoF: 1
Expected frequency:  [[10455.01490784 10430.98509216]
 [ 9993.98509216  9971.01490784]]
```

- The extremely small p-value P: 1.972297048410639e-43 suggests that the observed association between gender and stroke is highly significant.
  Therefore, based on the data, we reject the null hypothesis.

```
Relation= pd.crosstab(df1.heart_disease, df1.stroke)
Relation
```

stroke	No	Yes
heart_disease		
No	19366	16269
Yes	1083	4133

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
Chi: 2051.217299112176
P: 0.0
DoF: 1
Expected frequency:  [[17837.99943698 17797.00056302]
 [ 2611.00056302  2604.99943698]]
```

- The chi-square test for BMI and Average Glucose levels yields a significant result (p = 0.0), rejecting the null hypothesis.
- The observed and expected frequencies differ, suggesting a relationship between variables in the dataset.

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
Chi: 13920.253205630004
P: 0.0
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
Chi: 39838.94536612531
P: 0.0
```

ML MODELS

Classification Models

- **Logistic Regression:** Estimates the probability of a binary outcome by fitting data to a logistic function. Commonly used in classification problems with two discrete classes.
- **Cat-Boost:** A high-performance gradient-boosting algorithm optimized for categorical variables. Delivers robust predictions, especially with datasets containing categories.
- **Random Forest:** Creates an ensemble of decision trees during training and aggregates their results, improving accuracy. Applicable for both classification and regression predictive modeling.
- **K Nearest Neighbors:** A non-parametric supervised learning method that predicts points by identifying their closest neighbors and taking a vote. Classifies data based on proximity in feature space.

SMOTE

- Through value counts, it is identified that our data set is balanced.
- We prioritized natural distribution, ensuring improved accuracy and reliability in our model by not including smote.

```
y=df1['stroke']
print(y.value_counts())
print(y.value_counts(dropna=False, normalize=True)*100)
```

```
stroke
Yes    20460
No     20450
Name: count, dtype: int64
stroke
Yes    50.012222
No     49.987778
Name: proportion, dtype: float64
```

DEPENDENT VARIABLE

```
y = df2['stroke']
```

- The code `y = df2['stroke']` designates 'stroke' as our dependent variable, enabling its isolation for further analysis or modeling.

TRAIN-TEST SPLIT

```
X_train, X_test, y_train, y_test_log = train_test_split(x, y, test_size=0.2, random_state=42)
```

- This code utilizes the `train_test_split` function from the `sklearn.model_selection` library to divide the dataset into training and testing sets.
- The `train_test_split` function is employed to partition a dataset into training and testing subsets, facilitating model evaluation and validation.

LOGISTIC REGRESSION

- Imported libraries train_test_split from sklearn.model_selection, LogisticRegression f sklearn.linear_model, confusion_matrix and classification_report from sklearn.metrics

- Logistic Regression Model Building and Splitting the Data using the train_test_split function

- Evaluating the Model by Creating a confusion matrix

- Model Performance Metrics

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']
```

```
X_train, X_test, y_train, y_test_log = train_test_split(X, y, test_size=0.2, random_state=42)
logreg_model = LogisticRegression(max_iter=1000,random_state=42)
logreg_model.fit(X_train, y_train)
y_pred_log = logreg_model.predict(X_test)
```

```
conf_matrix = confusion_matrix(y_test_log, y_pred_log)
TN, FP, FN, TP = conf_matrix.ravel()
```

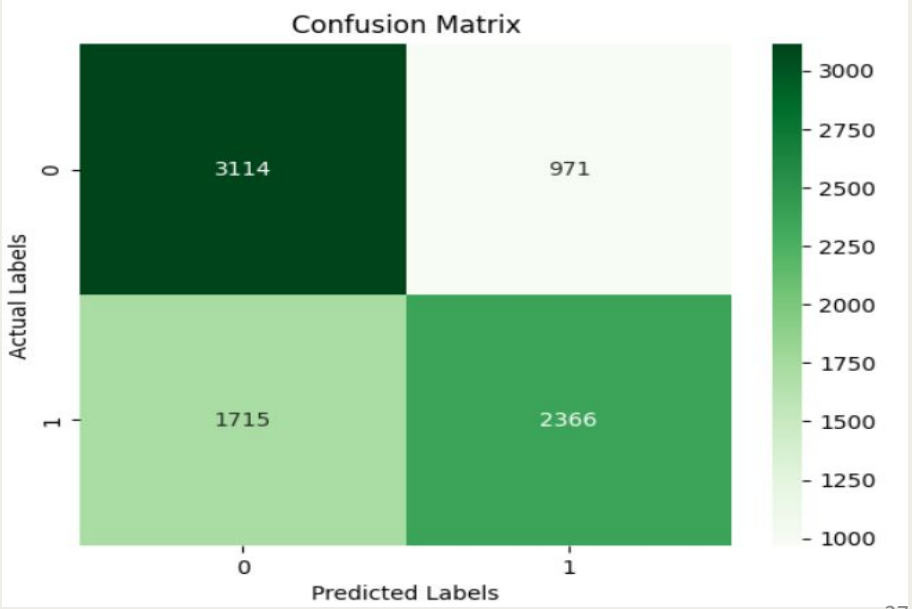
```
accuracy = accuracy_score(y_test_log, y_pred_log)
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test_log, y_pred_log))
sensitivity = TP / (TP + FN)
print("Sensitivity:", sensitivity)
specificity = TN / (TN + FP)
print("Specificity:", specificity)
```

CONFUSION MATRIX VISUALIZATION

The confusion matrix assesses logistic regression model performance. It displays following values.

- Sensitivity: 0.579 (57.9%)
- Specificity: 0.762 (76.2%)
- True Positives (TP): 3114,
- True Negatives (TN): 2366
- False Positives (FP): 1715
- False Negatives (FN): 971.

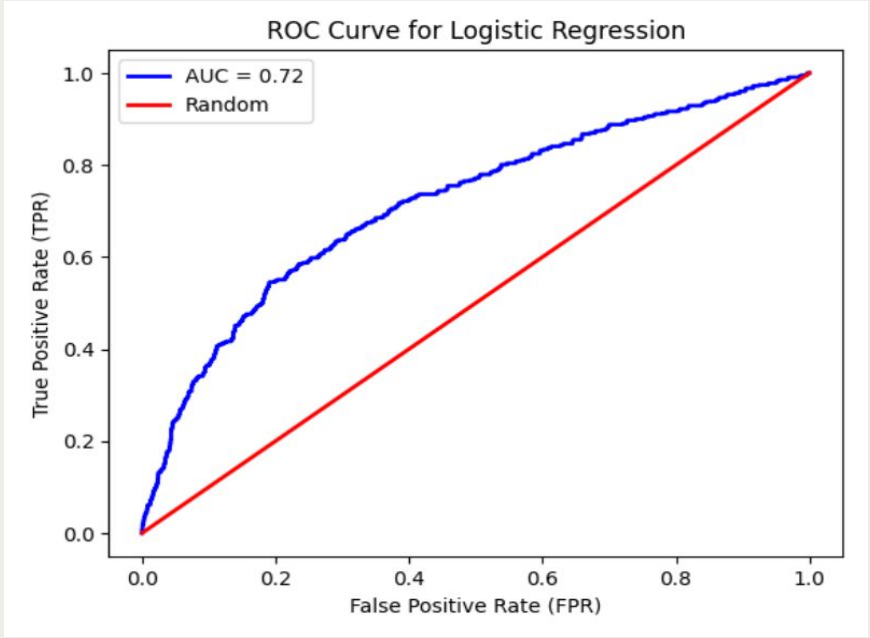
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')
plt.show()
```



CLASSIFICATION REPORT AND ROC CURVE FOR LOGISTIC REGRESSION

Classification Report:

	precision	recall	f1-score	support
0	0.64	0.76	0.70	4085
1	0.71	0.58	0.64	4081
accuracy			0.67	8166
macro avg	0.68	0.67	0.67	8166
weighted avg	0.68	0.67	0.67	8166



- Accuracy of the model is 68%.
- Area under the curve is 0.72.

CAT BOOST CLASSIFIER

- Import the catboost libraries, including, CatBoostClassifier, Pool from catboost

```
import catboost
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']
```

- Splitting the Data using the train_test_split function

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Model Performance Metrics

```
model = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, loss_function='Logloss',eval_metric='Accuracy', random_seed=42,verbose=False)
model.fit(train_pool)
```

- Make predictions on the test set

```
y_pred = model.predict(X_test)
```

- Evaluating the Model

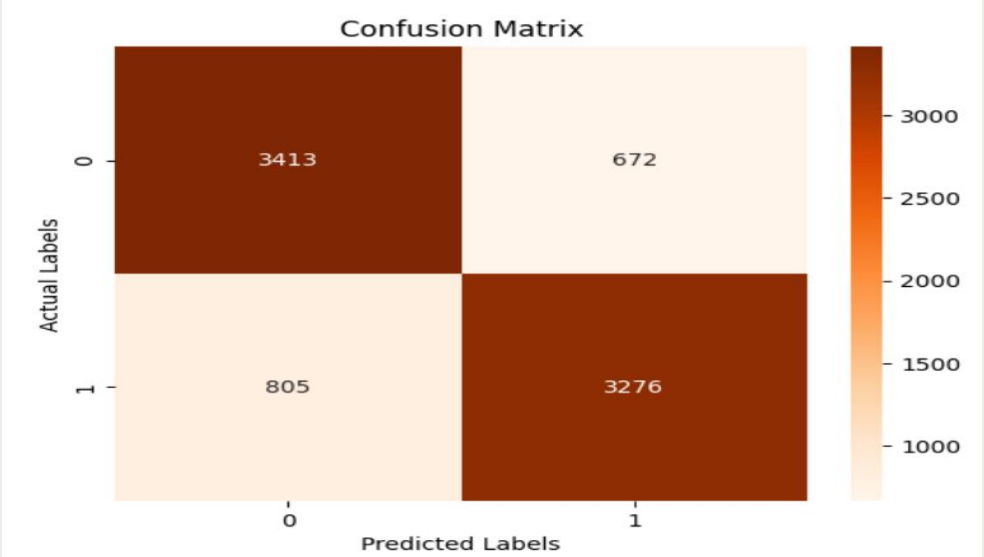
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

CONFUSION MATRIX VISUALIZATION

The confusion matrix assesses CatBoost Classifier performance. It displays the following values.

- True Positives: 3414
- True Negatives: 3276
- False Positives: 672
- False Negative: 805

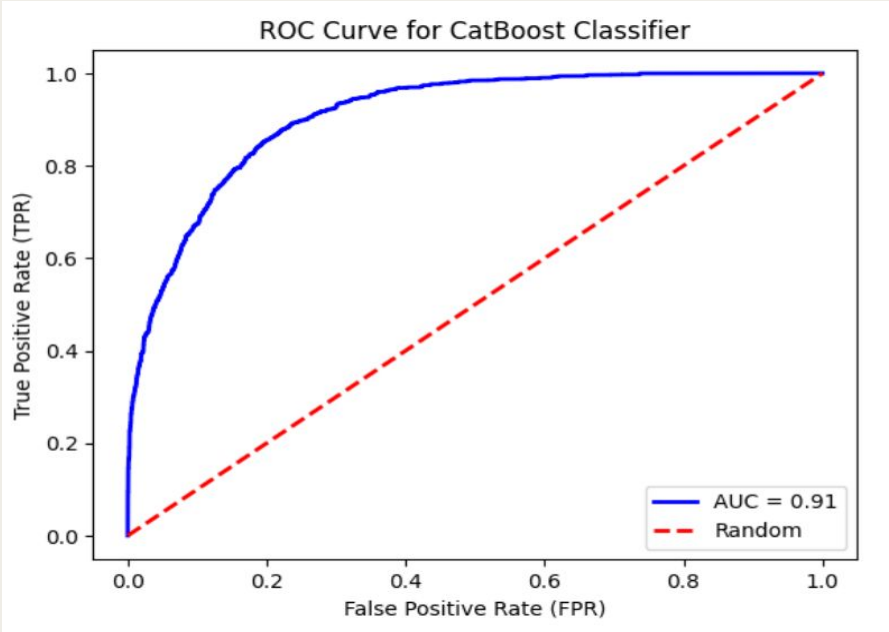
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Oranges')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')
plt.show()
```



CLASSIFICATION REPORT AND ROC CURVE FOR CATBOOST

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.84	0.82	4085
1	0.83	0.80	0.82	4081
accuracy			0.82	8166
macro avg	0.82	0.82	0.82	8166
weighted avg	0.82	0.82	0.82	8166



- Accuracy of the model is 82%
- Area under the curve value is 0.91 indicating excellent discriminatory power

RANDOM FOREST CLASSIFIER

- Imported RandomForestClassifier from the sklearn.ensemble library

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

- Logistic Regression Model Building and Splitting the Data using the train_test_split

```
X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']
```

- Creating the Random Forest Classifier

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
```

- Fitting the Model

```
model.fit(X_train, y_train)
```

- Making Predictions

```
predictions = model.predict(X_test)
```

- Evaluating the Model by Creating a confusion

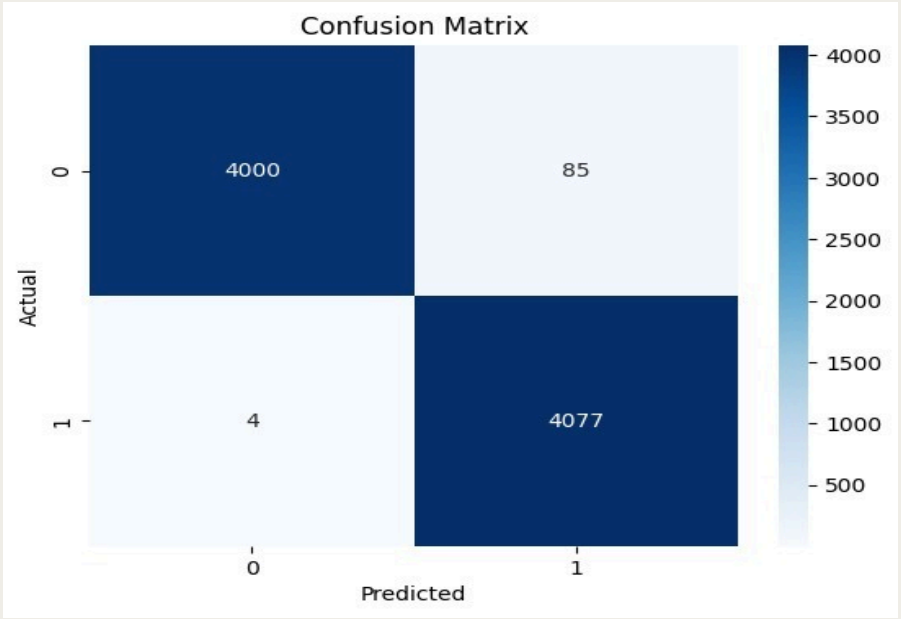
```
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy}')
```

CONFUSION MATRIX VISUALIZATION

The confusion matrix for the Random Forest Classifier reveals the following data.

- True Positives:4077
- True Negatives:4000
- False Positives: 85
- FalseNegatives:4

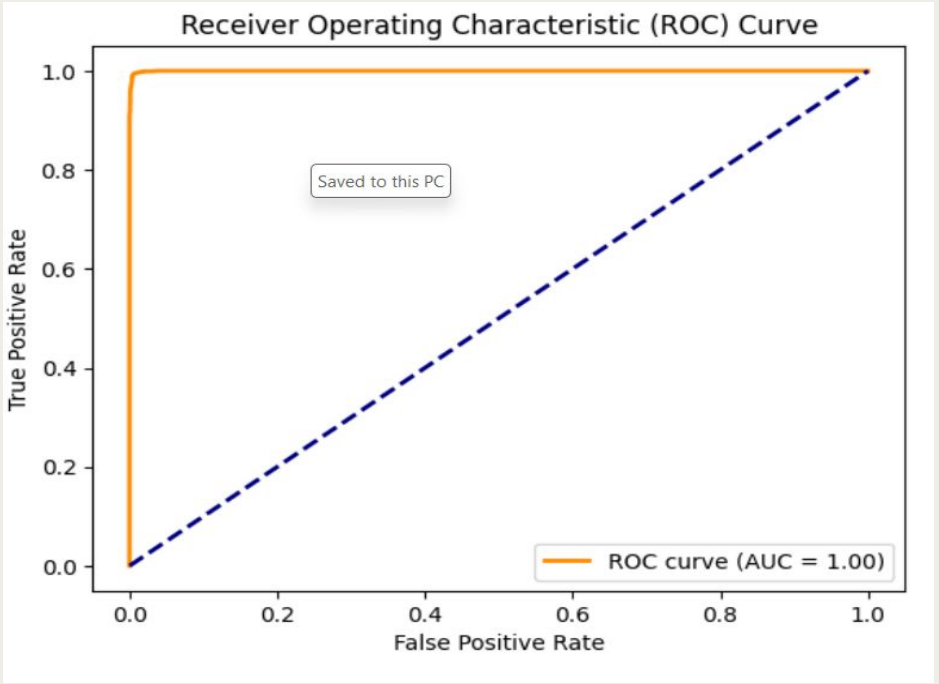
```
print('\nClassification Report:\n', classification_report(y_test, predictions))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



CLASSIFICATION REPORT AND ROC CURVE FOR RANDOM FOREST

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.98	0.99	4085
1	0.98	1.00	0.99	4081
accuracy			0.99	8166
macro avg	0.99	0.99	0.99	8166
weighted avg	0.99	0.99	0.99	8166



- Accuracy of the model is 99%
- Area under the curve value is 1 indicating an outstanding model

CROSS VALIDATION

- We stratified k-fold cross-validation to assess model generalization and mitigate overfitting risks by evaluating the mean Area Under the Curve (AUC).

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')
for i, auc in enumerate(cv_results, 1):
    print(f'Fold {i}: AUC = {auc:.4f}')
print(f'Mean AUC: {cv_results.mean():.4f}')
```

- The mean AUC is exceptionally high at 0.9998 indicating robust model performance.

```
Fold 1: AUC = 0.9998
Fold 2: AUC = 0.9999
Fold 3: AUC = 0.9997
Fold 4: AUC = 0.9998
Fold 5: AUC = 0.9998
Mean AUC: 0.9998
```

K NEAREST ALGORITHM

- Imported RandomForestClassifier from the sklearn.ensemble library

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, roc_auc_score
```

- K nearest Model Building

```
X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']
```

- Creating k –N N Classifier

```
k_value = 3
model = KNeighborsClassifier(n_neighbors=k_value)
```

- Fitting the Model

```
model.fit(X_train, y_train)
```

- Making Predictions

```
predictions = model.predict(X_test)
```

- Evaluating the Model by Creating a confusion matrix

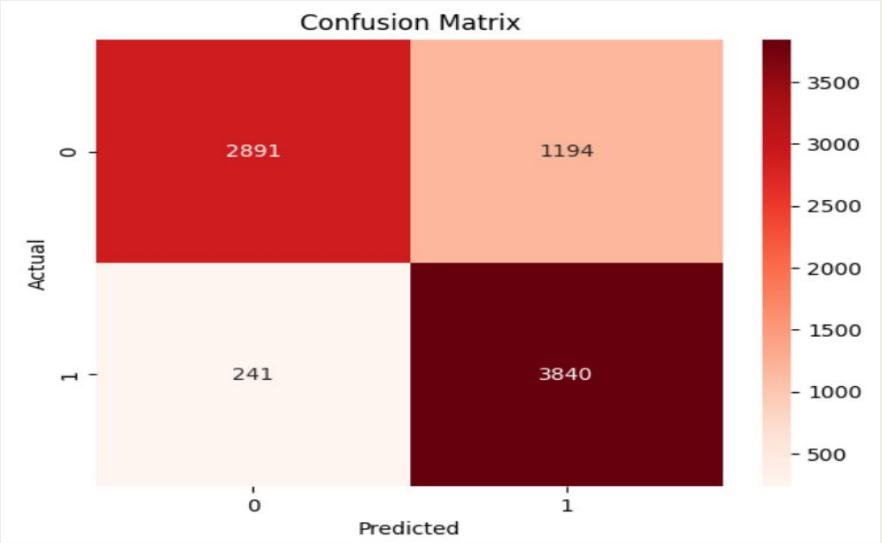
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

CONFUSION MATRIX VISUALIZATION

The confusion matrix for the K-Nearest Neighbors (k-NN) model illustrates the following values

- True Positives: 2891
- True Negatives: 3840
- False Positives: 1194
- False Negatives: 241

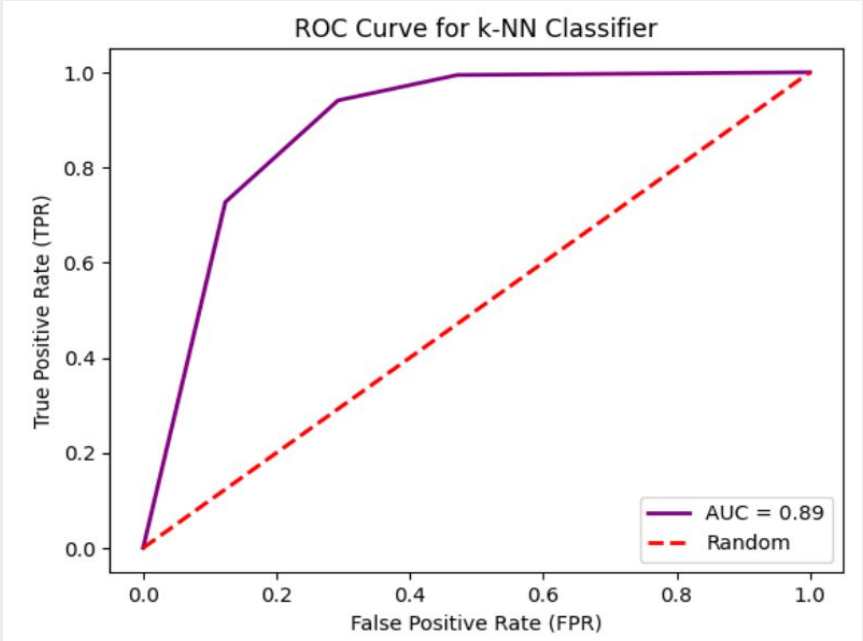
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



CLASSIFICATION REPORT AND ROC CURVE FOR KNN CLASSIFIER

Classification Report:

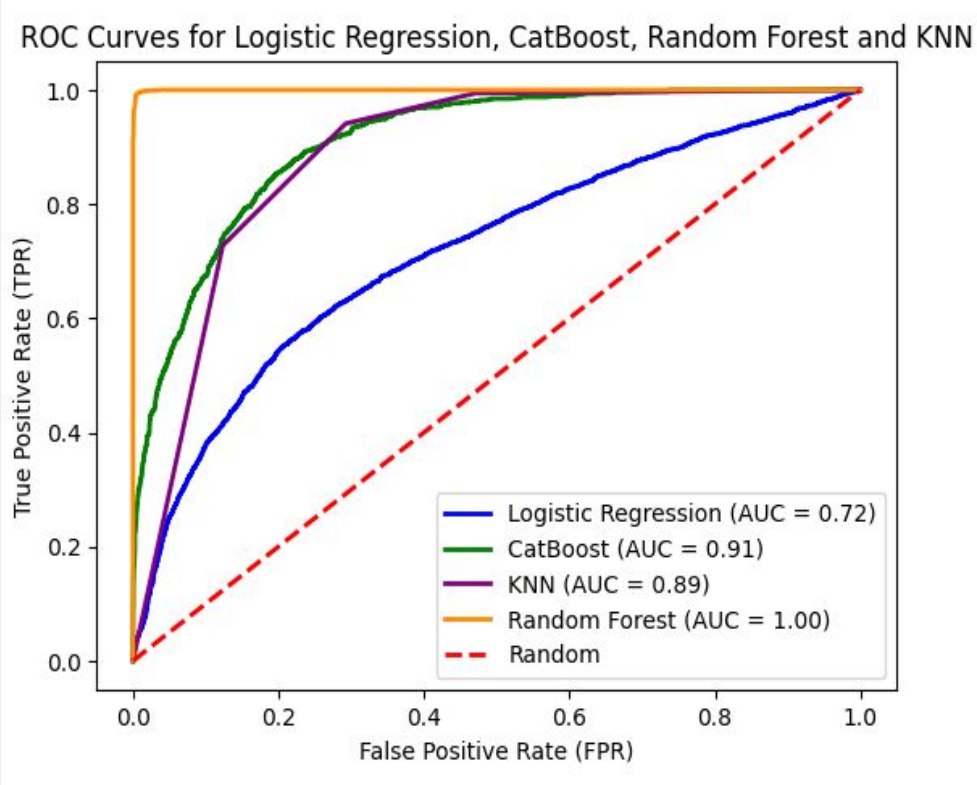
	precision	recall	f1-score	support
0	0.92	0.71	0.80	4085
1	0.76	0.94	0.84	4081
accuracy			0.82	8166
macro avg	0.84	0.82	0.82	8166
weighted avg	0.84	0.82	0.82	8166



- Accuracy of the model is 84%
- Area under the curve value is 0.89 indicating the model has excellent discrimination.

ROC CURVES FOR ALL MODELS

- The ROC curves reveal performance of all models.
- Logistic Regression has an AUC of 0.72
- Random Forest achieved high accuracy.
- KNN showed strong discrimination ability with an AUC of 0.89.
- CatBoost performed exceptionally well with an AUC of 0.91.



RESULT

- We reject the null hypothesis, signifying an underlying association among sex, age, hypertension, heart disease, glucose level, BMI, smoking, and the incidence of stroke.
- The Random Forest model demonstrates superior accuracy, precision, and F1 score compared to other models, underscoring its efficacy in predicting and understanding the complex relationships within the dataset.

LIMITATIONS

- We considered consolidating datasets, but lacking unique columns for merging, we opted to use only one dataset.
- Additionally, upon closer examination, we identified inconsistencies in the age values within our selected dataset.

REFERENCES

Alloubani, A., Saleh, A., Abdelhafiz, I., & Abdelhafiz, A. (2018). The Impact of Diabetes Mellitus on Stroke: A Review. Journal of Stroke and Cerebrovascular Diseases, 27(7), 1970-1978.

Diabetes, hypertension and stroke prediction. (2022, December 19). Kaggle. https://www.kaggle.com/datasets/prosperchuks/health-dataset?select=diabetes_data.csv

Volkow, N. D., Koob, G. F., & McLellan, A. T. (2017). Neurobiologic Advances from the Brain Disease Model of Addiction. New England Journal of Medicine, 374(4), 363-371.

THANK YOU

APPENDIX

```
pip install mysql-connector-python
import mysql.connector
import pandas as pd
```

```
# Replace the placeholders with your actual database credentials
db_config = {
    'host': 'localhost',
    'user': 'bbethi',
    'password': 'chrysalis steerage odometer',
    'database': 'I501_Fall2023_Sec22490_group04_db'
}

# Establish a connection to the database
connection = mysql.connector.connect(**db_config)

try:
    # Create a cursor object to execute SQL queries
    cursor = connection.cursor()

    # Example query: Select all rows from a table
    query = 'SELECT * FROM stroke_data'



    # Now, you can use Pandas to read data from the database
    df = pd.read_sql(query, connection)
    print("DataFrame from SQL:")
    print(df)

finally:
    # Close the cursor and connection
    cursor.close()
    connection.close()
```

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('Stroke_data.csv')
df
```

```
Null_values = df.isnull().sum()
Null_values
```

```
mode = df['sex'].mode().iloc[0]
mode
```

```
df['sex'] = df['sex'].fillna(mode)
```

```
df.shape
```

```
df.size
```

```
df.info()
```

```
df.duplicated().sum()
```

```
df.dtypes
```

```
df.isnull()
```

```
Statistics = df.describe()
Statistics
```

```
df=df.loc[df['age'] > 0]
df
```

```
df1 = pd.read_csv('STROKE_data.csv')
df1
```

```
del df1['Unnamed: 0']
```

```
df1.shape
```

```
df1['sex'].value_counts()
```

```
df1['hypertension'].value_counts()
```

```
df['sex'] = df['sex'].replace({0: "female", 1: "male"})
df['hypertension'] = df['hypertension'].replace({0: "No", 1: "Yes"})
df['heart_disease'] = df['heart_disease'].replace({0: "No", 1: "Yes"})
df['ever_married'] = df['ever_married'].replace({0: "unmarried", 1: "married"})
df['work_type'] = df['work_type'].replace({0: "Never_worked", 1: "children", 2:"Govt_job", 3:"Self-employed", 4:"Private "})
df['Residence_type'] = df['Residence_type'].replace({0: "Rural", 1: "Urban"})
df['smoking_status'] = df['smoking_status'].replace({0: "Never smoked", 1: "smokes"})
df['stroke'] = df['stroke'].replace({0: "No", 1: "Yes"})

new_csv_file = 'STROKE_data.csv'
df.to_csv('STROKE_data.csv', encoding='utf-8')
```

```
df1['heart_disease'].value_counts()
```

```
df1['ever_married'].value_counts()
```

```
df1['work_type'].value_counts()
```

```
df1['Residence_type'].value_counts()
```

```
df1['smoking_status'].value_counts()
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.boxplot(df1['age'])
plt.title('Box Plot for age')
plt.xlabel('Age')
plt.ylabel('Age (yrs)')
plt.show()
```

```
import numpy as np
from scipy.stats.mstats import winsorize
import warnings
import pandas as pd

warnings.filterwarnings("ignore")
columns_of_interest = ['bmi']
# Set the winsorizing limits (capping at the 5th and 95th percentiles)
winsorizing_limits = [0.05, 0.05]

# Winsorize BMI column
df1['bmi'] = winsorize(df1['bmi'], limits=winsorizing_limits)

# Print or use the winsorized DataFrame
print(df1)
```

```
import pandas as pd
import numpy as np

# Assuming 'bmi_column' is the column containing BMI data in your DataFrame
bmi_data = df1['bmi']

# Calculate IQR
Q1 = bmi_data.quantile(0.25)
Q3 = bmi_data.quantile(0.75)
IQR = Q3 - Q1
print(Q1)
print(Q3)
print(IQR)
print(np.median(bmi_data))

# Define lower and upper bounds
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + 1.5 * IQR

# Replace outliers with the median
bmi_data_no_outliers = np.where((bmi_data < lower_bound) | (bmi_data > upper_bound), np.median(bmi_data), bmi_data)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.boxplot(df1['bmi'])
plt.title('Box Plot for bmi')
plt.xlabel('bmi')
plt.ylabel('values')
plt.show()
```

```
import matplotlib.pyplot as plt
values = df1['age']
plt.hist(values, bins = 'auto', color='purple', edgecolor='black', alpha = 0.7)
plt.xlim(0, 100)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
```

```
import matplotlib.pyplot as plt
gender_proportion = df1['sex'].value_counts()
Gender = gender_proportion.index
Count = gender_proportion.values
plt.bar(Gender, Count, color='purple', edgecolor='black', alpha = 0.7)
plt.title('Distribution of Gender')
plt.xlabel('Gender')
plt.ylabel('Number of people')
plt.show()
```

```
Count = df1['stroke'].value_counts()
Count
```

```
plt.close()
import matplotlib.pyplot as plt
labels = ['STROKE', 'NO-STROKE']
Count = [df1['stroke'].value_counts()['Yes'], df1['stroke'].value_counts()['No']]
colors =  ['lightpink', 'lightblue']
plt.figure(figsize=(5,5))
plt.pie(Count, labels=labels, colors=colors, autopct = '%1.1f%%', startangle=75)
plt.title("Distribution of Stroke")
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.DataFrame({'sex': ['male', 'male', 'female', 'female', 'male', 'female'],
                   'stroke': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No']})
stroke_counts = df.groupby(['sex', 'stroke']).size()
stroke_counts_stroke = stroke_counts['male']['Yes'], stroke_counts['female']['Yes']
gender_labels = ['male', 'female']
colors = ['lightblue', 'pink']
plt.figure(figsize=(5,5))
plt.pie(stroke_counts_stroke, labels=gender_labels, colors=colors, autopct='%1.1f%%', startangle=180)
plt.title("Distribution of Stroke based on Gender")
plt.show()
```

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.DataFrame({'sex': ['male', 'male', 'female', 'female', 'male', 'female'],
                   'stroke': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No']})
stroke_counts = df.groupby(['sex', 'stroke']).size()
stroke_counts_stroke = stroke_counts['male']['Yes'], stroke_counts['female']['Yes']
gender_labels = ['male', 'female']
colors = ['lightblue', 'pink']
plt.figure(figsize=(5,5))
plt.pie(stroke_counts_stroke, labels=gender_labels, colors=colors, autopct='%1.1f%%', startangle=180)
plt.title("Distribution of Stroke based on Gender")
plt.show()
```

```
plt.close()
import matplotlib.pyplot as plt
values = df1['bmi']
plt.hist(values, bins = 'auto', color='purple', edgecolor='black', alpha = 0.7)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.title('BMI Distribution')
plt.show()
```

```
plt.close()
import matplotlib.pyplot as plt
values = df1['avg_glucose_level']
plt.hist(values, bins = 'auto', color='purple', edgecolor='black', alpha = 0.9)
plt.grid(axis='y', alpha=0.2)
plt.xlabel('Average Glucose Levels')
plt.ylabel('Frequency')
plt.title('Average Glucose Levels Distribution')
plt.show()
```

```
df1['bmi'].mean()
```

```
df1['avg_glucose_level'].mean()
```

```
import pandas as pd
import matplotlib.pyplot as plt

stroke_df = df1[df1['stroke'] == 'Yes']
stroke_counts_hypertension = stroke_df['hypertension'].value_counts()
plt.bar(stroke_counts_hypertension.index, stroke_counts_hypertension.values, color=['violet', 'lightgreen'])
plt.xlabel('Hypertension')
plt.ylabel('Number of Stroke Patients')
plt.title('Distribution of Stroke Patients based on Hypertension')
plt.show()
```

```
Count_work = df1['work_type'].value_counts()
Count_work
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.countplot(x='work_type', hue='stroke', data=df1, palette='Set1')
plt.title('Distribution of Stroke Cases by Work Type')
plt.xlabel('Work Type')
plt.ylabel('Count')
plt.show()
```

```
correlation_matrix = pd.crosstab(df1['stroke'], df1['hypertension'], normalize='index')
print(correlation_matrix)
```

```
pip install seaborn
```

```
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('Stroke_data.csv')
correlation_matrix = data.corr()
plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap for Stroke Data')
plt.show()
print("Correlation Coefficients:")
print(correlation_matrix)
```

```
#Distribution of Stroke Patients with smoking history
stroke_smoking_counts = df1[df1['stroke'] == 'Yes'].groupby('smoking_status').size()
labels = stroke_smoking_counts.index
values = stroke_smoking_counts.values

colors = ['lightgreen', 'lightcoral']
plt.figure(figsize=(5, 5))
plt.pie(values, labels=labels, colors=colors, autopct='%1.1f%%', startangle=180)
plt.title("Distribution of Stroke Patients with Smoking History")
plt.show()
```

```
#Chi square test for sex and stroke
Relation= pd.crosstab(df1.sex, df1.stroke)
Relation
```

```
from scipy.stats import chi2_contingency
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for hypertension and stroke
Relation= pd.crosstab(df1.hypertension, df1.stroke)
Relation

c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for heart_disease and stroke
Relation= pd.crosstab(df1.heart_disease, df1.stroke)
Relation
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for ever_married and stroke
Relation= pd.crosstab(df1.ever_married, df1.stroke)
Relation
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for work_type and stroke
Relation= pd.crosstab(df1.work_type, df1.stroke)
Relation
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for bmi and stroke
Relation= pd.crosstab(df1.bmi, df1.stroke)
Relation
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for avg_glucose_level and stroke
Relation= pd.crosstab(df1.avg_glucose_level, df1.stroke)
Relation
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
#Chi square test for smoking status and stroke
Relation= pd.crosstab(df1.smoking_status, df1.stroke)
Relation
```

```
c, p, dof, expected = chi2_contingency(Relation)

print("Chi:",c)
print("P:",p)
print("DoF:",dof)
print("Expected frequency: ", expected)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats


plt.figure(figsize=(12, 5))

# Histogram for 'BMI'
plt.subplot(1, 2, 1)
sns.histplot(df1['bmi'], kde=True)
plt.title('Histogram for BMI')

# Q-Q plot for 'BMI'
plt.subplot(1, 2, 2)
stats.probplot(df1['bmi'], plot=plt)
plt.title('Q-Q Plot for BMI')

plt.tight_layout()
plt.show()
```

```
#Logistic Regression
#Importing all the required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']

# Spliting the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test_log = train_test_split(X, y, test_size=0.2, random_state=42)
logreg_model = LogisticRegression(max_iter=1000,random_state=42)
logreg_model.fit(X_train, y_train)
y_pred_log = logreg_model.predict(X_test)

# Create a confusion matrix with actual and predicted values
conf_matrix = confusion_matrix(y_test_log, y_pred_log)

# Extract TN, FP, FN, TP from the confusion matrix
TN, FP, FN, TP = conf_matrix.ravel()

accuracy = accuracy_score(y_test_log, y_pred_log)
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test_log, y_pred_log))
sensitivity = TP / (TP + FN)
print("Sensitivity:", sensitivity)
specificity = TN / (TN + FP)
print("Specificity:", specificity)
```

```
# Display the confusion matrix as a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
#ROC curve for logistic regression
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training a logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Get predicted probabilities
y_probs = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Calculate AUC
roc_auc = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='-', label='Random')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Logistic Regression')
plt.legend()
plt.show()
```

```
pip install catboost
```

```
import catboost
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a CatBoost training pool
train_pool = Pool(X_train, label=y_train)
model = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, loss_function='Logloss',eval_metric='Accuracy', random_seed=42,verbose=False)
model.fit(train_pool)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))

# Displaying the confusion matrix as a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Oranges')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
#ROC curve for catboost
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from catboost import CatBoostClassifier
from sklearn.metrics import roc_curve, roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
train_pool = Pool(X_train, label=y_train)
model = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, loss_function='Logloss',eval_metric='Accuracy', random_seed=42,verbose=False)
model.fit(train_pool)
y_probs = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Calculate AUC
roc_auc = roc_auc_score(y_test, y_probs)

# Plot ROC curve
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for CatBoost Classifier')
plt.legend()
plt.show()
```

```python
#K nearest algorithm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a k-NN classifier
k_value = 3
model = KNeighborsClassifier(n_neighbors=k_value)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))

# Displaying the confusion matrix as a heatmap
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```python
#ROC curve for knearest algorithm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a k-NN classifier
k_value = 3
model = KNeighborsClassifier(n_neighbors=k_value)
model.fit(X_train, y_train)

# Get decision scores instead of probabilities
y_scores = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

# Calculate AUC
roc_auc = roc_auc_score(y_test, y_scores)

# Plot ROC curve
plt.plot(fpr, tpr, color='purple', lw=2, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for k-NN Classifier')
plt.legend()
plt.show()
```

```
#Random Forest
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model performance
accuracy = accuracy_score(y_test, predictions)
print(f'Accuracy: {accuracy}')

# Additional evaluation metrics (classification report)
print('\nClassification Report:\n', classification_report(y_test, predictions))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

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

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

# Fit the model to the training data
model.fit(X_train, y_train)

# Get predicted probabilities for the positive class (class 1)
y_probs = model.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
#plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier

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

# Specifying the features and target variable
X = df2[['sex','age','hypertension','heart_disease','avg_glucose_level','bmi','smoking_status']]
y = df2['stroke']

# Set up k-fold cross-validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation and calculate AUC for each fold
cv_results = cross_val_score(model, X, y, cv=kfold, scoring='roc_auc')

# Print the AUC for each fold
for i, auc in enumerate(cv_results, 1):
    print(f'Fold {i}: AUC = {auc:.4f}')

# Print the mean AUC across all folds
print(f'Mean AUC: {cv_results.mean():.4f}')
```

```python
#ROC curve for 3 models
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import roc_curve, roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train models
# Model 1: Logistic Regression
model_lr = LogisticRegression(max_iter=1000,random_state=42)
model_lr.fit(X_train, y_train)

#Model 2:CatBoost
model_catboost = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, loss_function='Logloss',eval_metric='Accuracy', random_seed=42,verbose=F
model_catboost.fit(X_train, y_train)

# Model 3: k-Nearest Neighbors
model_knn = KNeighborsClassifier(n_neighbors=3)
model_knn.fit(X_train, y_train)

#Model 4: Random Forest
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Get predicted probabilities for the positive class
y_probs_lr = model_lr.predict_proba(X_test)[:, 1]
y_probs_catboost = model_catboost.predict_proba(X_test)[:, 1]
y_probs_knn = model_knn.predict_proba(X_test)[:, 1]
y_probs_rf = model.predict_proba(X_test)[:, 1]

# Calculate ROC curves and AUCs
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_probs_lr)
fpr_catboost, tpr_catboost, _ = roc_curve(y_test, y_probs_catboost)
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_probs_knn)
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_probs_rf)

roc_auc_lr = roc_auc_score(y_test, y_probs_lr)
roc_auc_catboost = roc_auc_score(y_test, y_probs_catboost)
roc_auc_knn = roc_auc_score(y_test, y_probs_knn)
roc_auc_rf = roc_auc_score(y_test, y_probs_rf)

# Plot ROC curves
plt.plot(fpr_lr, tpr_lr, color='blue', lw=2, label=f'Logistic Regression (AUC = {roc_auc_lr:.2f})')
plt.plot(fpr_catboost, tpr_catboost, color='green', lw=2, label=f'CatBoost (AUC = {roc_auc_catboost:.2f})')
plt.plot(fpr_knn, tpr_knn, color='purple', lw=2, label=f'KNN (AUC = {roc_auc_knn:.2f})')
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curves for Logistic Regression, CatBoost, and KNN')
plt.legend()
plt.show()
```