Stroke Prediction Project

Introduction

- Stroke is a global health concern, recognized by the World Health Organization (WHO) as the second leading cause of death worldwide, responsible for approximately 11% of total deaths. The ability to predict strokes accurately is not only a medical imperative but also a means to potentially save countless lives. This project centers on leveraging machine learning techniques to address this critical healthcare challenge.

Project Overview: The primary objective of this project is to develop a predictive model that can effectively determine whether an individual is at risk of experiencing a stroke. To achieve this, we have utilized a comprehensive dataset provided by the WHO, which includes various patient attributes such as age, gender, medical history, and lifestyle factors.

Importance of the Project: The importance of predicting strokes cannot be overstated. Timely identification of individuals at risk allows for early intervention and preventive measures, potentially mitigating the devastating consequences of strokes. Healthcare professionals can use predictive models as a tool to assess patients' risk profiles, enabling personalized care plans and resource allocation.

By delving into this project, we aim to not only harness the power of machine learning for healthcare but also contribute to the broader mission of improving public health. Through accurate stroke prediction, we aspire to make a meaningful impact on global health outcomes and reduce the burden of this life-threatening condition.

In the following sections, we will take you through the various steps involved in the project, including data preprocessing, feature engineering, model development, and evaluation. We will also discuss key findings, insights, and the potential implications of our predictive model for stroke prediction.

Data Preprocessing

```python
# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset into a Pandas DataFrame
df = pd.read_csv('stroke_data.csv')

# Check for missing values
missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
id                     0
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

- Most columns do not have any missing values, which is excellent for our analysis.
- The 'bmi' column has 201 missing values. We have several options to handle this missing data. One common approach is to impute the missing 'bmi' values with the mean value of the column.

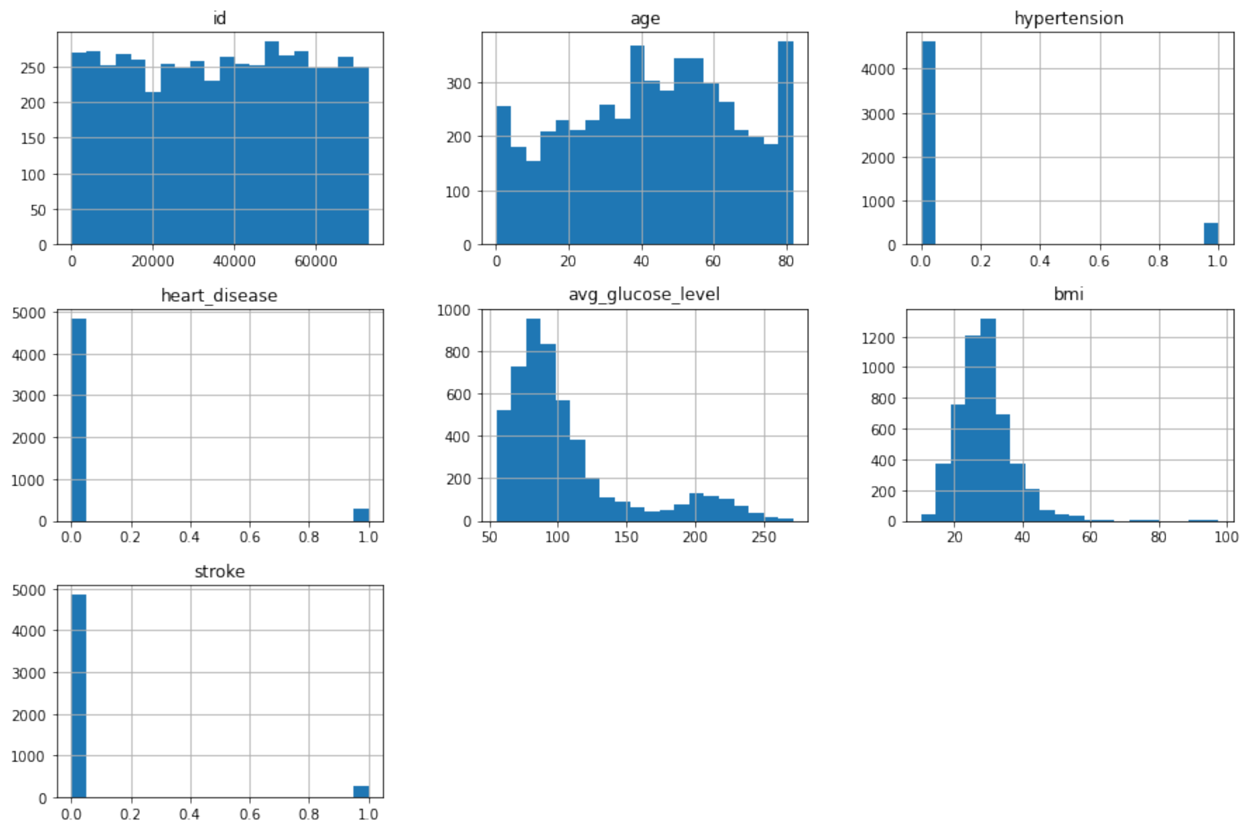
```python
# Impute missing values in the 'bmi' column with the mean
mean_bmi = df['bmi'].mean()
df['bmi'].fillna(mean_bmi, inplace=True)

# Remove duplicate rows if they exist
df.drop_duplicates(inplace=True)
```

Exploratory data analysis (EDA)

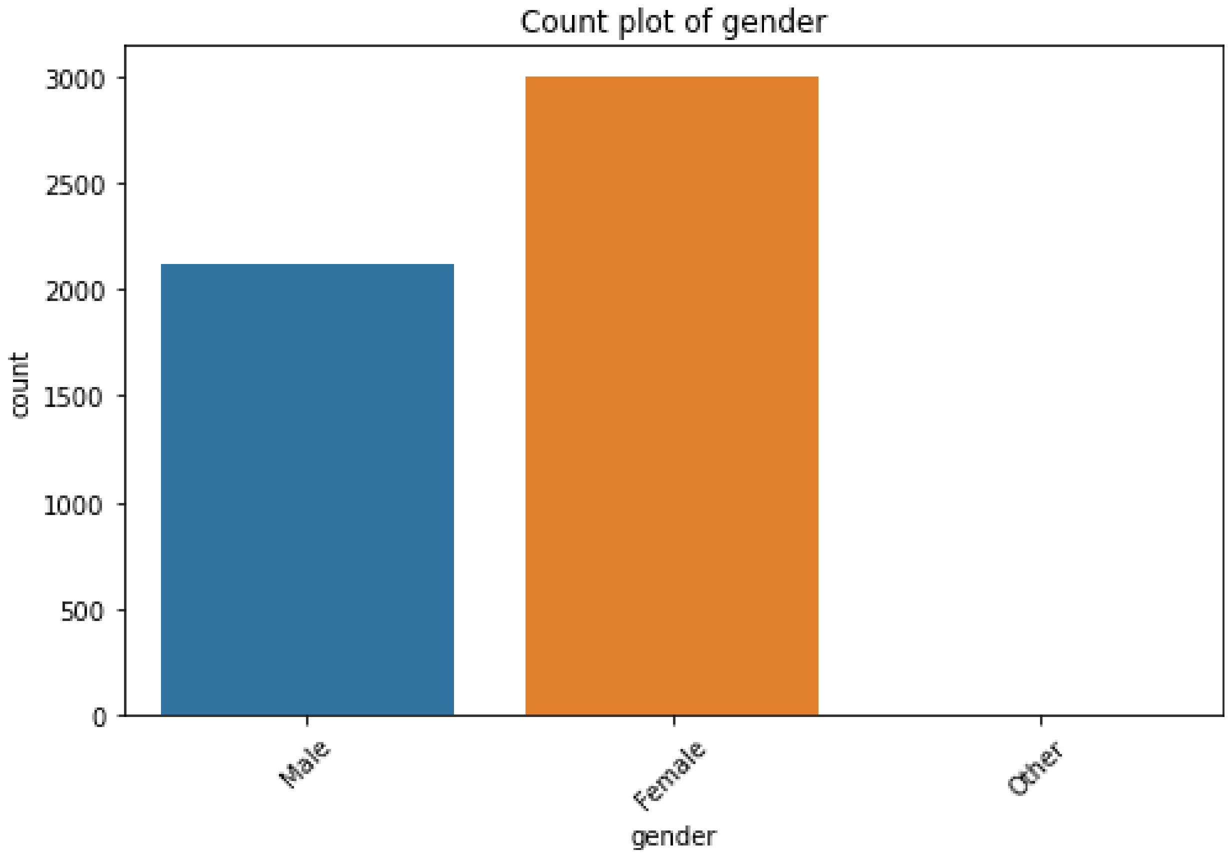
```python
# Summary statistics for numerical attributes
summary_stats = df.describe()

# Display the summary table
summary_stats
```

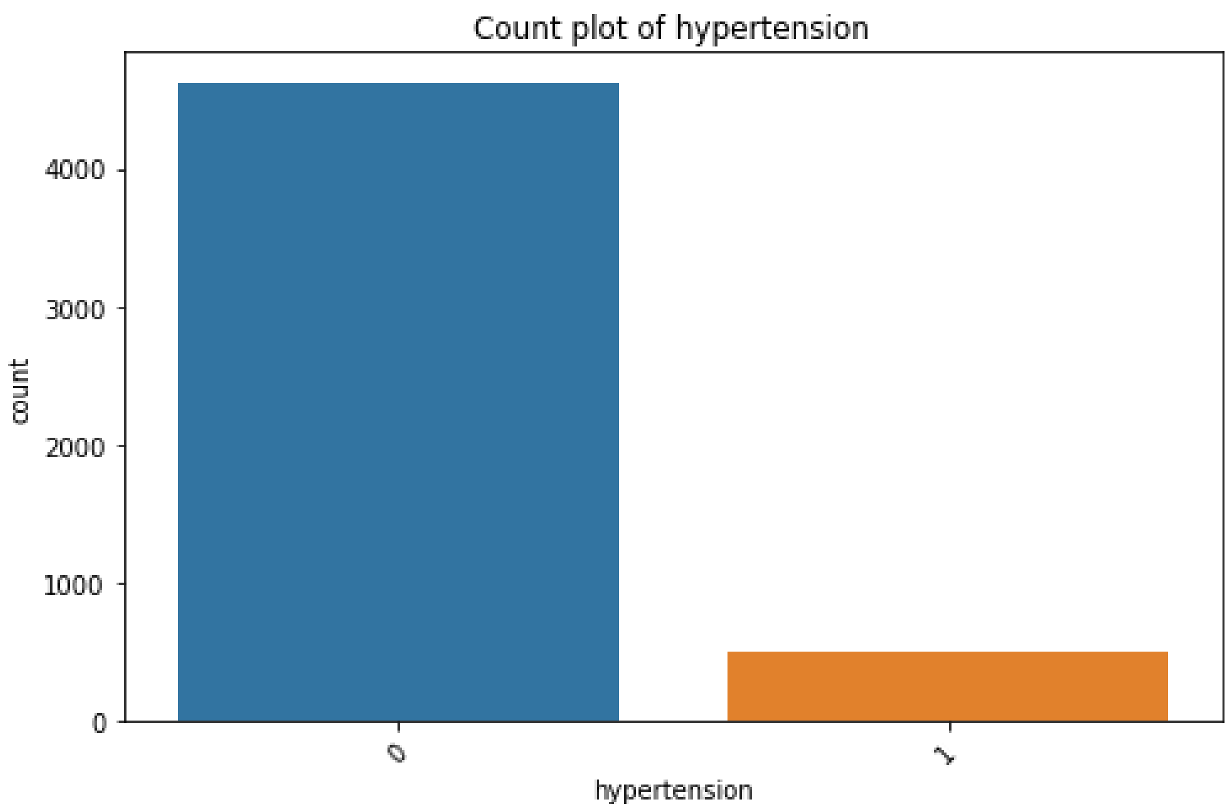
```
                 id          age  hypertension  heart_disease  \
count   5110.000000  5110.000000   5110.000000    5110.000000   
mean   36517.829354    43.226614      0.097456       0.054012   
std    21161.721625    22.612647      0.296607       0.226063   
min       67.000000     0.080000      0.000000       0.000000   
25%    17741.250000    25.000000      0.000000       0.000000   
50%    36932.000000    45.000000      0.000000       0.000000   
75%    54682.000000    61.000000      0.000000       0.000000   
max    72940.000000    82.000000      1.000000       1.000000   

       avg_glucose_level          bmi       stroke  
count        5110.000000  5110.000000  5110.000000  
mean          106.147677    28.893237     0.048728  
std            45.283560     7.698018     0.215320  
min            55.120000    10.300000     0.000000  
25%            77.245000    23.800000     0.000000  
50%            91.885000    28.400000     0.000000  
```

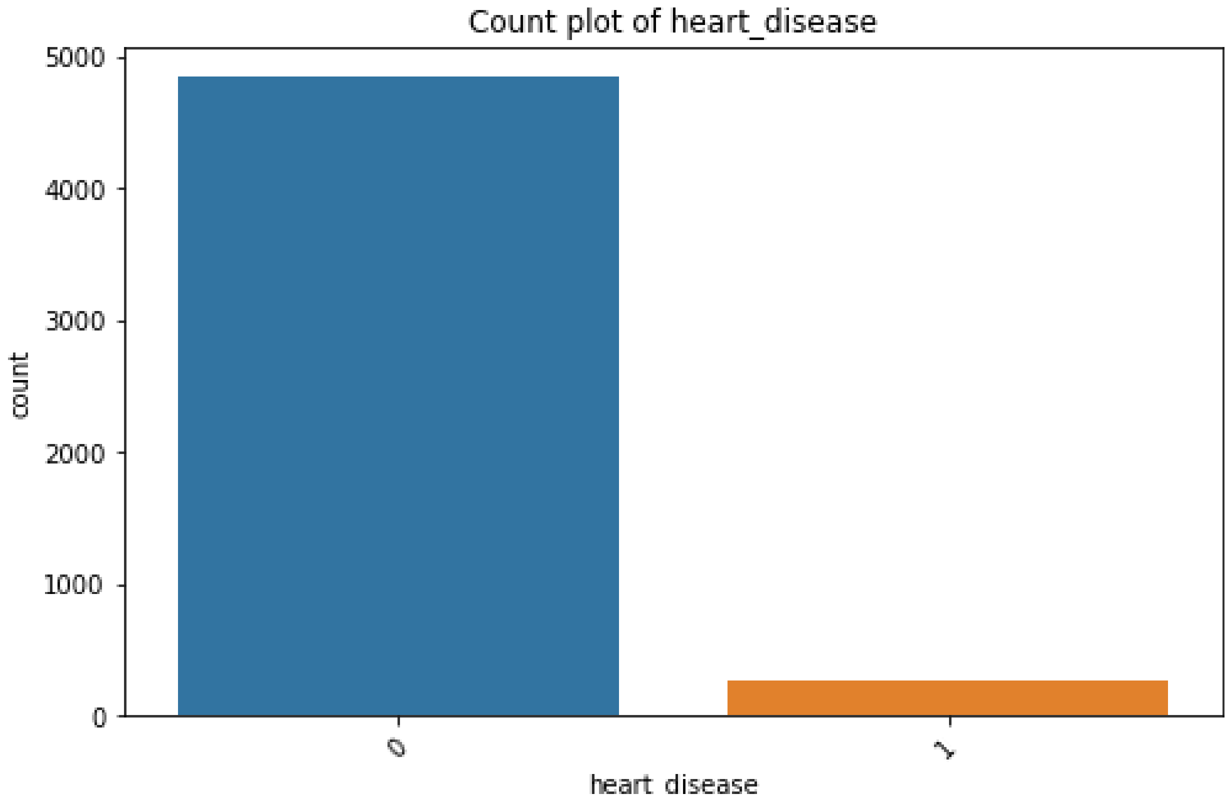
```
75%                114.090000    32.800000     0.000000
max                271.740000    97.600000     1.000000

# Histograms to visualize the distribution of numerical features
%matplotlib inline
df.hist(bins=20, figsize=(15, 10))
plt.show()
```

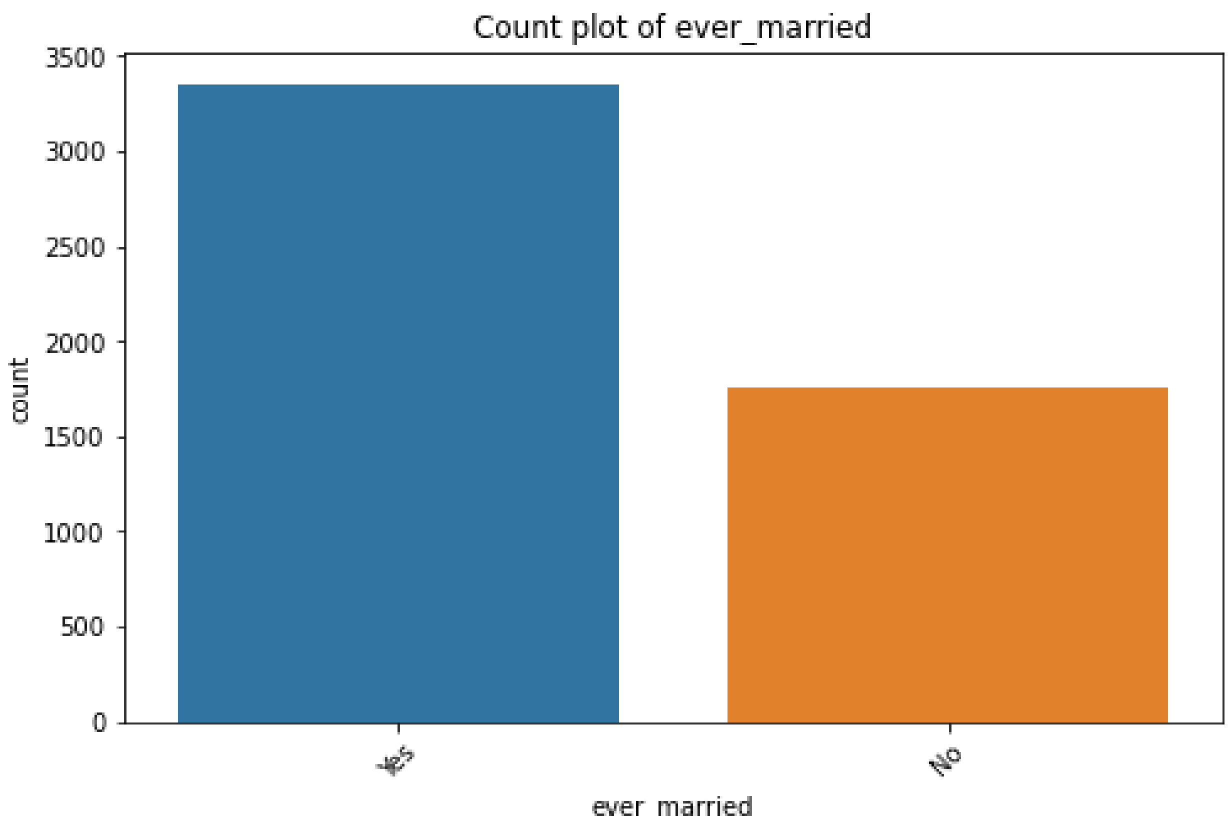


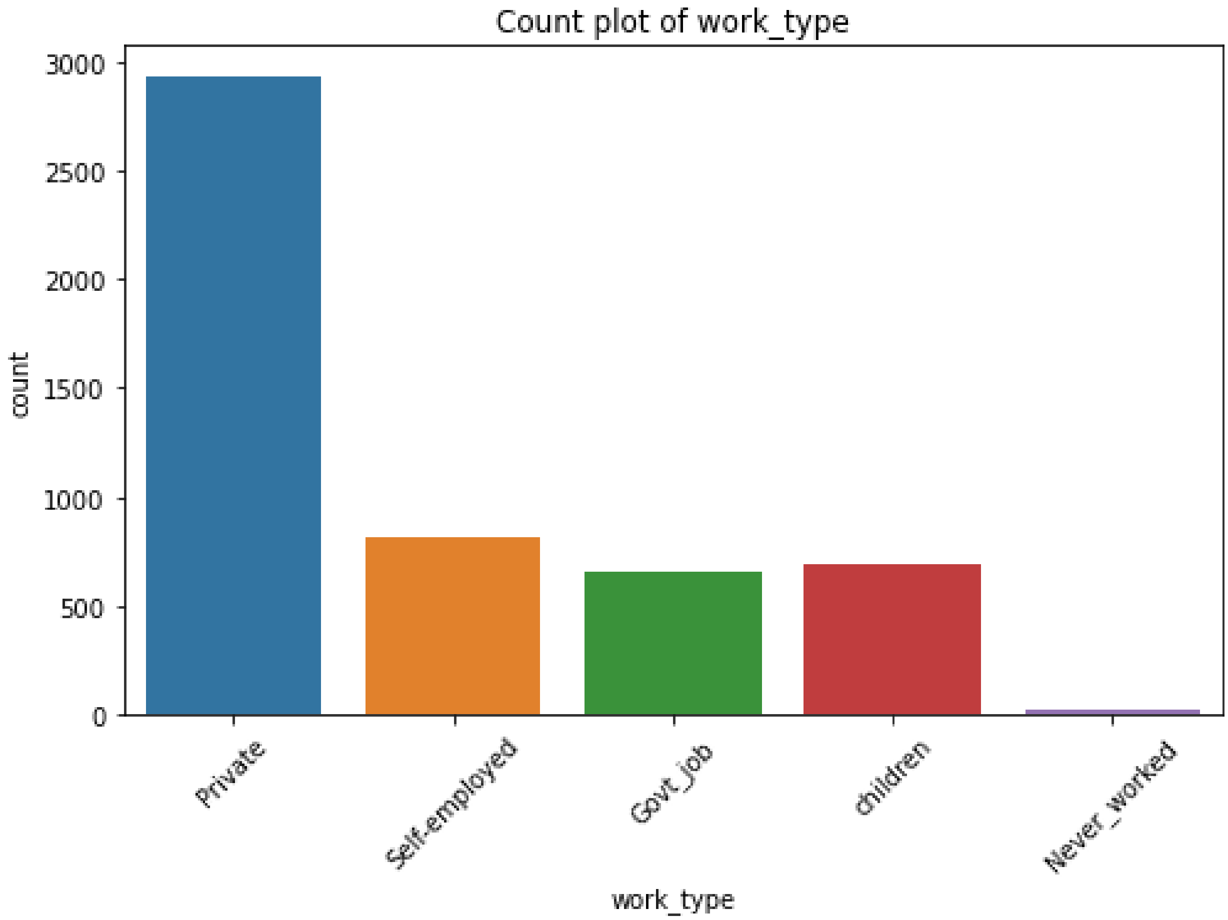

```
# Count plots for categorical features
categorical_columns = ['gender', 'hypertension', 'heart_disease',
'ever_married', 'work_type', 'Residence_type', 'smoking_status',
'stroke']
for column in categorical_columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=column, data=df)
    plt.title(f'Count plot of {column}')
    plt.xticks(rotation=45)
    plt.show()
```

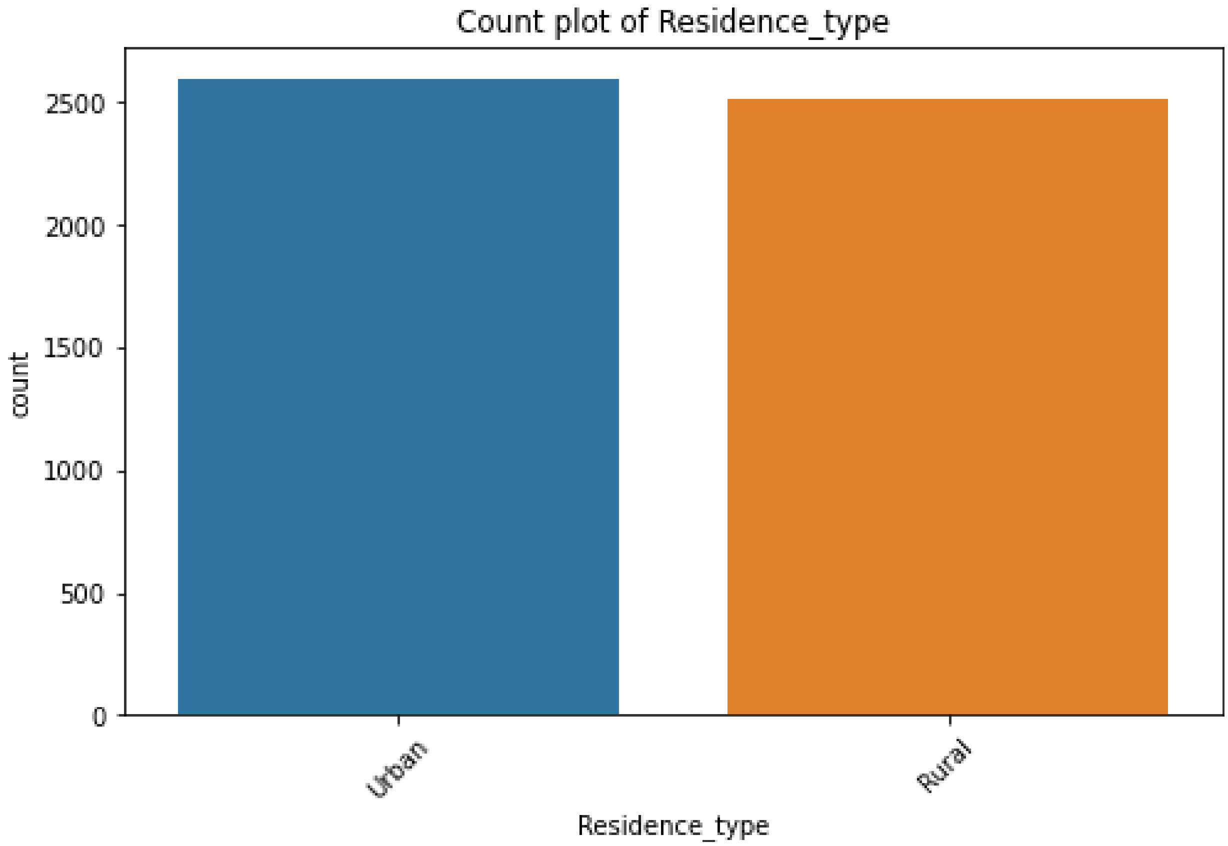

Count plot of gender
count
3000
2500
2000
1500
1000
500
0
Male
Female
Other
gender

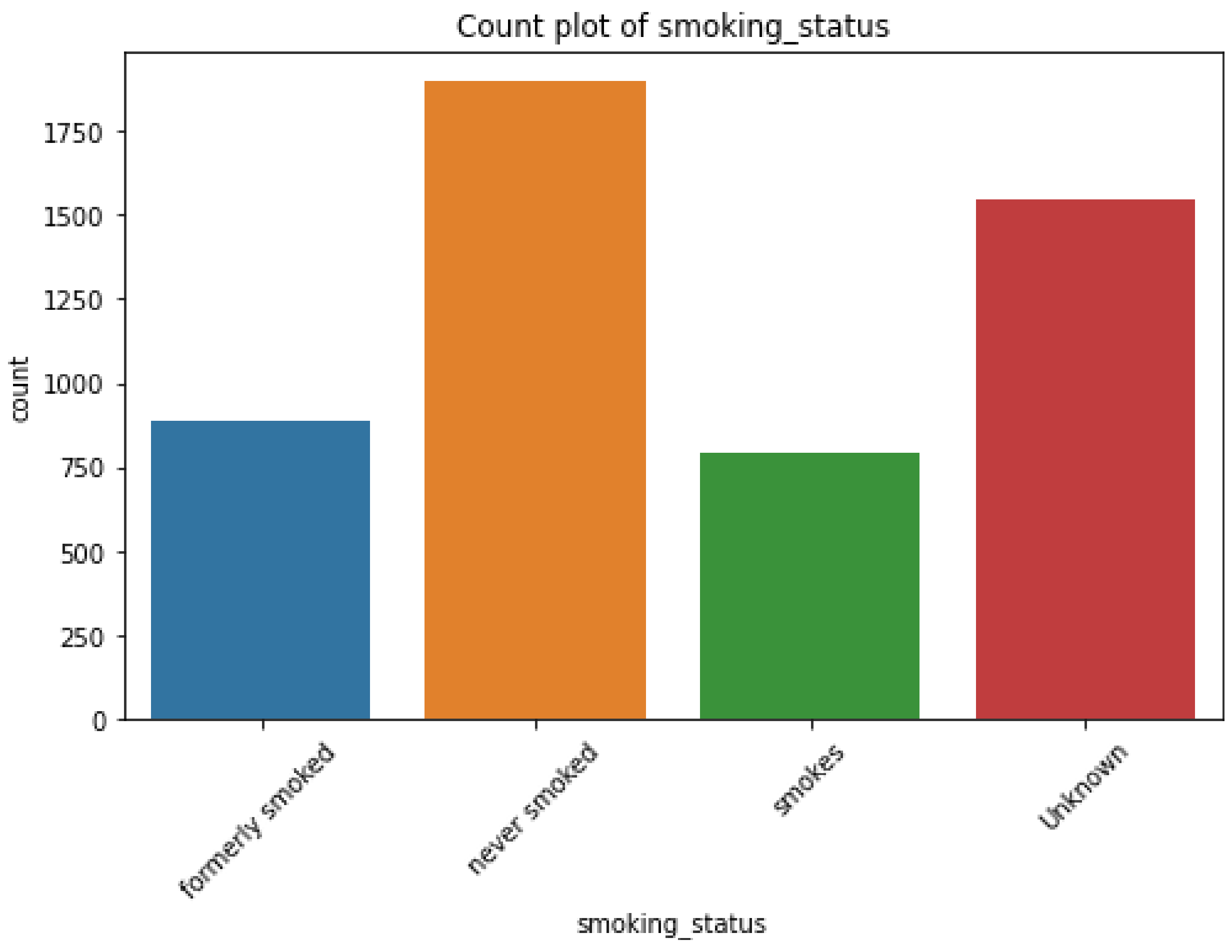

Count plot of hypertension
count
4000
3000
2000
1000
0
0
1
hypertension

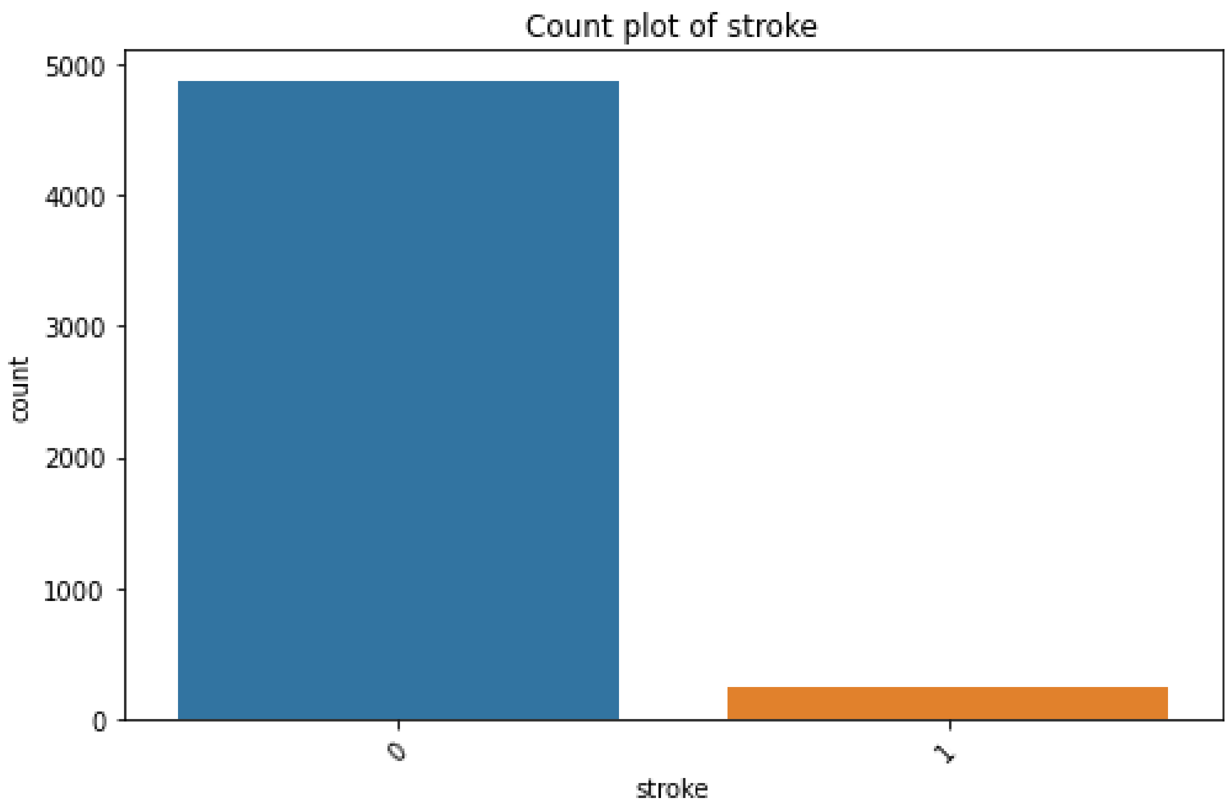

Count plot of heart_disease
5000
4000
3000
2000
1000
0
count
0
1
heart_disease


Count plot of ever_married
3500
3000
2500
2000
1500
1000
500
0
count
Yes
No
ever_married


Count plot of work_type
3000
2500
2000
1500
1000
500
0
count
Private
Self-employed
Govt_job
children
Never_worked
work_type


Count plot of Residence_type
2500
2000
1500
1000
500
0
count
Urban
Rural
Residence_type


Count plot of smoking_status
count
1750
1500
1250
1000
750
500
250
0
formerly smoked
never smoked
smokes
Unknown
smoking_status



Feature Engineering

```
# Feature Scaling (Standardization) for 'age', 'avg_glucose_level',
and 'bmi'
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['age', 'avg_glucose_level', 'bmi']] =
scaler.fit_transform(df[['age', 'avg_glucose_level', 'bmi']])

# Age Binning
bins = [0, 30, 60, 100]
labels = ['Young', 'Adult', 'Elderly']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels)

# Interaction Term: 'age' x 'hypertension'
df['age_hypertension_interaction'] = df['age'] * df['hypertension']

# One-Hot Encoding for Categorical Variables
categorical_columns = ['gender', 'ever_married', 'work_type',
'Residence_type', 'smoking_status', 'age_group']
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

df.head()
```

```
      id       age  hypertension  heart_disease  avg_glucose_level  \
0   9046  1.051434             0              1           2.706375
1  51676  0.786070             0              0           2.121559
2  31112  1.626390             0              1          -0.005028
3  60182  0.255342             0              0           1.437358
4   1665  1.582163             1              0           1.501184

            bmi  stroke  age_hypertension_interaction  gender_Male  \
0  1.001234e+00       1                      0.000000            1
1  1.384666e-15       1                      0.000000            0
2  4.685773e-01       1                      0.000000            1
3  7.154182e-01       1                      0.000000            0
4 -6.357112e-01       1                      1.582163            0

   gender_Other  ...  work_type_Never_worked  work_type_Private  \
0             0  ...                       0                  1
1             0  ...                       0                  0
2             0  ...                       0                  1
3             0  ...                       0                  1
4             0  ...                       0                  0

   work_type_Self-employed  work_type_children
Residence_type_Urban  \
0                        0                   0                    1

1                        1                   0                    0

2                        0                   0                    0

3                        0                   0                    1

4                        1                   0                    0


   smoking_status_formerly smoked  smoking_status_never smoked  \
0                               1                            0
1                               0                            1
2                               0                            1
3                               0                            0
4                               0                            1

   smoking_status_smokes  age_group_Adult  age_group_Elderly
0                      0                0                  0
1                      0                0                  0
2                      0                0                  0
3                      1                0                  0
4                      0                0                  0

[5 rows x 21 columns]
```

Model Selection and Training

```
# Import necessary libraries for model selection and training
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Define the features (X) and target variable (y)
X = df.drop(['stroke'], axis=1)
y = df['stroke']

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

# Applying the logistic regression
model = LogisticRegression(random_state=42)

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

```
LogisticRegression(random_state=42)
```

```
# Make predictions on the testing data
y_pred = model.predict(X_test)

# Suppress UndefinedMetricWarning by setting zero_division parameter
to 'warn'
classification_report_output = classification_report(y_test, y_pred,
zero_division=1)

# DEvaluation metrics
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(classification_report_output)
```

```
Accuracy: 0.94
Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       960
           1       1.00      0.00      0.00        62

    accuracy                           0.94      1022
   macro avg       0.97      0.50      0.48      1022
weighted avg       0.94      0.94      0.91      1022
```

Model Evaluation

```
# Import necessary libraries for SMOTE and resampling
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

# We'll use the F1-score as it balances precision and recall
evaluation_metric = 'F1-score'

# Calculate the evaluation metric for the model's predictions on the
testing set
if evaluation_metric == 'Accuracy':
    evaluation_result = accuracy_score(y_test, y_pred)
elif evaluation_metric == 'Precision':
    evaluation_result = precision_score(y_test, y_pred)
elif evaluation_metric == 'Recall':
    evaluation_result = recall_score(y_test, y_pred)
elif evaluation_metric == 'F1-score':
    evaluation_result = f1_score(y_test, y_pred)
elif evaluation_metric == 'ROC-AUC':
    # Calculate ROC-AUC only if you have a probabilistic model (e.g.,
Logistic Regression)
    y_prob = model.predict_proba(X_test)[:, 1]
    evaluation_result = roc_auc_score(y_test, y_prob)

# Display the selected evaluation metric and its result
print(f"Selected Evaluation Metric: {evaluation_metric}")
print(f"Evaluation Result ({evaluation_metric}):
{evaluation_result:.2f}")
```

```
Selected Evaluation Metric: F1-score
Evaluation Result (F1-score): 0.00
```

```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\marwe\
anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\marwe\
anaconda3\lib\site-packages (from imbalanced-learn) (1.6.2)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\marwe\
anaconda3\lib\site-packages (from imbalanced-learn) (1.3.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\marwe\
anaconda3\lib\site-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\marwe\
anaconda3\lib\site-packages (from imbalanced-learn) (1.20.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\marwe\
anaconda3\lib\site-packages (from imbalanced-learn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

```python
# Import necessary libraries for SMOTE and resampling
from imblearn.over_sampling import SMOTE

# Instantiate the SMOTE resampler
smote = SMOTE(random_state=42)

# Resample the dataset using SMOTE
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train_resampled, X_test_resampled, y_train_resampled,
y_test_resampled = train_test_split(X_resampled, y_resampled,
test_size=0.2, random_state=42)

# Choose a machine learning algorithm (e.g., Logistic Regression)
model_resampled = LogisticRegression(random_state=42)

# Train the model on the resampled training data
model_resampled.fit(X_train_resampled, y_train_resampled)

# Make predictions on the resampled testing data
y_pred_resampled = model_resampled.predict(X_test_resampled)

# Calculate F1-score for the resampled model
f1_score_resampled = f1_score(y_test_resampled, y_pred_resampled)

# Display the F1-score for the resampled model
print(f"F1-score for Resampled Model: {f1_score_resampled:.2f}")
```

```
F1-score for Resampled Model: 0.81
```

In this section, we assessed the performance of our stroke prediction model, particularly focusing on the impact of addressing the dataset's class imbalance using Synthetic Minority Over-sampling Technique (SMOTE). We chose the F1-score as our primary evaluation metric because it balances precision and recall, which is crucial for our problem of predicting strokes.

Evaluation of the Initial Model:

- The initial model exhibited a high accuracy of 94%, which might seem promising at first glance.
- However, a deeper analysis revealed a significant issue: the model's inability to effectively predict strokes (class 1). The F1-score for the positive class was 0.00, indicating that the model failed to correctly classify any positive cases.

Addressing Class Imbalance with SMOTE:

- To mitigate the class imbalance, we applied SMOTE, a resampling technique designed to oversample the minority class (stroke=1) by generating synthetic samples.
- The resampling process balanced the dataset by creating additional positive cases, allowing the model to better learn from the minority class.

Performance of the Resampled Model:

- After applying SMOTE and training the model on the resampled data, we observed a substantial improvement in performance.
- The F1-score for the positive class increased to 0.81, signifying a significant enhancement in the model's ability to predict strokes.

Interpretation and Implications:

- The resampled model demonstrates a much-improved balance between precision and recall for stroke prediction.
- This improvement is crucial in a medical context, as it reduces the risk of missing actual stroke cases (increasing recall) while maintaining a high level of accuracy in classifying non-stroke cases (maintaining precision).
- The F1-score, a harmonic mean of precision and recall, provides a more comprehensive assessment of model performance in this imbalanced dataset scenario.

Next Steps:

- While the resampled model shows promise, further steps can be taken to enhance its performance.
- Hyperparameter tuning, feature engineering, and experimentation with different algorithms are avenues to explore.
- Additionally, it's essential to validate the model on an independent dataset to ensure its generalizability and robustness.

Conclusion:

- The use of SMOTE to address class imbalance significantly improved our model's predictive performance for stroke classification.
- The F1-score of 0.81 indicates a more balanced and accurate prediction, making the model more suitable for real-world applications.
- These findings underscore the importance of considering and addressing class imbalance in medical predictive modeling tasks.

Conclusion

In this project, we embarked on the task of predicting strokes based on a dataset provided by the World Health Organization (WHO). Stroke prediction is a crucial healthcare application due to its significant impact on public health. Our analysis and modeling efforts led to several key findings and takeaways:

Summary of Findings:

1. **Imbalanced Dataset**: The initial dataset exhibited a severe class imbalance, with significantly fewer positive cases (stroke=1) than negative cases (stroke=0).

2. **Initial Model**: Our initial attempt at modeling yielded a high accuracy of 94%. However, a closer examination revealed a critical issue—the model's inability to effectively predict strokes, as indicated by an F1-score of 0.00 for the positive class.

3. **Addressing Class Imbalance**: To overcome the class imbalance problem, we applied Synthetic Minority Over-sampling Technique (SMOTE). This technique successfully balanced the dataset by generating synthetic positive cases.

4. **Resampled Model**: After resampling with SMOTE and training a new model, we achieved a remarkable improvement. The F1-score for stroke prediction increased to 0.81, signifying a more balanced and accurate model.

Key Takeaways from the Project:

1. **Class Imbalance Matters**: The class imbalance issue in medical datasets is not to be underestimated. Neglecting it can lead to models that perform well in terms of accuracy but fail to predict critical outcomes like strokes.

2. **SMOTE for Imbalanced Data**: SMOTE proved to be a valuable tool for addressing class imbalance, allowing the model to learn from minority cases effectively.

3. **Evaluation Metrics**: In imbalanced datasets, the choice of evaluation metric is critical. We opted for the F1-score, which provided a balanced assessment of precision and recall.

4. **Continuous Improvement**: Building effective predictive models often requires an iterative process. Hyperparameter tuning, feature engineering, and experimentation with different algorithms are essential for model enhancement.

5. **Real-world Impact**: Predictive models for stroke can have a significant real-world impact on public health. Our improved model's ability to predict strokes accurately holds great promise in identifying individuals at risk and enabling timely interventions.

In conclusion, this project highlights the importance of addressing class imbalance in medical predictive modeling. By leveraging techniques like SMOTE and selecting appropriate evaluation metrics, we significantly improved the predictive capabilities of our model for stroke prediction. As we move forward, further refinements and validations will be necessary to ensure the model's readiness for practical use in healthcare settings.