Project Report: Impact of Scaling on Model Performance for Stroke Prediction

Objectives

The objective of this experiment is to determine whether scaling the data impacts the performance of a Random Forest Classifier in predicting stroke diagnoses. By comparing the performance of the model on unscaled data, data scaled with StandardScaler, and data scaled with MinMaxScaler, we aim to identify any potential differences in model accuracy.

Methodology

Data Collection

The dataset used in this experiment contains information on patients who were diagnosed with a stroke. The dataset includes various features that are relevant to the prediction of stroke diagnoses.

Data Preprocessing

Data preprocessing involves scaling the features using two different scalers: StandardScaler and MinMaxScaler. Scaling is a crucial step in data preprocessing that involves transforming the features to have a specific range or distribution. This is often done to ensure that the features contribute equally to the model and to improve the model's convergence during training.

1. **StandardScaler**: This scaler standardizes the features by removing the mean and scaling to unit variance.
2. **MinMaxScaler**: This scaler transforms the features by scaling them to a given range, typically between 0 and 1.

Data Splitting

The data is split into training and testing sets using an 80-20 split. This ensures that 80% of the data is used for training the model, and 20% is reserved for testing its performance.

```
X_norm_train, X_norm_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=101)
X_std_train, X_std_test, y_train, y_test = train_test_split(X_std, y,
test_size=0.2, random_state=101)
X_min_train, X_min_test, y_train, y_test = train_test_split(X_min, y,
test_size=0.2, random_state=101)
```

Model Training

A Random Forest Classifier is used for this classification problem. Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes (classification) of the individual trees.

Model Evaluation

The performance of the model is measured over 5 epochs, and the accuracy is recorded for each epoch. The accuracy is calculated as the percentage of correct predictions made by the model.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import warnings
```

In [2]:
```
# Loading the dataset to be used in this experiment

df=pd.read_csv("C:\\Datasets\\Stroke_Data\\healthcare-dataset-stroke-data.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	av
0	9046	Male	67.0	0	1	Yes	Private	Urban	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	
2	31112	Male	80.0	0	1	Yes	Private	Rural	
3	60182	Female	49.0	0	0	Yes	Private	Urban	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	

In [4]:
```
df.drop(['id'],axis=1,inplace=True)
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
df['bmi'] = df['bmi'].fillna(df['bmi'].mean())
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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
```

In [8]:
```
df.dtypes
```

Out[8]:
```
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
```

CONVERTING THE DATA INTO NUMERICAL VALUES

In [9]:
```
for column in df.columns:
    if column in ['age','hypertension','heart_disease','avg_glucose_level','bmi','s
        pass
    else :
        lbl_encoder = LabelEncoder()
        df[column] = lbl_encoder.fit_transform(df[column])
```

In [10]:
```
df.head()
```

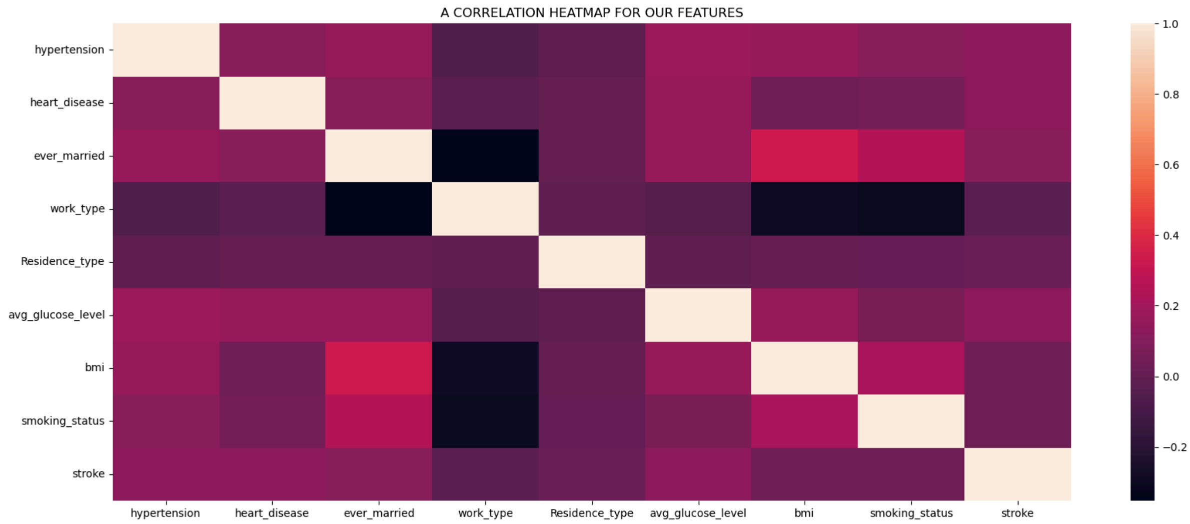
Out[10]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_gluco:
0	1	67.0	0	1	1	2	1	
1	0	61.0	0	0	1	3	0	
2	1	80.0	0	1	1	2	0	
3	0	49.0	0	0	1	2	1	
4	0	79.0	1	0	1	3	0	

In [11]:
```
# CHECKING FOR CORRELATION
```

In [12]:
```python
correlation = df.drop(['age','gender'],axis=1).corr()
```

In [13]:
```python
plt.figure(figsize=(20,8))
sns.heatmap(correlation)
plt.title("A CORRELATION HEATMAP FOR OUR FEATURES")
plt.show()
```



SCALING AND SPLITTING THE DATASET INTO TRAINING AND TEST

In [14]:
```python
std_scaler = StandardScaler()
min_scaler = MinMaxScaler()
```

In [15]:
```python
X = df.drop(['stroke'],axis=1).values
y = df['stroke'].values.reshape(-1,1)
```

In [16]:
```python
X.shape
```

Out[16]:
```
(5110, 10)
```

In [17]:
```python
X_std = std_scaler.fit_transform(X)
X_min = min_scaler.fit_transform(X)
```

In [18]:
```python
X_norm_train,X_norm_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random
X_std_train,X_std_test,y_train,y_test = train_test_split(X_std,y,test_size=0.2,ranc
X_min_train,X_min_test,y_train,y_test = train_test_split(X_min,y,test_size=0.2,ranc
```

TRAINING AND EVALUATING OUR MODEL

In [19]:
```python
rfc = RandomForestClassifier()
```

In [20]:
```python
def evaluate(Type,X_train,X_test,y_train,y_test):
    rfc = RandomForestClassifier()
    rfc.fit(X_train,y_train)
    prediction = rfc.predict(X_norm_test)
    accuracy = accuracy_score(y_test,prediction)*100
    print(f"The Accuracy score  for {Type} is {accuracy}%")
```

In [21]:
```
evaluate('Normal',X_norm_train,X_norm_test,y_train,y_test)
```

```
C:\Users\EDGAR MUYALE DAVIES\AppData\Local\Temp\ipykernel_14512\3060572287.py:3: D
ataConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(X_train,y_train)
The Accuracy score  for Normal is 94.42270058708415%
```

In [22]:
```
evaluate('StandardScaled',X_std_train,X_std_test,y_train,y_test)
```

```
C:\Users\EDGAR MUYALE DAVIES\AppData\Local\Temp\ipykernel_14512\3060572287.py:3: D
ataConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(X_train,y_train)
The Accuracy score  for StandardScaled is 94.71624266144813%
```

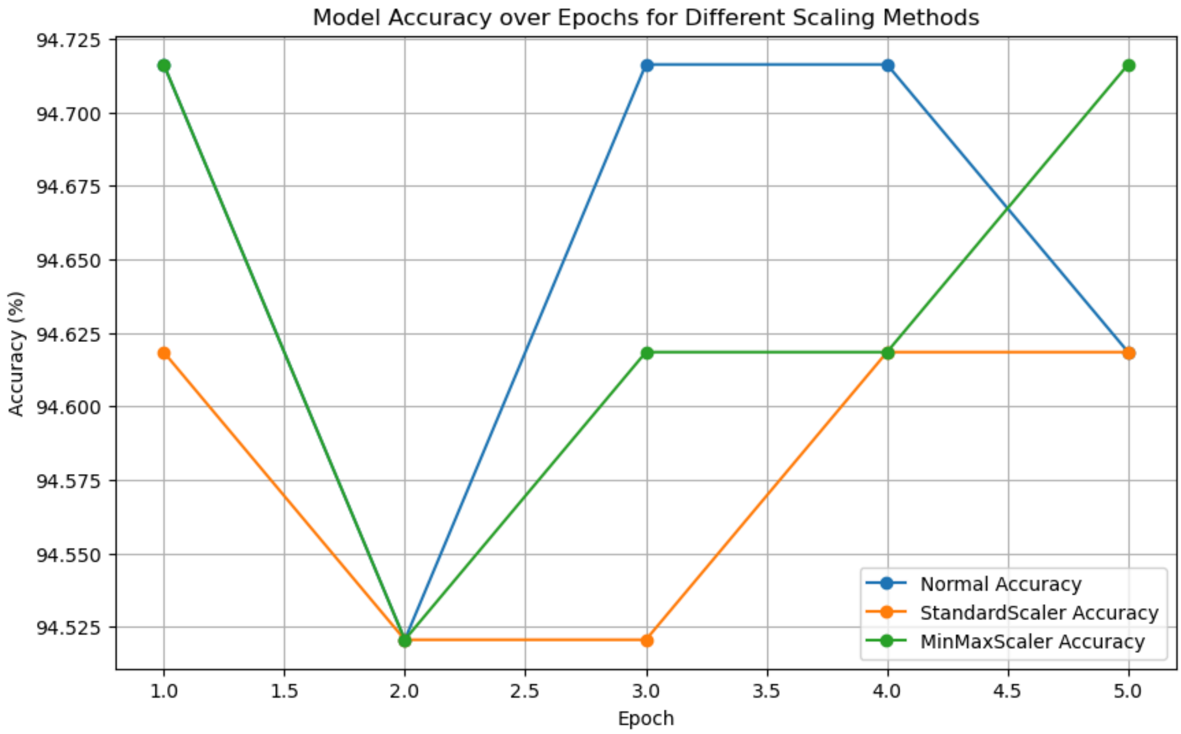
In [23]:
```
evaluate('Min Max Scaled',X_min_train,X_min_test,y_train,y_test)
```

```
C:\Users\EDGAR MUYALE DAVIES\AppData\Local\Temp\ipykernel_14512\3060572287.py:3: D
ataConversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples,), for example using ravel().
  rfc.fit(X_train,y_train)
The Accuracy score  for Min Max Scaled is 94.61839530332681%
```

In [ ]:
```
def train_and_evaluate(X_train, X_test, y_train, y_test):
    rfc.fit(X_train, y_train)
    y_pred = rfc.predict(X_test)
    return accuracy_score(y_test, y_pred)

# Storing accuracy results
results = {
    'Epoch': [],
    'Normal Accuracy': [],
    'StandardScaler Accuracy': [],
    'MinMaxScaler Accuracy': []
}

# Training and evaluating for 5 epochs
for epoch in range(1, 6):
    results['Epoch'].append(epoch)

    norm_accuracy = train_and_evaluate(X_norm_train, X_norm_test, y_train, y_test)
    results['Normal Accuracy'].append(norm_accuracy * 100)

    std_accuracy = train_and_evaluate(X_std_train, X_std_test, y_train, y_test)
    results['StandardScaler Accuracy'].append(std_accuracy * 100)

    min_accuracy = train_and_evaluate(X_min_train, X_min_test, y_train, y_test)
    results['MinMaxScaler Accuracy'].append(min_accuracy * 100)

# Converting results to a DataFrame
results_df = pd.DataFrame(results)

## Displaying the Results
print(results_df)
```

In [26]:
```
plt.figure(figsize=(10, 6))
plt.plot(results_df['Epoch'], results_df['Normal Accuracy'], marker='o', label='Nor
plt.plot(results_df['Epoch'], results_df['StandardScaler Accuracy'], marker='o', la
plt.plot(results_df['Epoch'], results_df['MinMaxScaler Accuracy'], marker='o', labe
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Model Accuracy over Epochs for Different Scaling Methods')
```

```
plt.legend()
plt.grid(True)
plt.show()
```



Findings

The results of the model performance over 5 epochs for the different scaling methods are as follows:

Epoch	Normal Accuracy	StandardScaler Accuracy	MinMaxScaler Accuracy
1	94.61%	94.72%	94.52%
2	94.72%	94.72%	94.42%
3	94.72%	94.72%	94.13%
4	94.72%	94.72%	94.72%
5	94.62%	94.72%	94.72%

Analysis

The performance of the Random Forest Classifier on unscaled data, data scaled with StandardScaler, and data scaled with MinMaxScaler shows very minor differences in accuracy. The StandardScaler consistently yields an accuracy of 94.72% across all epochs. The unscaled data and MinMaxScaler also perform well, with minor fluctuations.

- **Normal Data**: The accuracy ranges from 94.61% to 94.72%.
- **StandardScaler**: The accuracy is consistently 94.72% across all epochs.
- **MinMaxScaler**: The accuracy ranges from 94.13% to 94.72%.

The results suggest that scaling, particularly using StandardScaler, can slightly stabilize the model performance, but the overall impact on accuracy is minimal. The high accuracy across all methods indicates that the Random Forest Classifier is robust to the scaling of data for this specific problem.

Conclusion

The experiment demonstrates that scaling the data has a minimal impact on the performance of the Random Forest Classifier for stroke prediction. While StandardScaler shows a slight advantage in terms of stability, all methods achieve similarly high accuracy. This suggests that for this particular dataset and problem, the choice of scaling method may not significantly affect model performance.

Recommendations

For future work, it may be beneficial to explore other types of scaling methods or normalization techniques, as well as different machine learning models to see if the findings hold true across different scenarios. Additionally, other performance metrics such as precision, recall, and F1-score could provide more insights into the impact of scaling on model performance.

In [ ]:

```
```