```python
# prompt: import data from keggle

# Install the Kaggle API client.
!pip install kaggle

# Upload your Kaggle API key (kaggle.json).
# Go to https://www.kaggle.com/<username>/account, click "Create New
API Token", and upload the downloaded kaggle.json file.
from google.colab import files
files.upload()

# Move the kaggle.json file to the correct location.
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/

# Change the permissions of the file.
!chmod 600 ~/.kaggle/kaggle.json

# Download the dataset. Replace 'username/dataset-name' with the
actual username and dataset name.
!kaggle datasets download -d 'shantanudhakadd/diabetes-dataset-for-
beginners'

# Unzip the downloaded file (if necessary).
!unzip doctors-handwritten-prescription-bd-dataset.zip # replace with
actual filename if different

# Now you can load the data using pandas (or other libraries)
import pandas as pd

# Example: Load a CSV file
# df = pd.read_csv("your_data_file.csv")  # Replace with your actual
data file name
# print(df.head())
```

```
Requirement already satisfied: kaggle in
/usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in
/usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2024.8.30)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from kaggle) (4.66.6)
Requirement already satisfied: python-slugify in
/usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in
```

```
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in
/usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle)
(1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle)
(3.4.0)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)

<IPython.core.display.HTML object>

Saving kaggle.json to kaggle.json
Dataset URL: https://www.kaggle.com/datasets/shantanudhakadd/diabetes-
dataset-for-beginners
License(s): other
Downloading diabetes-dataset-for-beginners.zip to /content
  0% 0.00/8.91k [00:00<?, ?B/s]
100% 8.91k/8.91k [00:00<00:00, 16.7MB/s]
unzip:  cannot find or open doctors-handwritten-prescription-bd-
dataset.zip, doctors-handwritten-prescription-bd-dataset.zip.zip or
doctors-handwritten-prescription-bd-dataset.zip.ZIP.
```

Read Data

```
data_file = '/content/diabetes-dataset-for-beginners.zip'
data = pd.read_csv(data_file)

# Print column names to verify
print(data.columns)

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
```

```
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
data.head()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 768,\n \"fields\": [\n   {\n   \"column\": \"Pregnancies\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3,\n   \"min\": 0,\n   \"max\": 17,\n   \"num_unique_values\": 17,\n   \"samples\": [\n   6,\n   1,\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Glucose\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 31,\n   \"min\": 0,\n   \"max\": 199,\n   \"num_unique_values\": 136,\n   \"samples\": [\n   151,\n   101,\n   112\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"BloodPressure\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 19,\n   \"min\": 0,\n   \"max\": 122,\n   \"num_unique_values\": 47,\n   \"samples\": [\n   86,\n   46,\n   85\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"SkinThickness\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 15,\n   \"min\": 0,\n   \"max\": 99,\n   \"num_unique_values\": 51,\n   \"samples\": [\n   7,\n   12,\n   48\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Insulin\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 115,\n   \"min\": 0,\n   \"max\": 846,\n   \"num_unique_values\": 186,\n   \"samples\": [\n   52,\n   41,\n   183\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"BMI\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 7.884160320375441,\n   \"min\": 0.0,\n   \"max\": 67.1,\n   \"num_unique_values\": 248,\n   \"samples\": [\n   19.9,\n   31.0,\n   38.1\n   ],\n

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DiabetesPedigreeFunction\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.33132859501277484,\n        \"min\": 0.078,\n        \"max\": 2.42,\
n        \"num_unique_values\": 517,\n        \"samples\": [\n
1.731,\n          0.426,\n          0.138\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11,\n        \"min\": 21,\n
\"max\": 81,\n        \"num_unique_values\": 52,\n        \"samples\":
[\n          60,\n          47,\n          72\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Outcome\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"data"}
```

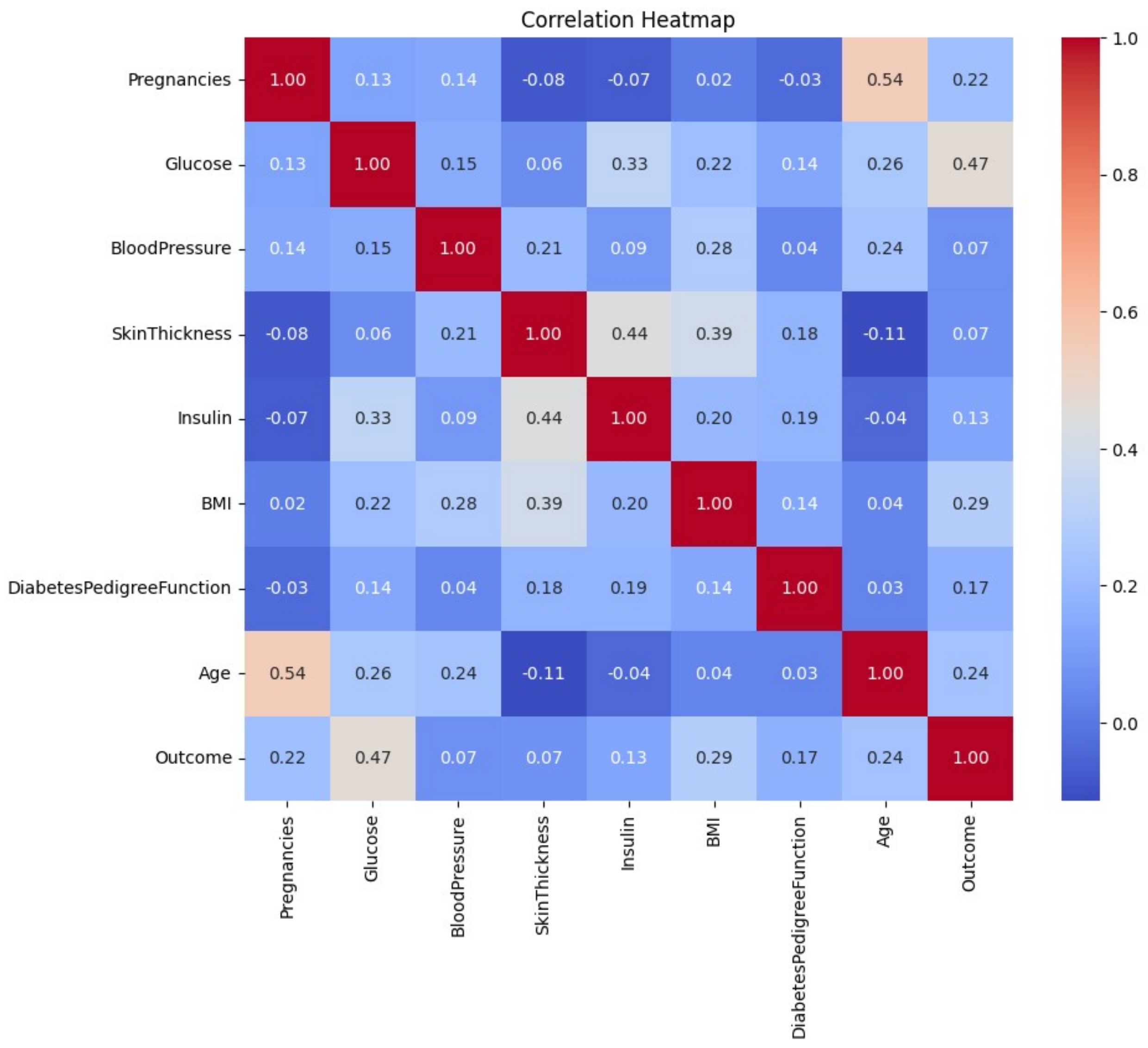
```
data.describe()
```

```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8,\n  \"fields\": [\
n    {\n      \"column\": \"Pregnancies\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 269.85223453356366,\n
\"min\": 0.0,\n        \"max\": 768.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          3.8450520833333335,\n
3.0,\n          768.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Glucose\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 243.73802348295857,\n        \"min\": 0.0,\n
\"max\": 768.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          120.89453125,\n          117.0,\n
768.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BloodPressure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 252.85250535810619,\n        \"min\":
0.0,\n        \"max\": 768.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          69.10546875,\n          72.0,\n
768.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SkinThickness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 263.7684730531098,\n        \"min\":
0.0,\n        \"max\": 768.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          768.0,\n          20.536458333333332,\n
32.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Insulin\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 350.26059167945886,\n        \"min\": 0.0,\n
\"max\": 846.0,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          768.0,\n          79.79947916666667,\n
127.25\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BMI\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 262.05117817552093,\n        \"min\": 0.0,\n        \"max\":
768.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
31.992578124999998,\n          32.0,\n          768.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DiabetesPedigreeFunction\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
271.3005221658502,\n        \"min\": 0.078,\n        \"max\": 768.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
0.47187630208333325,\n          0.3725,\n          768.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 260.1941178528413,\n
\"min\": 11.76023154067868,\n        \"max\": 768.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
33.240885416666664,\n          29.0,\n          768.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Outcome\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
271.3865920388932,\n        \"min\": 0.0,\n        \"max\": 768.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.3489583333333333,\n          1.0,\n          0.4769513772427971\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

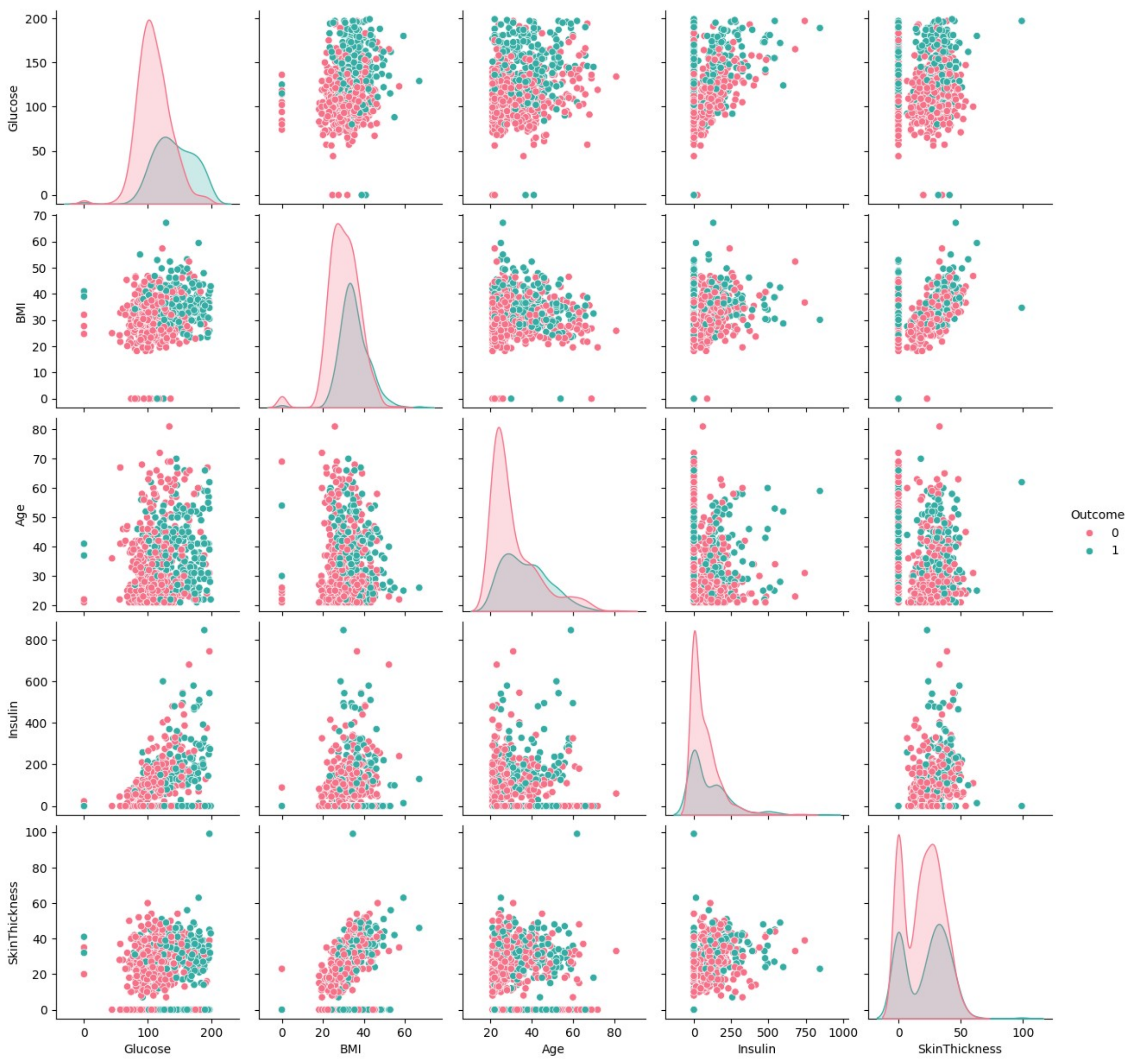
```
data.isnull().sum()
```

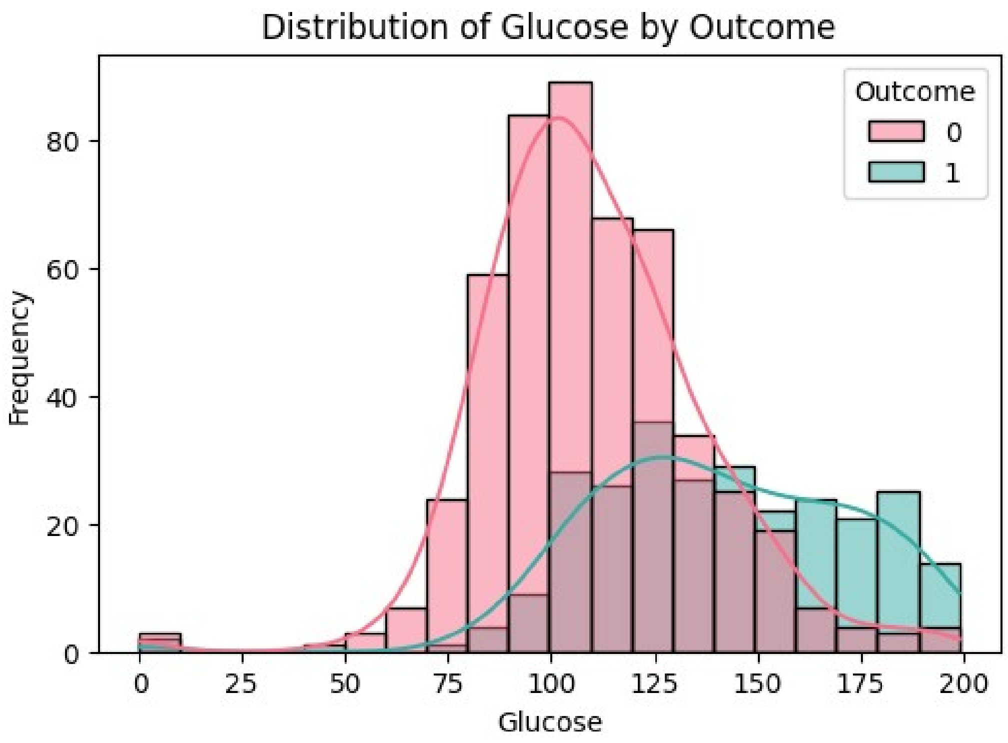
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

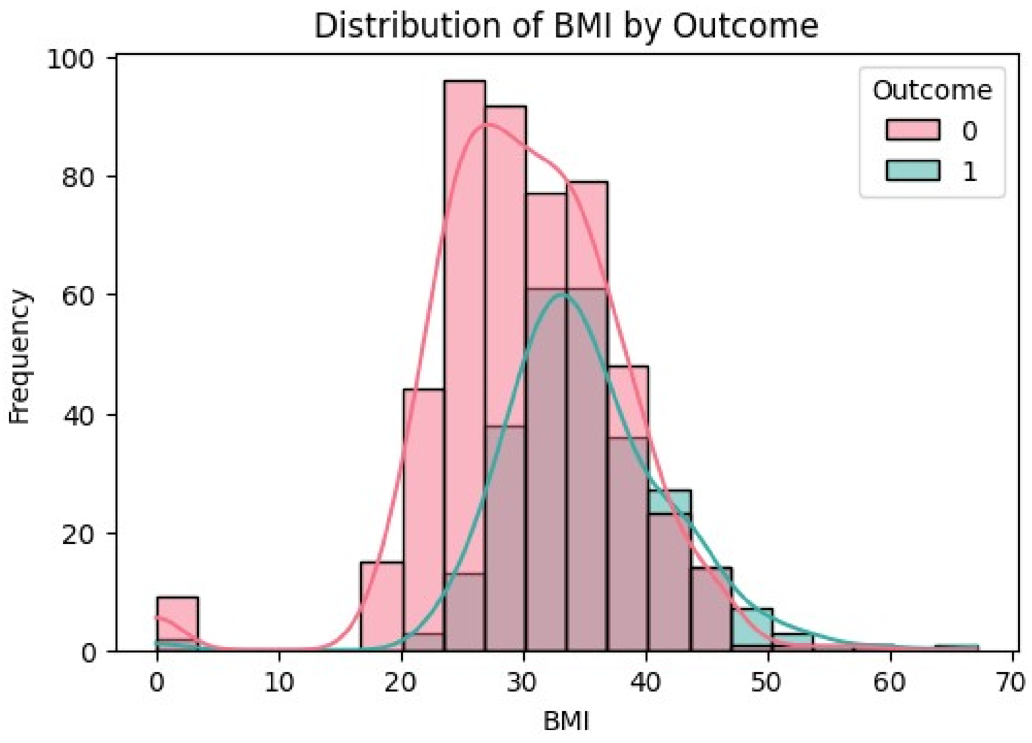
```
# Import Libraries
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Correlation Heatmap
plt.figure(figsize=(10, 8))
correlation_matrix = data.corr()  # Compute correlation matrix
sns.heatmap(
    correlation_matrix,
    annot=True,  # Show correlation coefficients
    fmt=".2f",   # Format float values
    cmap="coolwarm",
    cbar=True
)
plt.title("Correlation Heatmap")
plt.show()

# 2. Pair Plot (Relationships Between Features)
# Only plot selected features for simplicity
selected_features = ['Glucose', 'BMI', 'Age', 'Insulin',
'SkinThickness', 'Outcome']
sns.pairplot(data[selected_features], hue='Outcome', diag_kind='kde',
palette='husl')
plt.show()

# 3. Feature Distribution Plot
# Visualize distributions of selected features
for feature in selected_features:
    if feature != 'Outcome':
        plt.figure(figsize=(6, 4))
        sns.histplot(data, x=feature, hue="Outcome", kde=True,
palette="husl", bins=20)
        plt.title(f"Distribution of {feature} by Outcome")
        plt.xlabel(feature)
        plt.ylabel("Frequency")
        plt.show()

# 4. Box Plot (Feature vs Outcome)
# Analyze the distribution of numeric features based on the target
variable
for feature in selected_features:
    if feature != 'Outcome':
        plt.figure(figsize=(6, 4))
        sns.boxplot(data=data, x='Outcome', y=feature, palette="husl")
        plt.title(f"Box Plot of {feature} vs Outcome")
        plt.xlabel("Outcome")
        plt.ylabel(feature)
        plt.show()
```

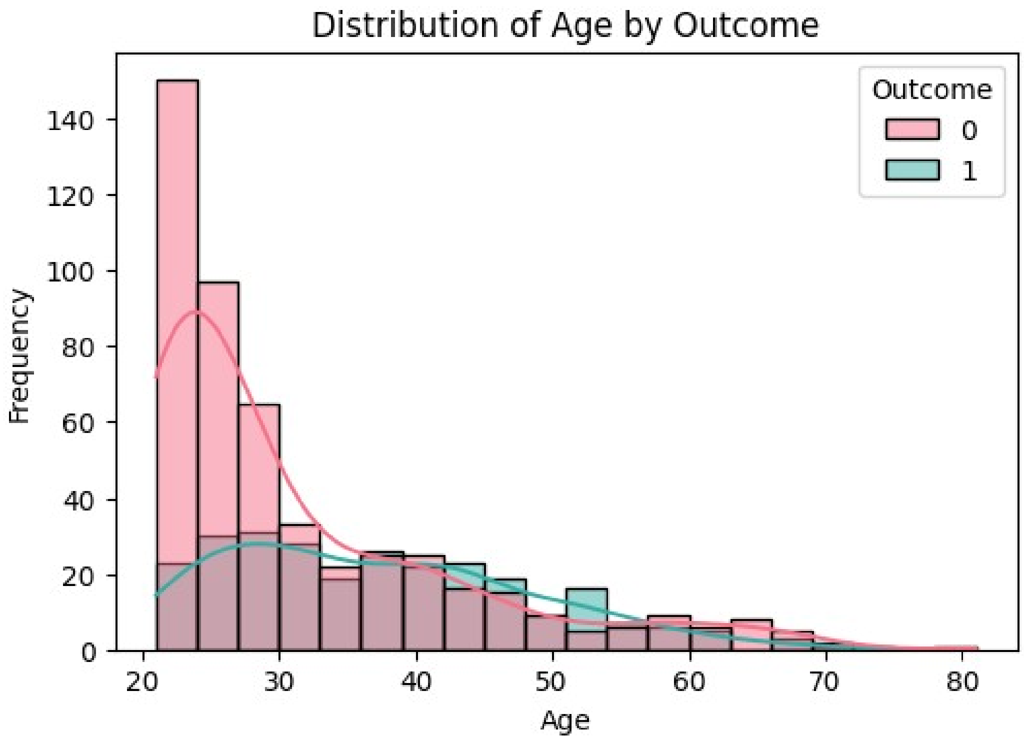


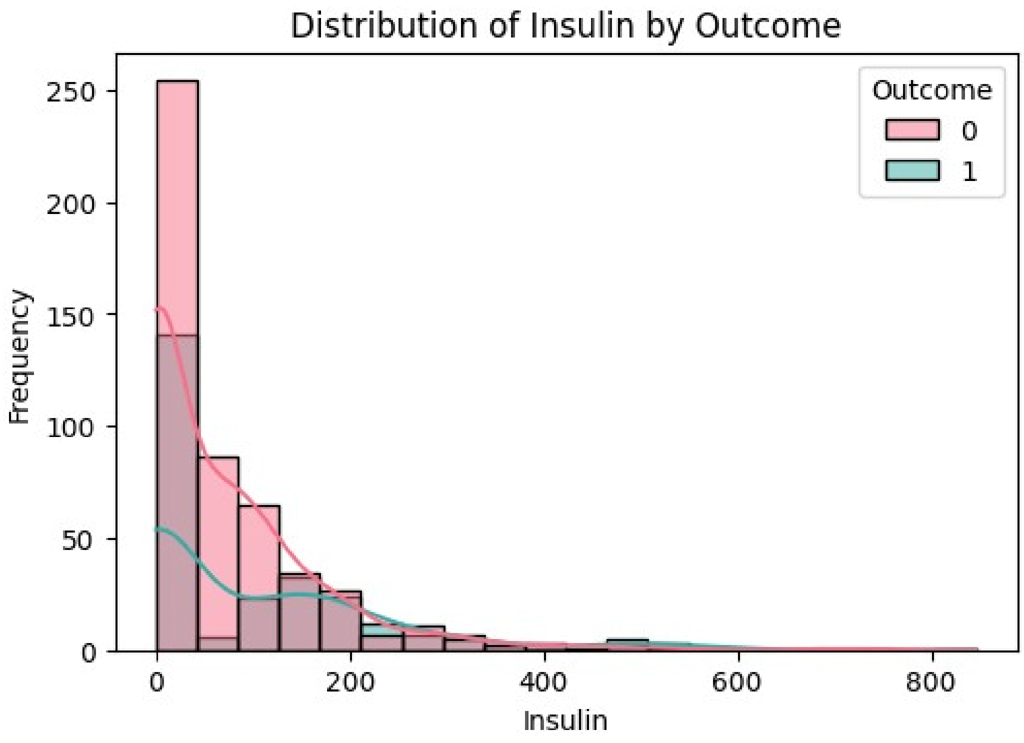
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.00	0.13	0.14	-0.08	-0.07	0.02	-0.03	0.54	0.22
Glucose	0.13	1.00	0.15	0.06	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1.00	0.21	0.09	0.28	0.04	0.24	0.07
SkinThickness	-0.08	0.06	0.21	1.00	0.44	0.39	0.18	-0.11	0.07
Insulin	-0.07	0.33	0.09	0.44	1.00	0.20	0.19	-0.04	0.13
BMI	0.02	0.22	0.28	0.39	0.20	1.00	0.14	0.04	0.29
DiabetesPedigreeFunction	-0.03	0.14	0.04	0.18	0.19	0.14	1.00	0.03	0.17
Age	0.54	0.26	0.24	-0.11	-0.04	0.04	0.03	1.00	0.24
Outcome	0.22	0.47	0.07	0.07	0.13	0.29	0.17	0.24	1.00

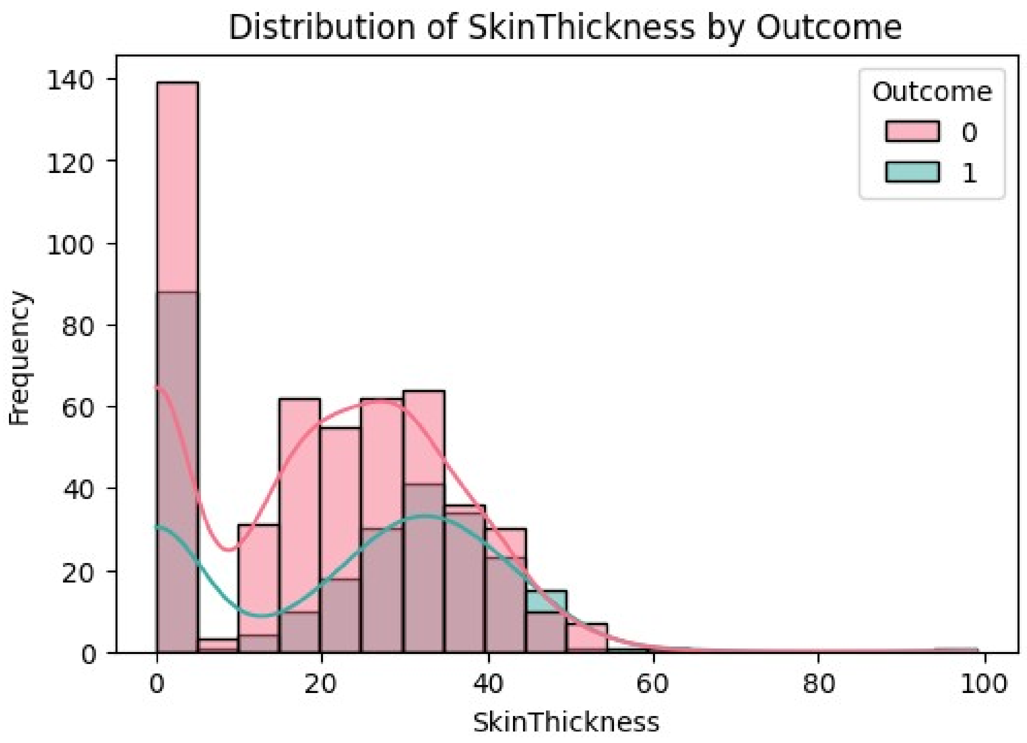

Glucose
BMI
Age
Insulin
SkinThickness
Outcome
0
1


Distribution of Glucose by Outcome
Outcome
0
1
Frequency
0
20
40
60
80
0
25
50
75
100
125
150
175
200
Glucose

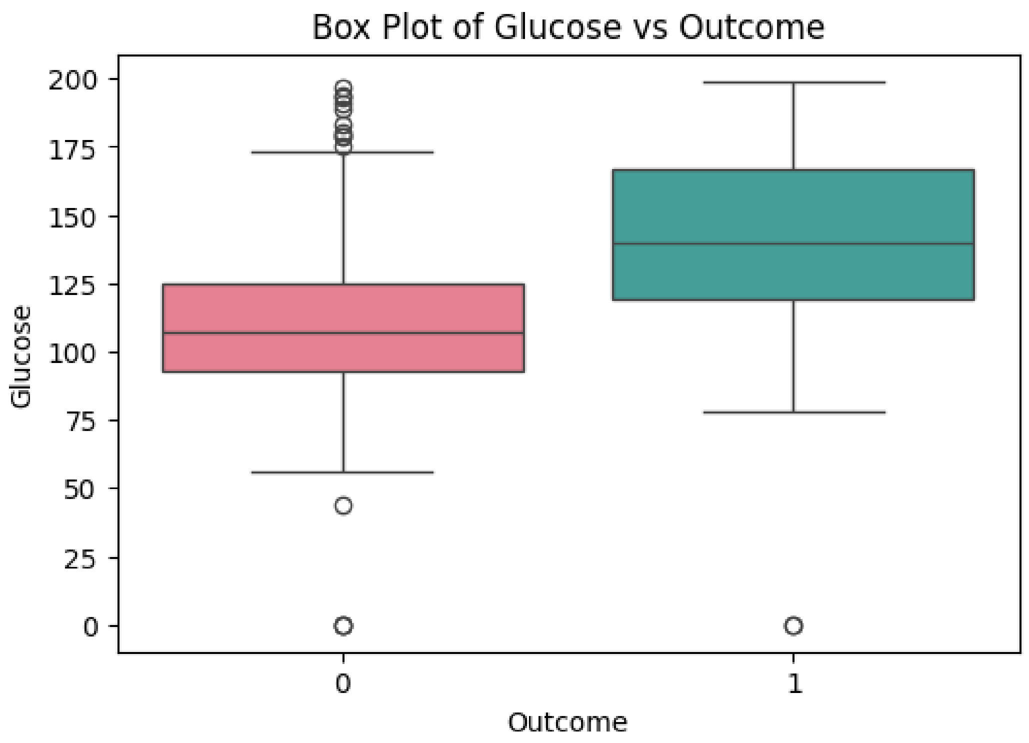

Distribution of BMI by Outcome
Outcome
0
1
Frequency
0
20
40
60
80
100
0
10
20
30
40
50
60
70
BMI


Distribution of Age by Outcome
Outcome
0
1
Frequency
0
20
40
60
80
100
120
140
20
30
40
50
60
70
80
Age

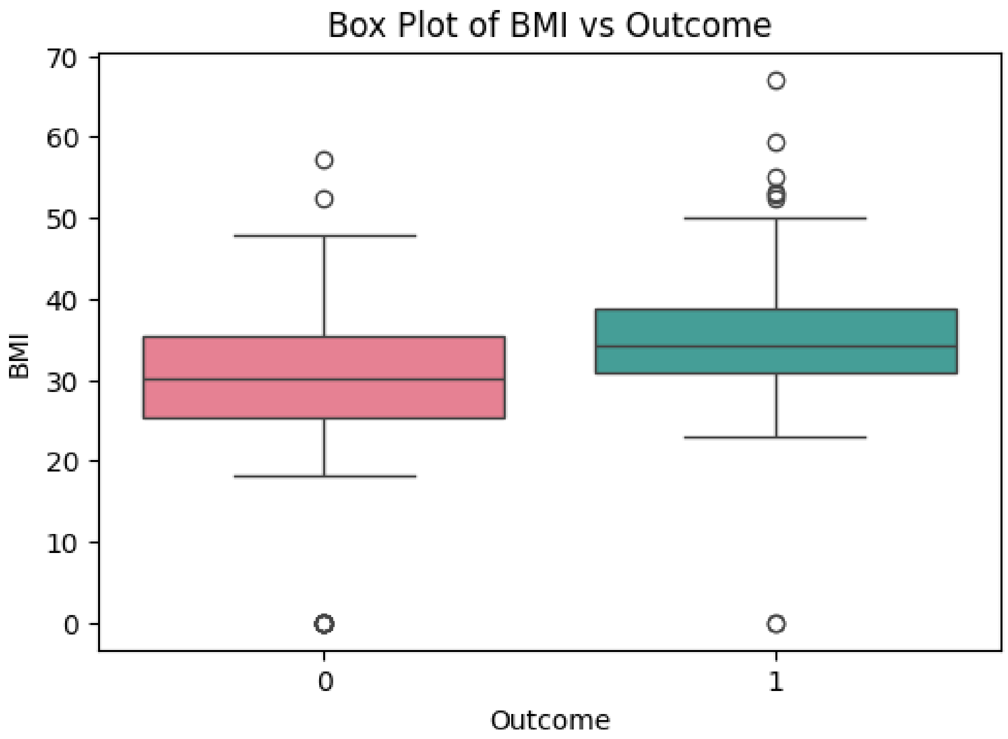

Distribution of Insulin by Outcome
Outcome
0
1
Frequency
0
50
100
150
200
250
0
200
400
600
800
Insulin



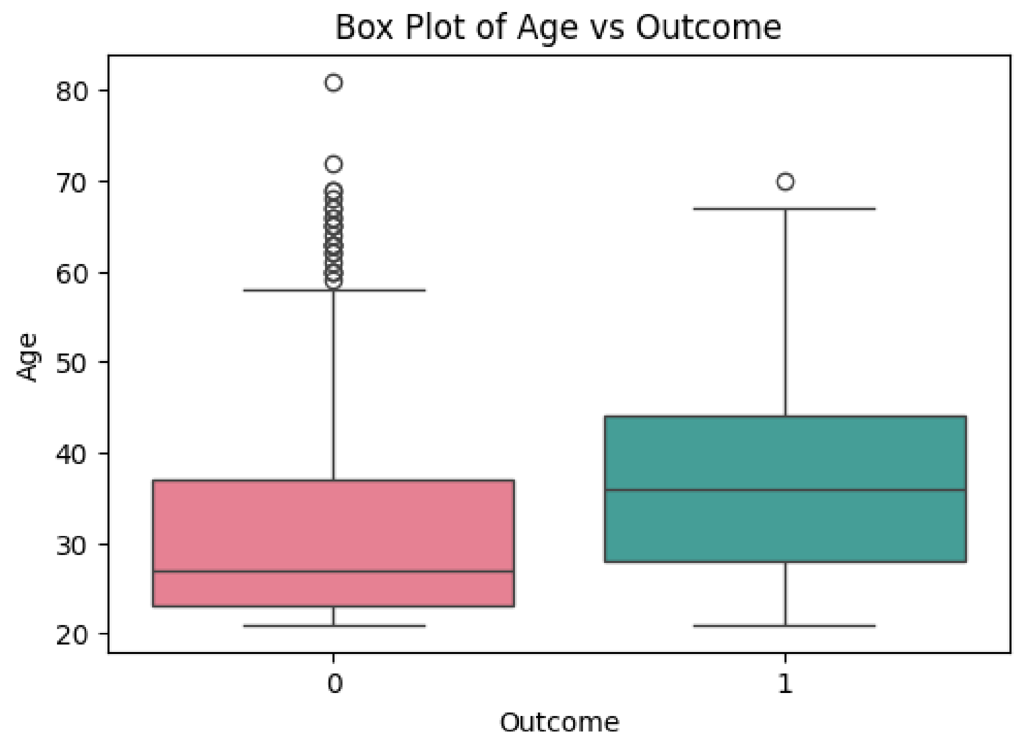
```
<ipython-input-7-568432ea0180>:40: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=data, x='Outcome', y=feature, palette="husl")
```



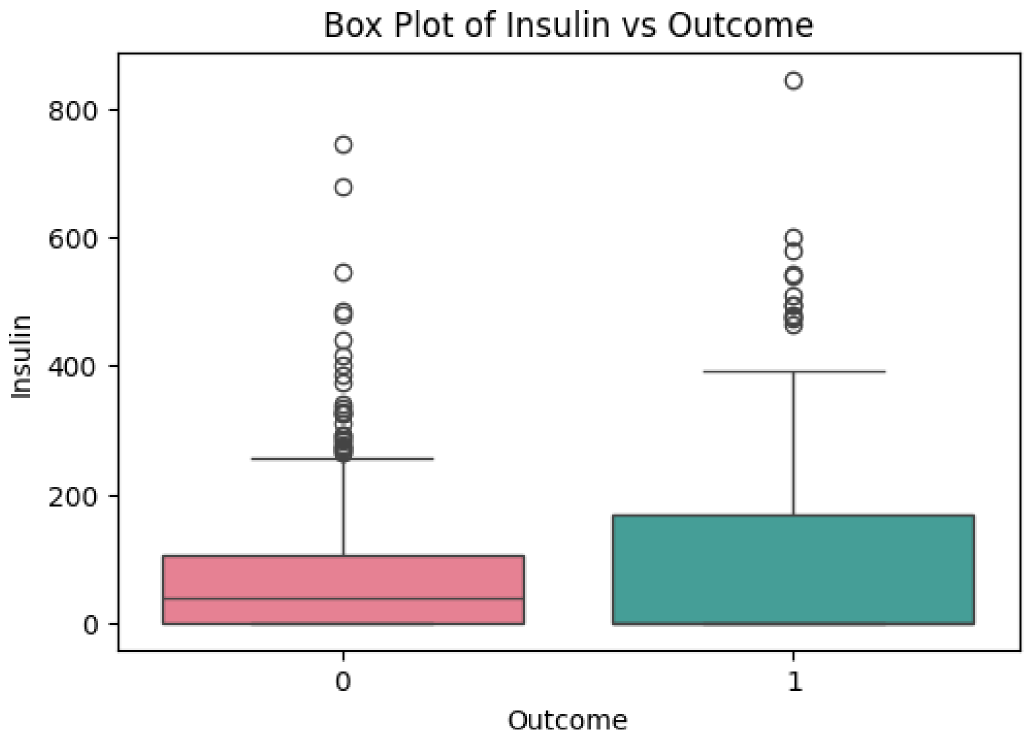
```
<ipython-input-7-568432ea0180>:40: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=data, x='Outcome', y=feature, palette="husl")
```



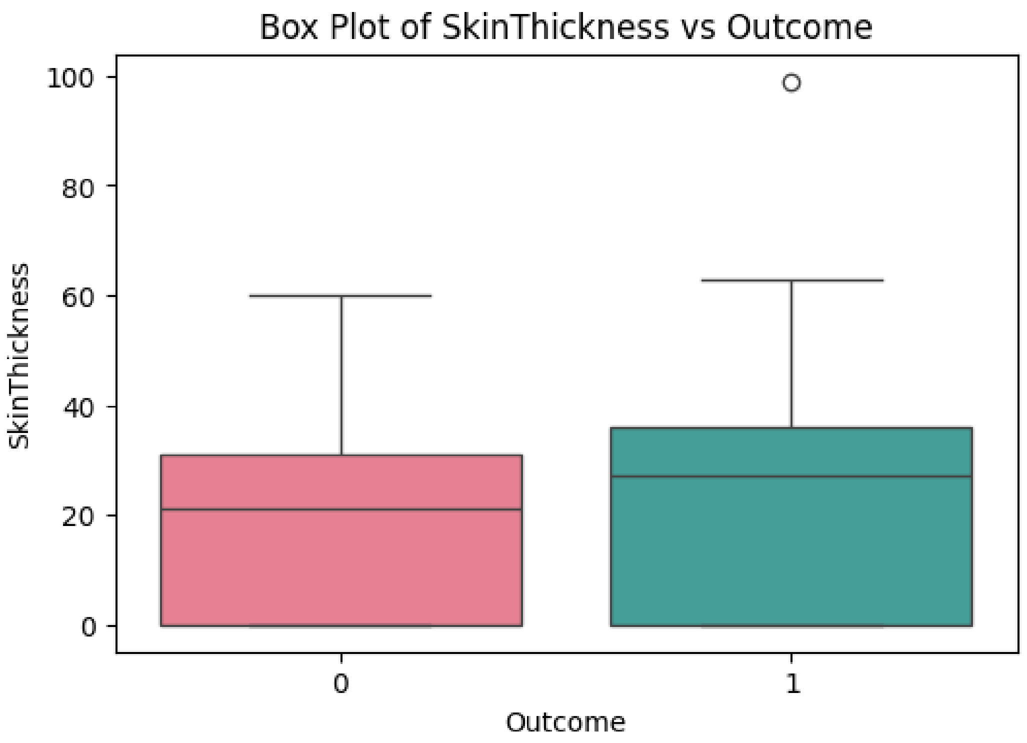
```
<ipython-input-7-568432ea0180>:40: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=data, x='Outcome', y=feature, palette="husl")
```



```
<ipython-input-7-568432ea0180>:40: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=data, x='Outcome', y=feature, palette="husl")
```



```
<ipython-input-7-568432ea0180>:40: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.boxplot(data=data, x='Outcome', y=feature, palette="husl")
```



```python
# let's look at the Corralation matrix of this data
corr_matrix = data.corr()
corr_matrix['Outcome'].sort_values(ascending=False)
```

```
Outcome                     1.000000
Glucose                     0.466581
BMI                         0.292695
Age                         0.238356
Pregnancies                 0.221898
DiabetesPedigreeFunction    0.173844
Insulin                     0.130548
SkinThickness               0.074752
BloodPressure               0.065068
Name: Outcome, dtype: float64
```

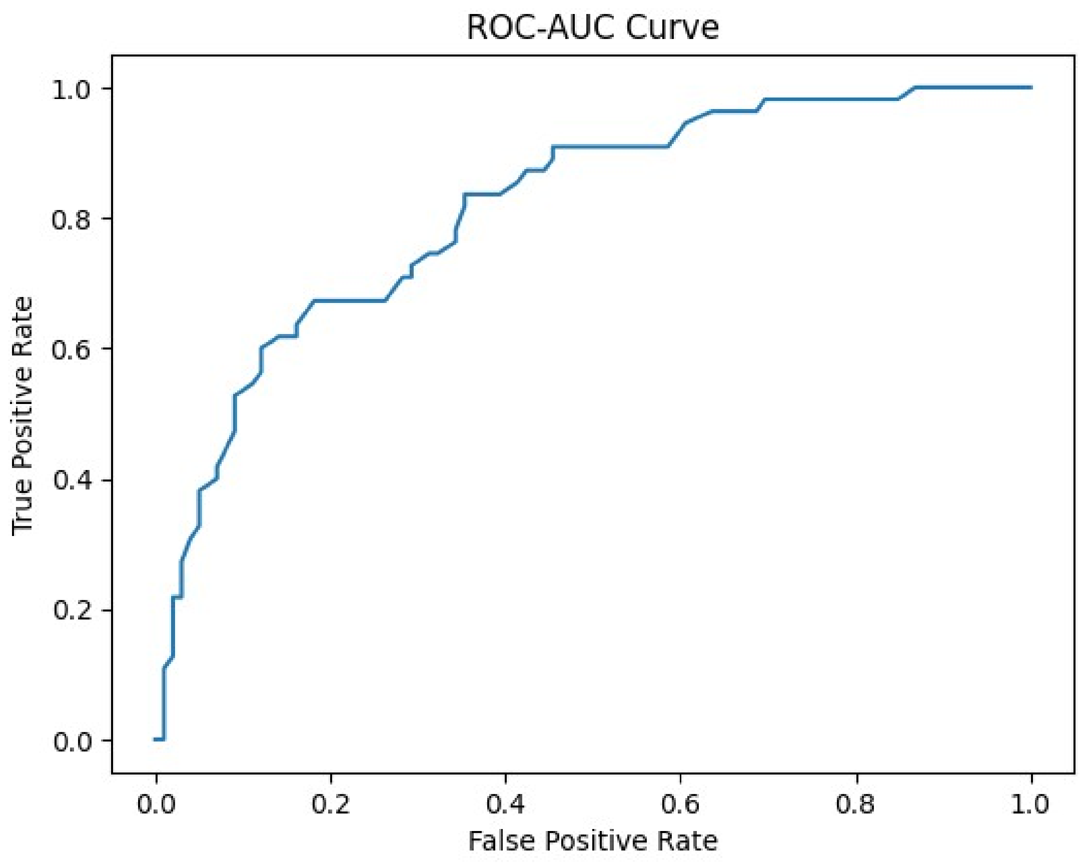
```python
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']  # Target
```

```python
from sklearn.feature_selection import SelectKBest, f_classif
#Feature Selection (Select Top 5 Features using ANOVA)
selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features.tolist())
```

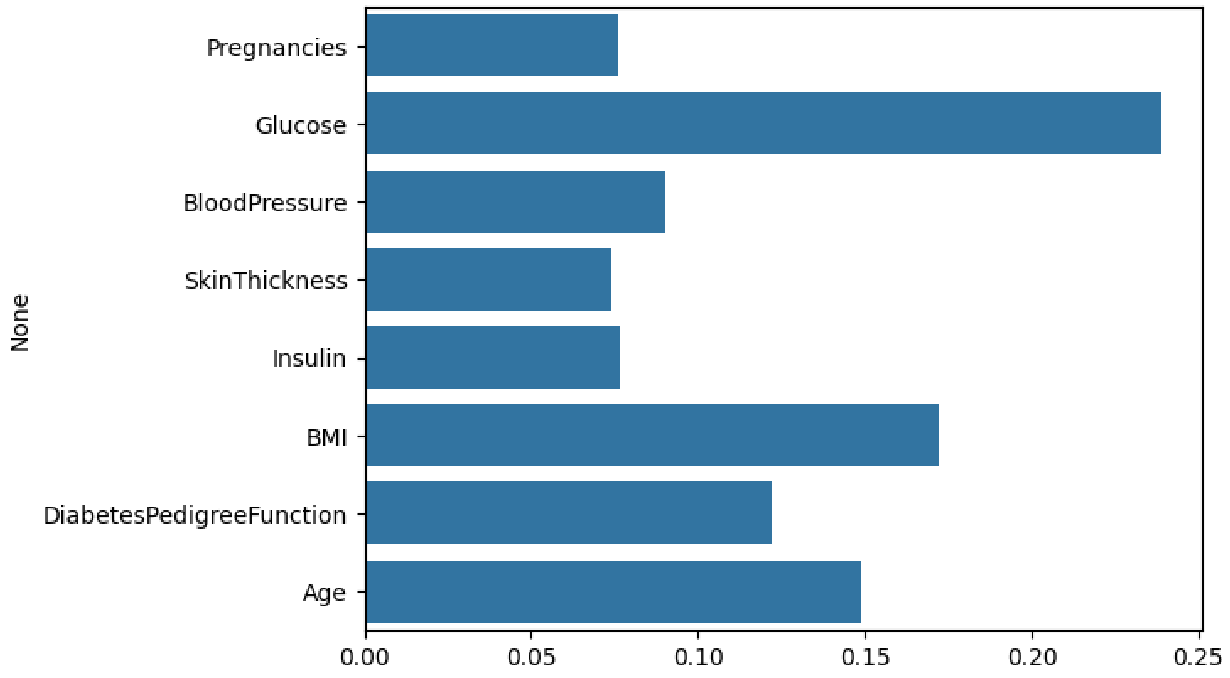
```
Selected Features: ['Pregnancies', 'Glucose', 'BMI',
'DiabetesPedigreeFunction', 'Age']
```

```
#Split the data into training and test sets:
from sklearn.model_selection import train_test_split

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

#select machine learing mdel
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

#Model Evaluation

# cross validarion
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
print("Cross-Validation Accuracy:", scores.mean())

# ROC-AUC curve
from sklearn.metrics import roc_auc_score, roc_curve
y_pred_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC-AUC Curve')
plt.show()


# Confusion Matrix
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))


# Precision and Recall
from sklearn.metrics import precision_score, recall_score
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
```

```
Cross-Validation Accuracy: 0.7683388506917919
```



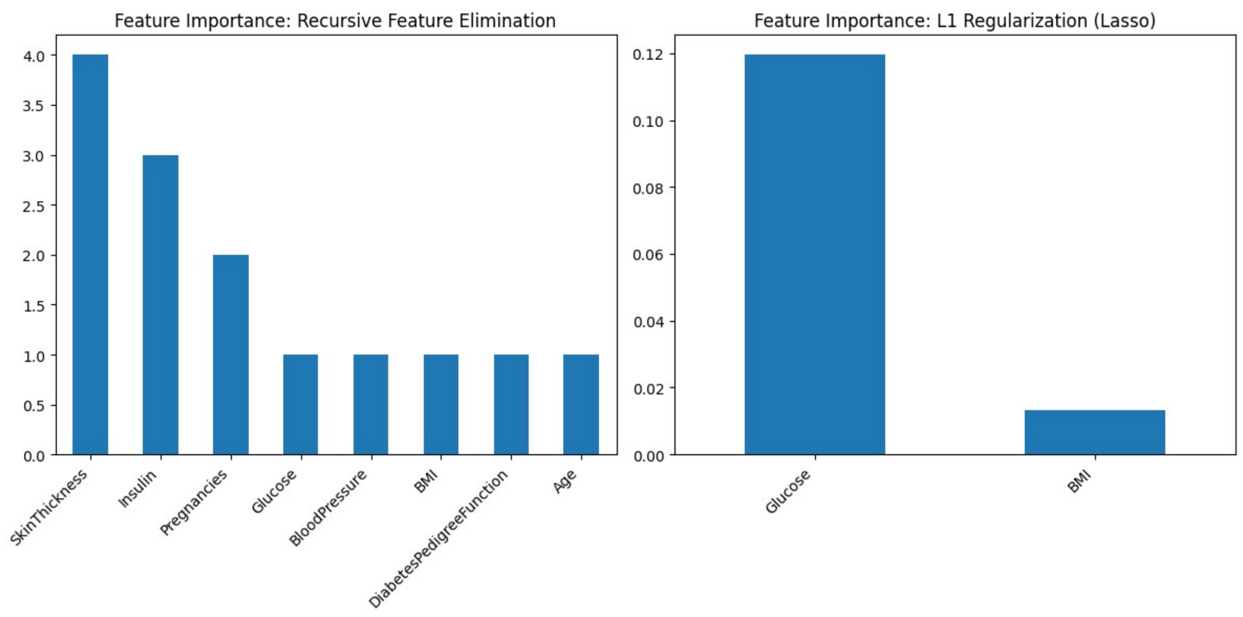
```
[[81 18]
 [18 37]]
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        99
           1       0.67      0.67      0.67        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154

Precision: 0.6727272727272727
Recall: 0.6727272727272727

# Visualize important features using the model's feature importances
(e.g., for Random Forest)

feature_importances = model.feature_importances_
sns.barplot(x=feature_importances, y=X.columns)
plt.show()
```



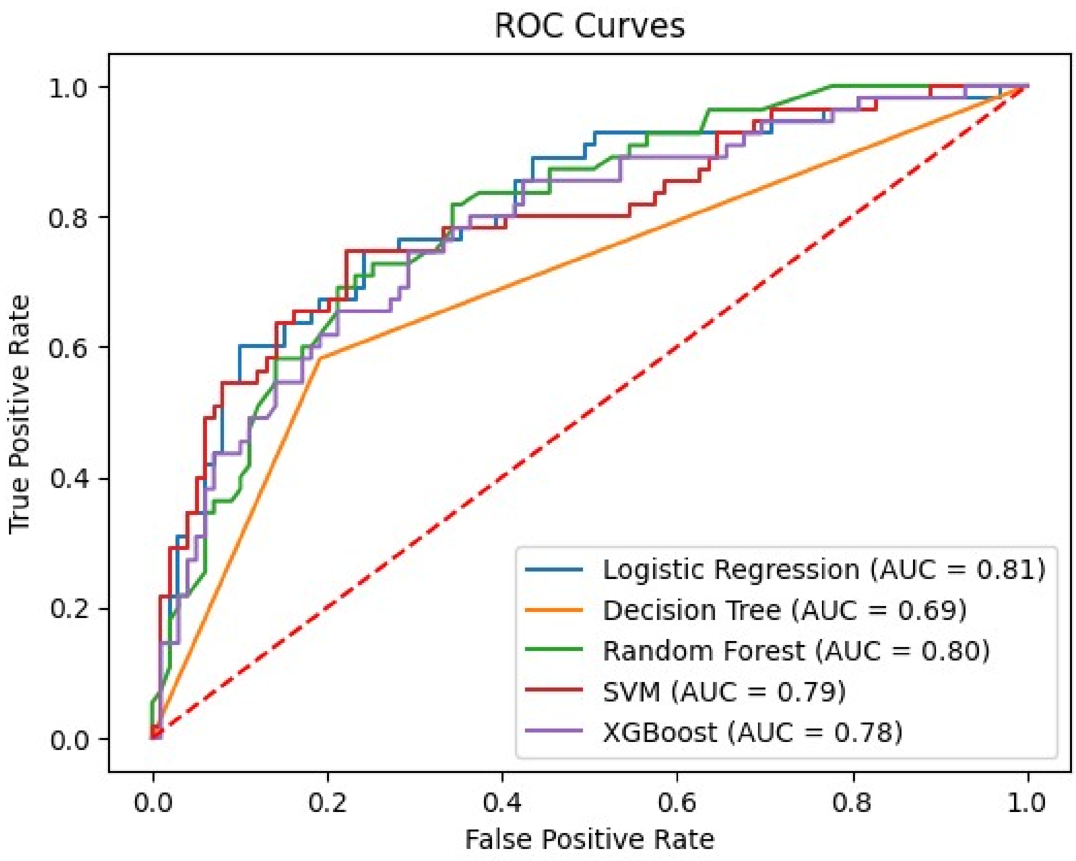
```
# Save the model using joblib for deployment:
import joblib
joblib.dump(model, 'random_forest_model.pkl')

# Load the saved model
loaded_model = joblib.load('/content/random_forest_model.pkl')

# 1. Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import (
    SelectKBest,
    f_classif,  # ANOVA F-value
    RFE
)
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression, Lasso
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, roc_auc_score,
roc_curve, confusion_matrix
from xgboost import XGBClassifier

# Advanced Feature Selection Function
def advanced_feature_selection(X, y):
    # Methods to compare
    feature_selection_methods = {
        'Univariate Selection (ANOVA)':
SelectKBest(score_func=f_classif, k=5),
        'Recursive Feature Elimination': RFE(
            estimator=RandomForestClassifier(),
            n_features_to_select=5
        ),
        'L1 Regularization (Lasso)': Lasso(alpha=0.1)
    }

    # Store selected features
    selected_features = {}
    feature_importances = {}

    # Standardize features for fair comparison
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Perform feature selection
    for name, selector in feature_selection_methods.items():
        # Fit selector
        if name == 'L1 Regularization (Lasso)':
            selector.fit(X_scaled, y)
            # Get feature importances for Lasso
            importances = np.abs(selector.coef_)
            selected_mask = importances > 0
            selected_features[name] = X.columns[selected_mask]
            feature_importances[name] = pd.Series(
                importances[selected_mask],
                index=X.columns[selected_mask]
            )
        elif hasattr(selector, 'get_support'):
            selector.fit(X_scaled, y)
            selected_features[name] =
X.columns[selector.get_support()]

            # For RFE, get feature ranking
            if name == 'Recursive Feature Elimination':
                feature_importances[name] = pd.Series(
                    selector.ranking_,
                    index=X.columns
                )
        else:
            selector.fit(X_scaled, y)
            selected_features[name] = X.columns

    # Visualize feature importances
    plt.figure(figsize=(12, 6))
    for i, (name, importances) in
```

```
enumerate(feature_importances.items(), 1):
        plt.subplot(1, len(feature_importances), i)
        importances.sort_values(ascending=False).plot(kind='bar')
        plt.title(f'Feature Importance: {name}')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
    plt.show()

    return selected_features

# 2. Load Dataset
data = pd.read_csv("/content/diabetes-dataset-for-beginners.zip")

# Check for missing values
print("Missing Values:\n", data.isnull().sum())

# Fill missing values with mean
data.fillna(data.mean(), inplace=True)

# Encode categorical variables (if any)
data = pd.get_dummies(data, drop_first=True)

# Perform advanced feature selection
X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Get selected features from different methods
selected_feature_sets = advanced_feature_selection(X, y)

# Print selected features
print("\nSelected Features:")
for method, features in selected_feature_sets.items():
    print(f"\n{method}:")
    print(features.tolist())

# Use the intersection of features from different methods
common_features = list(set.intersection(
    *[set(features) for features in selected_feature_sets.values()]
))

# If no common features, use features from the best method
if not common_features:
    # Prioritize methods in this order
    priority_methods = [
        'L1 Regularization (Lasso)',
        'Recursive Feature Elimination',
        'Univariate Selection (ANOVA)'
    ]

    for method in priority_methods:
        if len(selected_feature_sets[method]) > 0:
            common_features = list(selected_feature_sets[method])
            break

# Update X with selected features
X_selected = X[common_features]

# 3. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.2, random_state=42)

# 4. Define Models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(probability=True),
    "XGBoost": XGBClassifier(eval_metric='logloss')
}

# 5. Define Pipeline
pipelines = {
    name: Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', model)
    ])
    for name, model in models.items()
}

# 6. Train and Evaluate Models
results = {}
for name, pipeline in pipelines.items():
    print(f"\nTraining {name}...")
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    y_pred_proba = pipeline.predict_proba(X_test)[:, 1] if
hasattr(pipeline, 'predict_proba') else None

    # Evaluation Metrics
    print(f"Classification Report for {name}:\n",
classification_report(y_test, y_pred))
    confusion = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix for {name}:\n", confusion)

    # ROC-AUC Score
    if y_pred_proba is not None:
        roc_auc = roc_auc_score(y_test, y_pred_proba)
        print(f"ROC-AUC Score for {name}: {roc_auc:.2f}")
        results[name] = {'roc_auc': roc_auc}
        # Plot ROC Curve
        fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")
    else:
        results[name] = {'roc_auc': None}

# Show ROC Curves
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

# 7. Model Comparison
comparison = pd.DataFrame(results).T.sort_values(by='roc_auc',
ascending=False)
print("\nModel Performance Comparison:\n", comparison)

# 8. Save the Best Model
best_model_name = comparison.index[0]
best_pipeline = pipelines[best_model_name]
import joblib
joblib.dump(best_pipeline, f"{best_model_name}_pipeline.pkl")
print(f"\nBest Model ({best_model_name}) saved as
'{best_model_name}_pipeline.pkl'.")
```

```
Missing Values:
 Pregnancies                0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```



```
Selected Features:

Univariate Selection (ANOVA):
['Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction', 'Age']

Recursive Feature Elimination:
['Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'Age']

L1 Regularization (Lasso):
['Glucose', 'BMI']

Training Logistic Regression...
Classification Report for Logistic Regression:
               precision    recall  f1-score   support

           0       0.80      0.85      0.82        99
           1       0.69      0.62      0.65        55

    accuracy                           0.77       154
   macro avg       0.75      0.73      0.74       154
weighted avg       0.76      0.77      0.76       154

Confusion Matrix for Logistic Regression:
 [[84 15]
 [21 34]]
ROC-AUC Score for Logistic Regression: 0.81

Training Decision Tree...
Classification Report for Decision Tree:
               precision    recall  f1-score   support
           0       0.78      0.81      0.79        99
           1       0.63      0.58      0.60        55

    accuracy                           0.73       154
   macro avg       0.70      0.69      0.70       154
weighted avg       0.72      0.73      0.72       154

Confusion Matrix for Decision Tree:
 [[80 19]
 [23 32]]
ROC-AUC Score for Decision Tree: 0.69

Training Random Forest...
Classification Report for Random Forest:
               precision    recall  f1-score   support

           0       0.79      0.83      0.81        99
           1       0.66      0.60      0.63        55

    accuracy                           0.75       154
   macro avg       0.72      0.71      0.72       154
weighted avg       0.74      0.75      0.74       154

Confusion Matrix for Random Forest:
 [[82 17]
 [22 33]]
ROC-AUC Score for Random Forest: 0.80

Training SVM...
Classification Report for SVM:
               precision    recall  f1-score   support

           0       0.79      0.86      0.83        99
           1       0.70      0.60      0.65        55

    accuracy                           0.77       154
   macro avg       0.75      0.73      0.74       154
weighted avg       0.76      0.77      0.76       154

Confusion Matrix for SVM:
 [[85 14]
 [22 33]]
ROC-AUC Score for SVM: 0.79

Training XGBoost...
Classification Report for XGBoost:
               precision    recall  f1-score   support

           0       0.79      0.79      0.79        99
           1       0.62      0.62      0.62        55

    accuracy                           0.73       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.73      0.73      0.73       154

Confusion Matrix for XGBoost:
 [[78 21]
 [21 34]]
ROC-AUC Score for XGBoost: 0.78
```

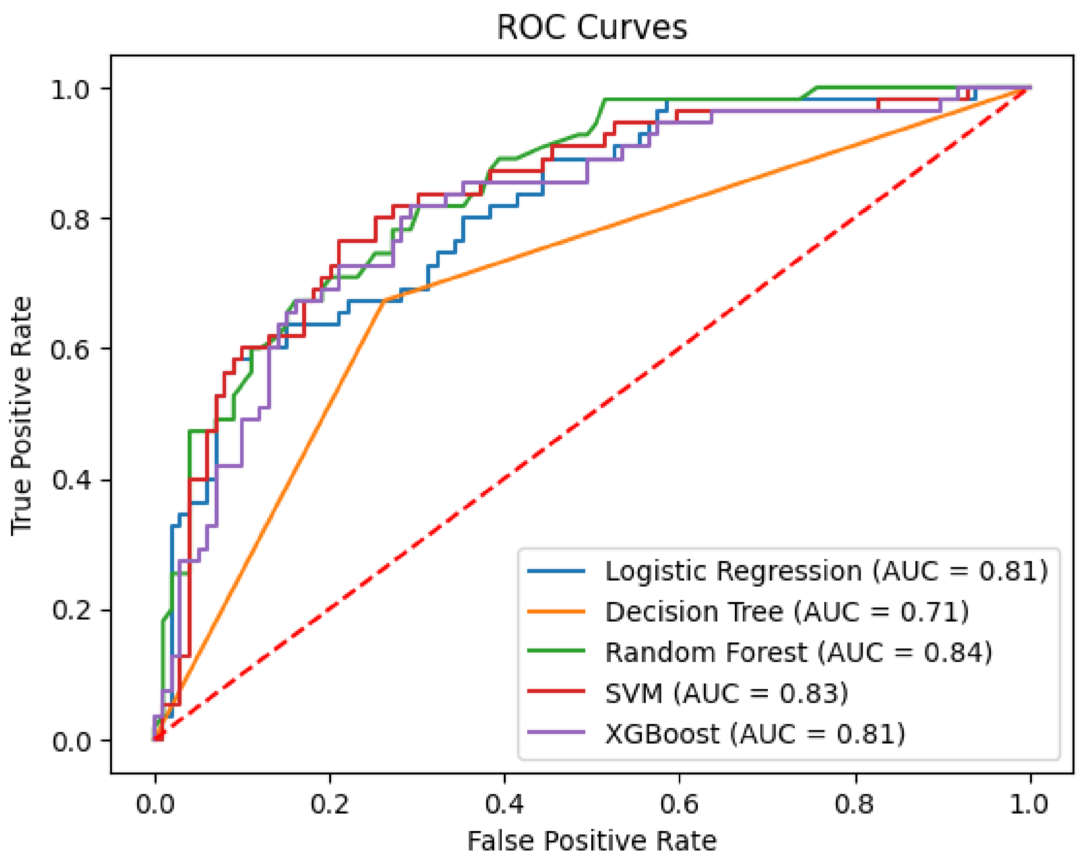


```
Model Performance Comparison:
                      roc_auc
Logistic Regression  0.808815
Random Forest        0.799541
SVM                  0.793205
XGBoost              0.779706
Decision Tree        0.694949

Best Model (Logistic Regression) saved as 'Logistic
Regression_pipeline.pkl'.
```

```python
# 1. Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, roc_auc_score,
roc_curve, confusion_matrix
from xgboost import XGBClassifier
import joblib

# 2. Load and Preprocess Data
data = pd.read_csv("/content/diabetes-dataset-for-beginners.zip")  #
Replace with your dataset path
data.fillna(data.mean(), inplace=True)  # Fill missing values
data = pd.get_dummies(data, drop_first=True)  # Encode categorical
variables

X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']  # Target

# 3. Feature Selection (Select Top 5 Features using ANOVA)
selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected Features:", selected_features.tolist())

# 4. Train-Test Split
X_train, X_test, y_train, y_test =
train_test_split(X[selected_features], y, test_size=0.2,
random_state=42)

# 5. Define Models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(probability=True),  # Enable probability for ROC
    "XGBoost": XGBClassifier(eval_metric='logloss')
}

# 6. Train and Evaluate Models
results = {}
for name, model in models.items():
    print(f"\nTraining {name}...")

    # Create pipeline: Scaling -> Model
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', model)
    ])
    pipeline.fit(X_train, y_train)  # Train

    # Predictions and Probabilities
    y_pred = pipeline.predict(X_test)
    y_pred_proba = pipeline.predict_proba(X_test)[:, 1] if
hasattr(pipeline, 'predict_proba') else None

    # Metrics
    print(f"\n{name} Classification Report:\n",
classification_report(y_test, y_pred))
    print(f"{name} Confusion Matrix:\n", confusion_matrix(y_test,
y_pred))

    # ROC-AUC
    if y_pred_proba is not None:
        roc_auc = roc_auc_score(y_test, y_pred_proba)
        print(f"{name} ROC-AUC Score: {roc_auc:.2f}")
        results[name] = roc_auc

        # Plot ROC Curve
        fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
        plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.2f})")

# Plot all ROC Curves
plt.plot([0, 1], [0, 1], 'r--')
plt.title("ROC Curves")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

# 7. Compare and Save Best Model
best_model_name = max(results, key=results.get)  # Get model with
highest ROC-AUC
print(f"\nBest Model: {best_model_name} with AUC =
{results[best_model_name]:.2f}")

# Save the best model pipeline
best_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', models[best_model_name])
])
best_pipeline.fit(X_train, y_train)
```

```
joblib.dump(best_pipeline, f"{best_model_name}_pipeline.pkl")
print(f"Best model saved as {best_model_name}_pipeline.pkl")

Selected Features: ['Pregnancies', 'Glucose', 'BMI',
'DiabetesPedigreeFunction', 'Age']

Training Logistic Regression...

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.82      0.81        99
           1       0.66      0.64      0.65        55

    accuracy                           0.75       154
   macro avg       0.73      0.73      0.73       154
weighted avg       0.75      0.75      0.75       154

Logistic Regression Confusion Matrix:
 [[81 18]
 [20 35]]
Logistic Regression ROC-AUC Score: 0.81

Training Decision Tree...

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.74      0.77        99
           1       0.59      0.67      0.63        55

    accuracy                           0.71       154
   macro avg       0.69      0.71      0.70       154
weighted avg       0.73      0.71      0.72       154

Decision Tree Confusion Matrix:
 [[73 26]
 [18 37]]
Decision Tree ROC-AUC Score: 0.71

Training Random Forest...

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        99
           1       0.67      0.67      0.67        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```

```
Random Forest Confusion Matrix:
 [[81 18]
 [18 37]]
Random Forest ROC-AUC Score: 0.84

Training SVM...

SVM Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.83      0.82        99
           1       0.68      0.65      0.67        55

    accuracy                           0.77       154
   macro avg       0.75      0.74      0.74       154
weighted avg       0.76      0.77      0.77       154

SVM Confusion Matrix:
 [[82 17]
 [19 36]]
SVM ROC-AUC Score: 0.83

Training XGBoost...

XGBoost Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.78      0.81        99
           1       0.65      0.73      0.68        55

    accuracy                           0.76       154
   macro avg       0.74      0.75      0.75       154
weighted avg       0.77      0.76      0.76       154

XGBoost Confusion Matrix:
 [[77 22]
 [15 40]]
XGBoost ROC-AUC Score: 0.81
```



```
Best Model: Random Forest with AUC = 0.84
Best model saved as Random Forest_pipeline.pkl
```