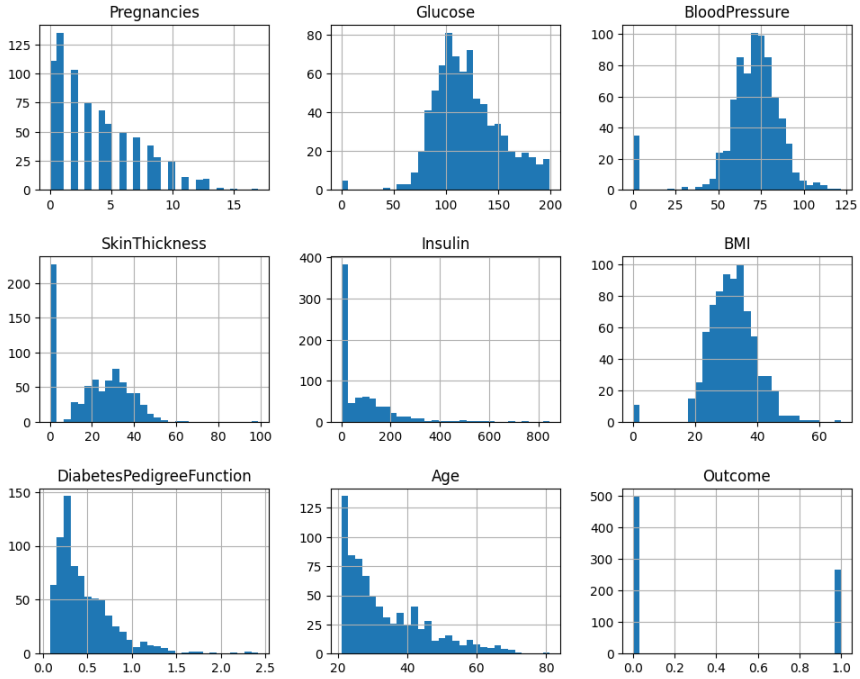
```
import pandas as pd
import matplotlib.pyplot as plt

diabetes=pd.read_csv("diabetes.csv")
print(diabetes.head())

print(diabetes.columns)
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
diabetes.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

```
#univariant analysis
diabetes.hist(figsize=(10,8),layout=(3,3),bins=30)
plt.tight_layout(pad=2.0)
plt.show()
```



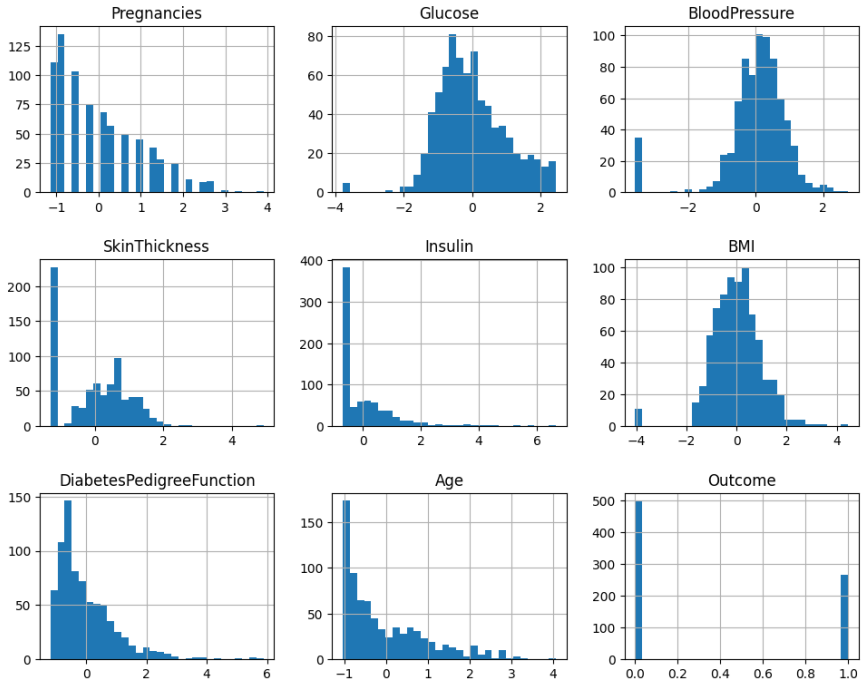

```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']
diabetes[columns]=scaler.fit_transform(diabetes[columns])
diabetes.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesI
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	

```
#preprocessing the data using mean
diabetes['Glucose'].replace(0,diabetes['Glucose'].mean(),inplace=True)
diabetes['BloodPressure'].replace(0,diabetes['BloodPressure'].mean(),inplace=True)
diabetes['Insulin'].replace(0,diabetes['Insulin'].mean(),inplace=True)
diabetes['SkinThickness'].replace(0,diabetes['SkinThickness'].mean(),inplace=True)
diabetes.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
count	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680(
mean	-6.476301e-17	-9.251859e-18	1.503427e-17	1.006140e-16	-3.006854e-17	2.590
std	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000(
min	-1.141852e+00	-3.783654e+00	-3.572597e+00	-1.288212e+00	-6.928906e-01	-4.060
25%	-8.448851e-01	-6.852363e-01	-3.673367e-01	-1.288212e+00	-6.928906e-01	-5.955
50%	-2.509521e-01	-1.218877e-01	1.496408e-01	1.545332e-01	-4.280622e-01	9.419

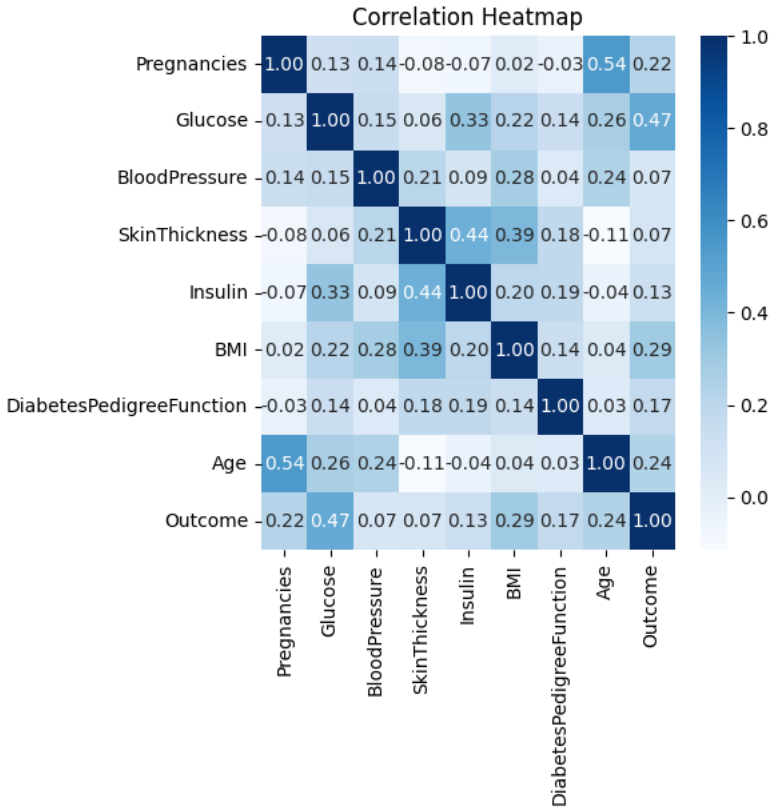
```
diabetes.hist(figsize=(10,8),layout=(3,3),bins=30)
plt.tight_layout(pad=2.0)
plt.show()
```




```
from sklearn.model_selection import cross_val_score,train_test_split
X=diabetes.drop(['Outcome'],axis=1)
Y=diabetes['Outcome']
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=100)

diabetes.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	

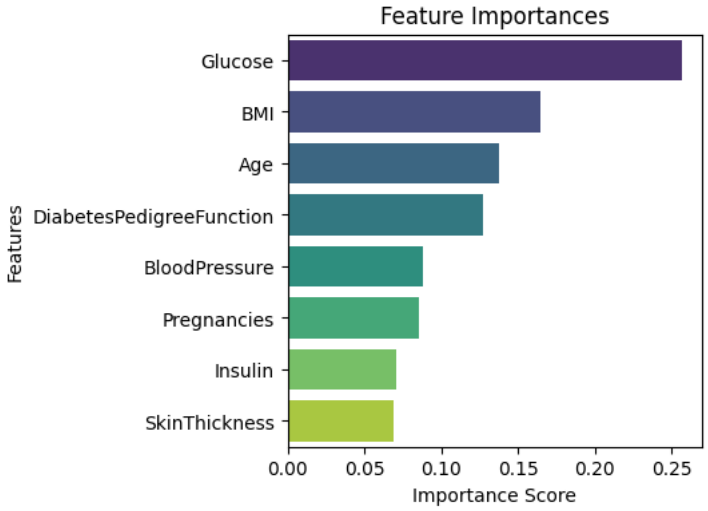
```
import seaborn as sns
correlation_matrix = diabetes.corr()

plt.figure(figsize=(5, 5))
sns.heatmap(correlation_matrix, annot=True, cmap='Blues', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```




```
#preprocessing the model to get more accuracy


# Finding the important feature importance
from sklearn.ensemble import RandomForestClassifier
X=diabetes.drop('Outcome',axis=1)
Y=diabetes['Outcome']

model=RandomForestClassifier()
model.fit(X,Y)

feature_importances=model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})
print(feature_importance_df.sort_values('Importance', ascending=False))
```

```
                     Feature  Importance
1                    Glucose    0.256626
5                        BMI    0.164623
7                        Age    0.137758
6  DiabetesPedigreeFunction    0.127614
2             BloodPressure    0.088148
0               Pregnancies    0.085360
4                    Insulin    0.071207
3             SkinThickness    0.068663
```

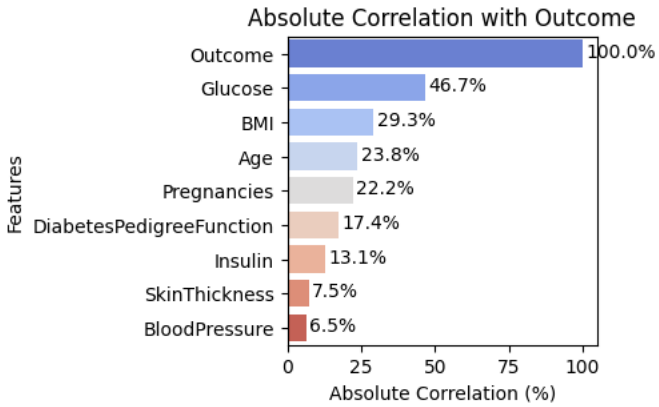
```
import matplotlib.pyplot as plt
import seaborn as sns

feature_importances = model.feature_importances_

feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Plotting feature importances using a bar plot
plt.figure(figsize=(4, 4))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df, palette='viridis')
plt.title('Feature Importances')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```




```
#correlation of each columns with the outcome
correlation_matrix = diabetes.corr()
correlation_with_target = correlation_matrix['Outcome'].abs().sort_values(ascending=False)
print(correlation_with_target)
```

```
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
```

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

correlation_matrix = diabetes.corr()

# Calculate absolute correlations with the Outcome
correlation_with_target = correlation_matrix['Outcome'].abs().sort_values(ascending=False)

# Convert absolute correlations to percentages
correlation_with_target_percent = correlation_with_target * 100

plt.figure(figsize=(3, 3))
ax = sns.barplot(x=correlation_with_target_percent.values, y=correlation_with_target.index, palette='coolwarm')
plt.title('Absolute Correlation with Outcome')
plt.xlabel('Absolute Correlation (%)')
plt.ylabel('Features')

# Adding percentage values to the bars
for i, v in enumerate(correlation_with_target_percent):
    ax.text(v + 1, i + .1, f'{v:.1f}%', color='black', fontsize=10, ha='left')

plt.show()
```




```
threshold = 0.1
low_variance_cols = diabetes.columns[diabetes.var() < threshold]
diabetes.drop(low_variance_cols, axis=1, inplace=True)
low_variance_cols
```

```
Index([], dtype='object')
```

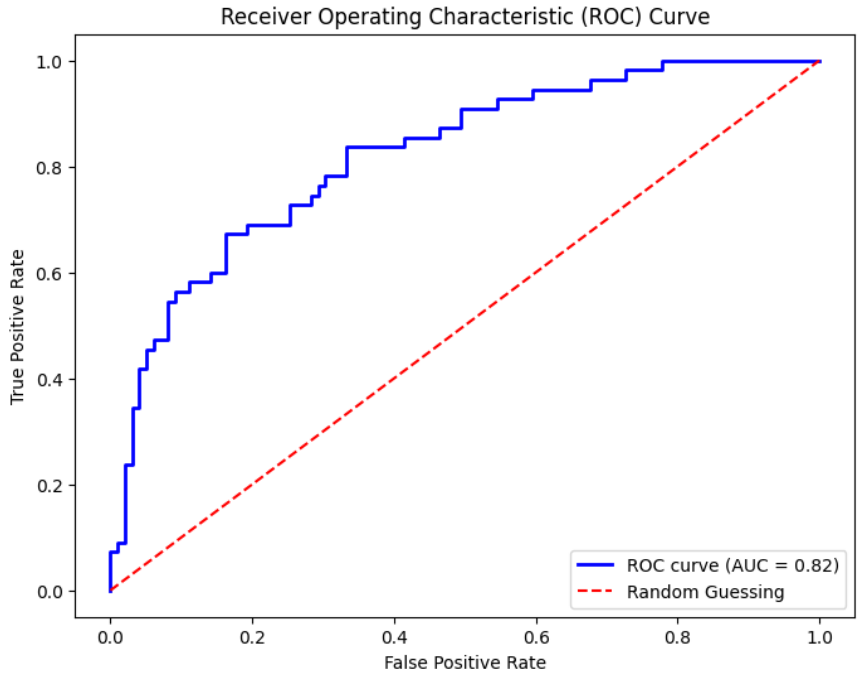
```
#it is concluded that no feature can be dropped from the dataset
#so we can't perform feature engineering
```

```
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

#random seed for reproducibility
seed_value = 100

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=seed_value)

#hyperparameter grids for RandomForest and XGBoost
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

xgb_param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7]
}

# Perform GridSearchCV for RandomForest with a fixed random_state
rf_grid_search = GridSearchCV(RandomForestClassifier(random_state=seed_value), rf_param_grid, cv=StratifiedKFold(n_splits=5, shuffle=Tru
rf_grid_search.fit(X_train, Y_train)
best_rf = rf_grid_search.best_estimator_

# Perform GridSearchCV for XGBoost with a fixed random_state
xgb_grid_search = GridSearchCV(xgb.XGBClassifier(objective='binary:logistic', random_state=seed_value), xgb_param_grid, cv=StratifiedKF
xgb_grid_search.fit(X_train, Y_train)
best_xgb = xgb_grid_search.best_estimator_

# Combine the best models into a stacking classifier
estimators = [
    ('rf', best_rf),
    ('xgb', best_xgb)
]

# Initialize a Logistic Regression final estimator without a seed
final_estimator = LogisticRegression()


stacking_model = StackingClassifier(estimators=estimators, final_estimator=final_estimator, cv=StratifiedKFold(n_splits=5, shuffle=True,
stacking_model.fit(X_train, Y_train)

# Evaluate the stacked model
stacked_predictions = stacking_model.predict(X_test)
stacked_accuracy = accuracy_score(Y_test, stacked_predictions)
stacked_conf_matrix = confusion_matrix(Y_test, stacked_predictions)
stacked_class_report = classification_report(Y_test, stacked_predictions)

print("Stacked Model Accuracy:", round(stacked_accuracy, 2))
print("Confusion Matrix:\n", stacked_conf_matrix)
print("Classification Report:\n", stacked_class_report)
```

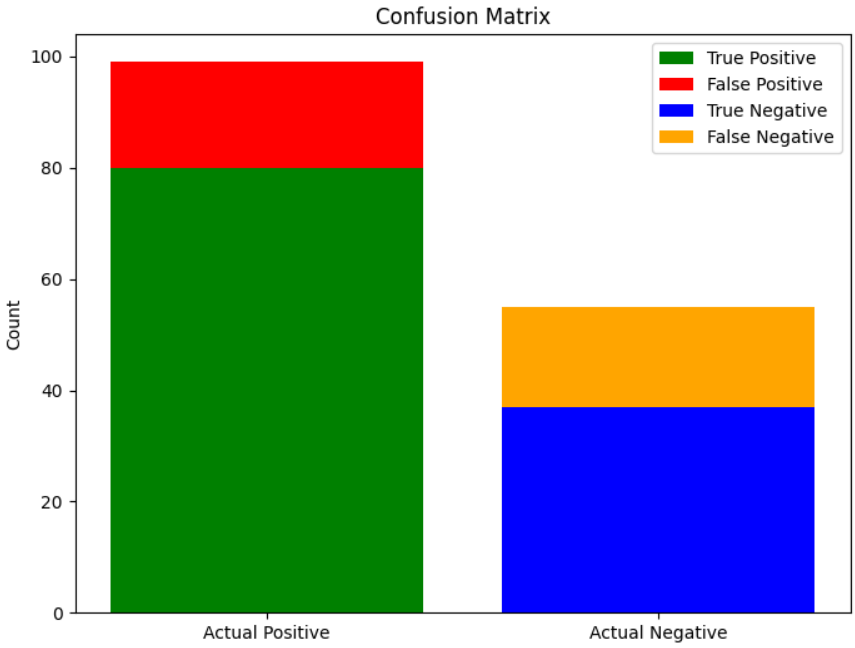
```
Stacked Model Accuracy: 0.76
Confusion Matrix:
 [[80 19]
 [18 37]]
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.81      0.81        99
           1       0.66      0.67      0.67        55

    accuracy                           0.76       154
   macro avg       0.74      0.74      0.74       154
weighted avg       0.76      0.76      0.76       154
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

probs = stacking_model.predict_proba(X_test)
preds = probs[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(Y_test, preds)
roc_auc = auc(fpr, tpr)
print(roc_auc)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

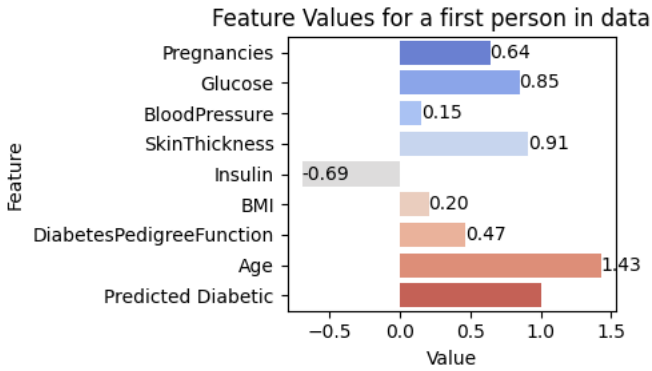
```
0.8225895316804407
```




```
#confusion matrix values
true_positive, false_positive, false_negative, true_negative = stacked_conf_matrix.ravel()

# Plotting the stacked bar chart
plt.figure(figsize=(8, 6))
plt.bar(0, true_positive, color='green', label='True Positive')
plt.bar(0, false_positive, bottom=true_positive, color='red', label='False Positive')
plt.bar(1, true_negative, color='blue', label='True Negative')
plt.bar(1, false_negative, bottom=true_negative, color='orange', label='False Negative')

plt.xticks([0, 1], ['Actual Positive', 'Actual Negative'])
plt.ylabel('Count')
plt.title('Confusion Matrix')
plt.legend()
plt.show()
```




```
#training an XGBoost model
model = xgb.XGBClassifier()
model.fit(X, Y)

model.save_model('xgb_diabetes_model.json')

loaded_model = xgb.XGBClassifier()
loaded_model.load_model('xgb_diabetes_model.json')

single_row = X.iloc[[0]]

# Predict whether the individual has diabetes or not using the loaded model
predicted_diabetes = loaded_model.predict(single_row)

# Get the feature names from the dataset
feature_names = list(diabetes.columns)[:-1]
feature_values = single_row.values[0]

feature_data = pd.DataFrame({'Feature': feature_names, 'Value': feature_values})

predicted_diabetes_data = pd.DataFrame({'Feature': ['Predicted Diabetic'], 'Value': [predicted_diabetes[0]]})

plt.figure(figsize=(5, 3))


ax = sns.barplot(x="Value", y="Feature", data=pd.concat([feature_data, predicted_diabetes_data]), palette='coolwarm')
plt.title('Feature Values for a first person in data set')

# Highlight the predicted diabetic feature with red color
for index, value in enumerate(feature_values):
    if feature_data.iloc[index]['Feature'] == 'Predicted Diabetic':
        color = 'red' if predicted_diabetes[0] == 1 else 'skyblue'
        ax.barh(index, value, color=color)
    else:
        # Display the feature values on the bar plot for non-predicted diabetic features
        ax.text(value, index, f'{value:.2f}', va='center')

plt.tight_layout()
plt.show()
if predicted_diabetes[0] == 0:
    print("Predicted Outcome: Non-Diabetic")
else:
    print("Predicted Outcome: Diabetic")
```

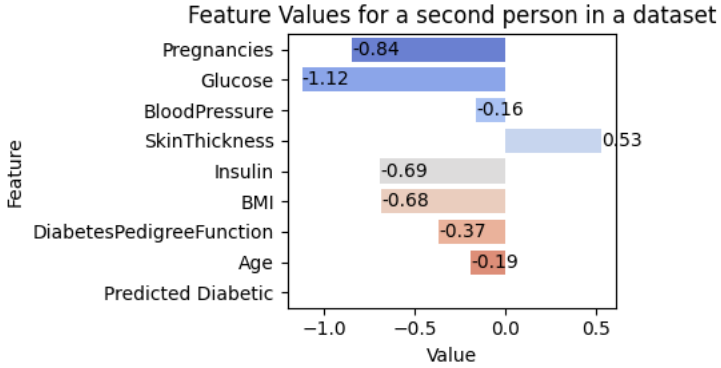



```
Predicted Outcome: Diabetic
```

```
model = xgb.XGBClassifier()
model.fit(X, Y)
```

```
model.save_model('xgb_diabetes_model.json')
```

```
loaded_model = xgb.XGBClassifier()
loaded_model.load_model('xgb_diabetes_model.json')

single_row = X.iloc[[1]]

# Predict whether the individual has diabetes or not using the loaded model
predicted_diabetes = loaded_model.predict(single_row)

# Get the feature names from the dataset
feature_names = list(diabetes.columns)[:-1]
feature_values = single_row.values[0]

feature_data = pd.DataFrame({'Feature': feature_names, 'Value': feature_values})

predicted_diabetes_data = pd.DataFrame({'Feature': ['Predicted Diabetic'], 'Value': [predicted_diabetes[0]]})

plt.figure(figsize=(5,3))

ax = sns.barplot(x="Value", y="Feature", data=pd.concat([feature_data, predicted_diabetes_data]), palette='coolwarm')
plt.title('Feature Values for a second person in a dataset')

# Highlight the predicted diabetic feature with red color
for index, value in enumerate(feature_values):
    if feature_data.iloc[index]['Feature'] == 'Predicted Diabetic':
        color = 'red' if predicted_diabetes[0] == 1 else 'skyblue'
        ax.barh(index, value, color=color)
    else:
        # Display the feature values on the bar plot for non-predicted diabetic features
        ax.text(value, index, f'{value:.2f}', va='center')

plt.tight_layout()
plt.show()
if predicted_diabetes[0] == 0:
    print("Predicted Outcome: Non-Diabetic")
else:
    print("Predicted Outcome: Diabetic")
```




```
Predicted Outcome: Non-Diabetic
```

```
pip install dash plotly pandas scikit-learn xgboost
```

```
Collecting dash
  Downloading dash-2.14.2-py3-none-any.whl (10.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.2/10.2 MB 17.4 MB/s eta 0:00:00
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from dash) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (from dash) (3.0.1)
Collecting dash-html-components==2.0.0 (from dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash) (4.5.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from dash) (2.31.0)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Collecting ansi2html (from dash)
  Downloading ansi2html-1.9.1-py3-none-any.whl (17 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from dash) (1.5.8)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dash) (67.7.2)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from dash) (7.0.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (23.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
```