Task No# 02: Disease Prediction

Introduction

- **Predictive Modeling for Disease Diagnosis**
- **Objective:** Develop a predictive model for accurate disease classification based on health attributes.
- **Importance:** Aid healthcare providers in diagnosis and prognosis.

Submitted By: SYED IRTIZA ABBAS ZAIDI

Problem Statement

- **Objective:** To classify individuals into diseased or non-diseased categories.
- Leveraging machine learning algorithms for reliable predictions.

Step 1:- Import Libraries

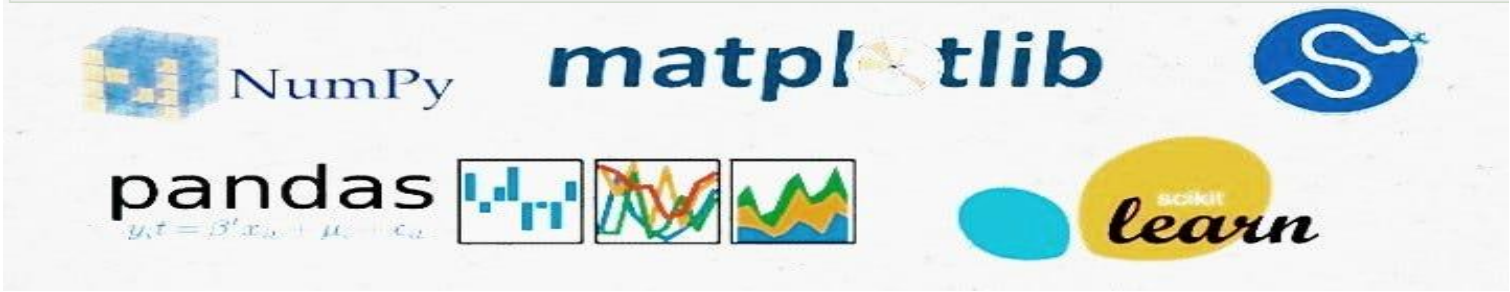
- **Explanation:** Importing necessary libraries for data manipulation, visualization, and modeling.

- **Libraries** include pandas, Numpy, Matplotlib, Seaborn, Plotly, and Sklearn.

```
# Importing Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

✓ 1m 55.5s



Step 2:- Load Data

- **Description:** Loading training and test datasets for model development and evaluation.

- Utilizing pandas to read CSV files containing health attribute data.

```
# Load the training and test datasets
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')
```

✓ 1.2s

Step 3:- Data Preprocessing

- **Overview:** Preprocessing steps including handling missing values and feature engineering.

- **Visualization:** Heatmaps to visualize missing values in the datasets.

Step 4:- Feature Engineering

- **Techniques:** Creating new features to enhance model performance.
- **Examples:** Squaring BMI, interaction features, and logarithmic transformation of Insulin.

```
# Create new features
X_train['BMI_squared'] = X_train['BMI'] ** 2
X_test['BMI_squared'] = X_test['BMI'] ** 2

X_train['Glucose_Cholesterol_interaction'] = X_train['Glucose'] * X_train['Cholesterol']
X_test['Glucose_Cholesterol_interaction'] = X_test['Glucose'] * X_test['Cholesterol']

X_train['Log_Insulin'] = np.log(X_train['Insulin'] + 1)
X_test['Log_Insulin'] = np.log(X_test['Insulin'] + 1)
```

✓ 0.3s

Step 5:- Feature Scaling

- **Purpose:** Scaling features to a standard range for improved model training.
- **Implementation:** Utilizing StandardScaler from sklearn.preprocessing.

```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

✓ 0.0s

Step 6:- Model Selection and Hyperparameter Tuning

- **Approach:** Employing Random Forest classifier for predictive modeling.
- **Tuning:** GridSearchCV for finding optimal hyperparameters.

```
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the Random Forest classifier
rf = RandomForestClassifier()

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Get best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Train the model with best parameters
best_rf_model = RandomForestClassifier(**best_params)
best_rf_model.fit(X_train_scaled, y_train)
```

✓ 3m 31.7s

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=50)
```

Step 7:- Model Evaluation

- **Metrics:** Evaluating model performance using accuracy, precision, recall, and F1-score.

- **Comparison:** Training and test performance metrics for assessing model generalization.

```
    # Predictions
    y_pred_train = best_rf_model.predict(X_train_scaled)
    y_pred_test = best_rf_model.predict(X_test_scaled)
[27] ✓ 0.0s
```

```
    # Evaluate the model
    train_accuracy = accuracy_score(y_train, y_pred_train) * 100
    test_accuracy = accuracy_score(y_test, y_pred_test) * 100

    # For multiclass classification, use average='macro', 'micro', or 'weighted'
    train_precision = precision_score(y_train, y_pred_train, average='weighted') * 100
    test_precision = precision_score(y_test, y_pred_test, average='weighted') * 100

    train_recall = recall_score(y_train, y_pred_train, average='weighted') * 100
    test_recall = recall_score(y_test, y_pred_test, average='weighted') * 100

    train_f1 = f1_score(y_train, y_pred_train, average='weighted') * 100
    test_f1 = f1_score(y_test, y_pred_test, average='weighted') * 100

    print(f"Training Accuracy: {train_accuracy:.2f}%")
    print(f"Test Accuracy: {test_accuracy:.2f}%")
    print(f"\nTraining Precision: {train_precision:.2f}%")
    print(f"Test Precision: {test_precision:.2f}%")
    print(f"\nTraining Recall: {train_recall:.2f}%")
    print(f"Test Recall: {test_recall:.2f}%")
    print(f"\nTraining F1-score: {train_f1:.2f}%")
    print(f"Test F1-score: {test_f1:.2f}%")
[28] ✓ 0.6s
```

```
Training Accuracy: 100.00%
Test Accuracy: 44.03%

Training Precision: 100.00%
Test Precision: 50.94%

Training Recall: 100.00%
Test Recall: 44.03%

Training F1-score: 100.00%
Test F1-score: 44.65%
```

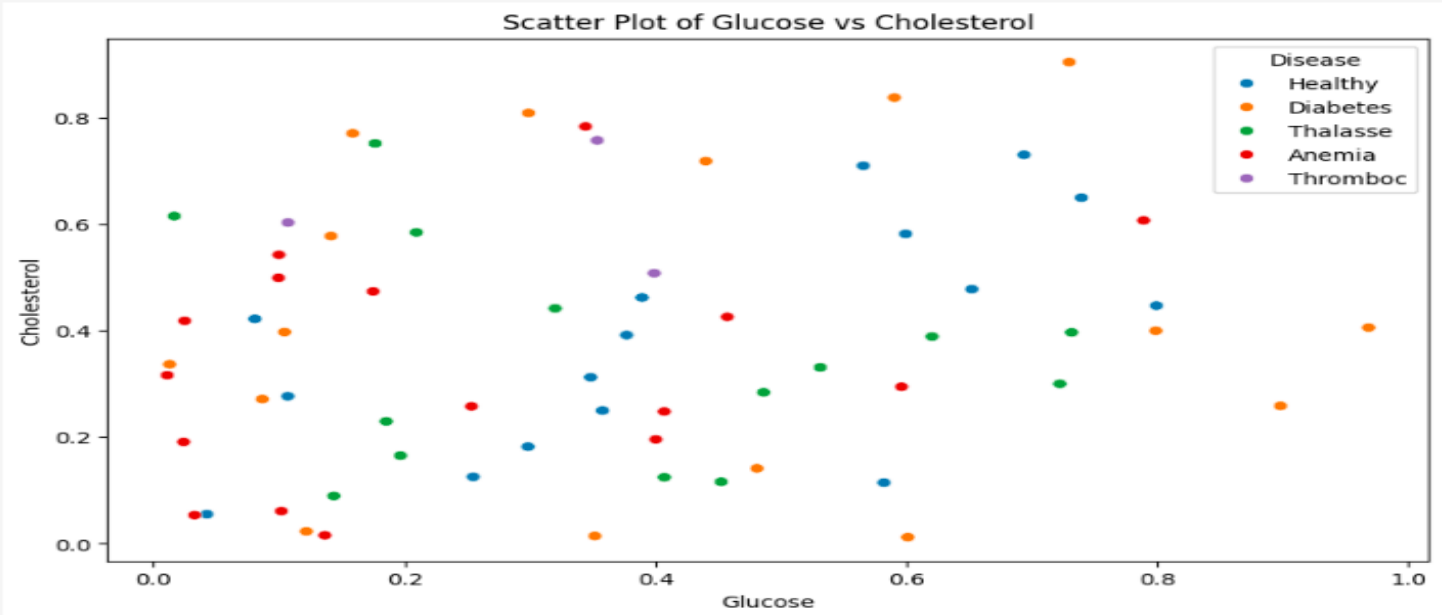
Step 8:- Visualize Feature Distributions

Visualizations: Scatter plot, pie chart, bar plot, and histogram of health attributes.

Insights: Understanding feature distributions and their impact on disease classification.

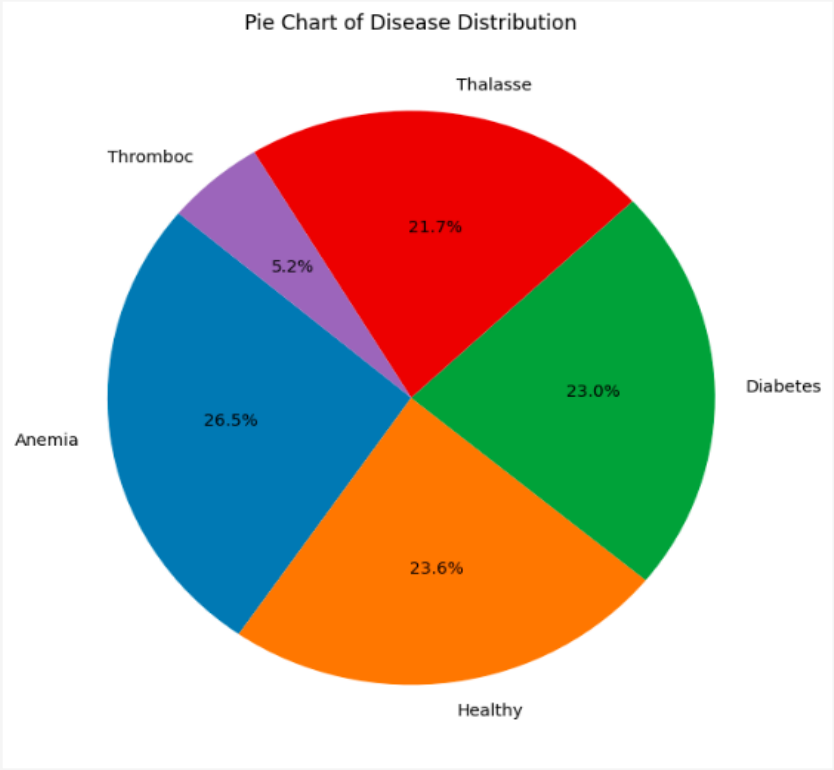
```
# Scatter Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=train_data, x='Glucose', y='Cholesterol', hue='Disease')
plt.title('Scatter Plot of Glucose vs Cholesterol')
plt.xlabel('Glucose')
plt.ylabel('Cholesterol')
plt.show()
```

✓ 1.1s



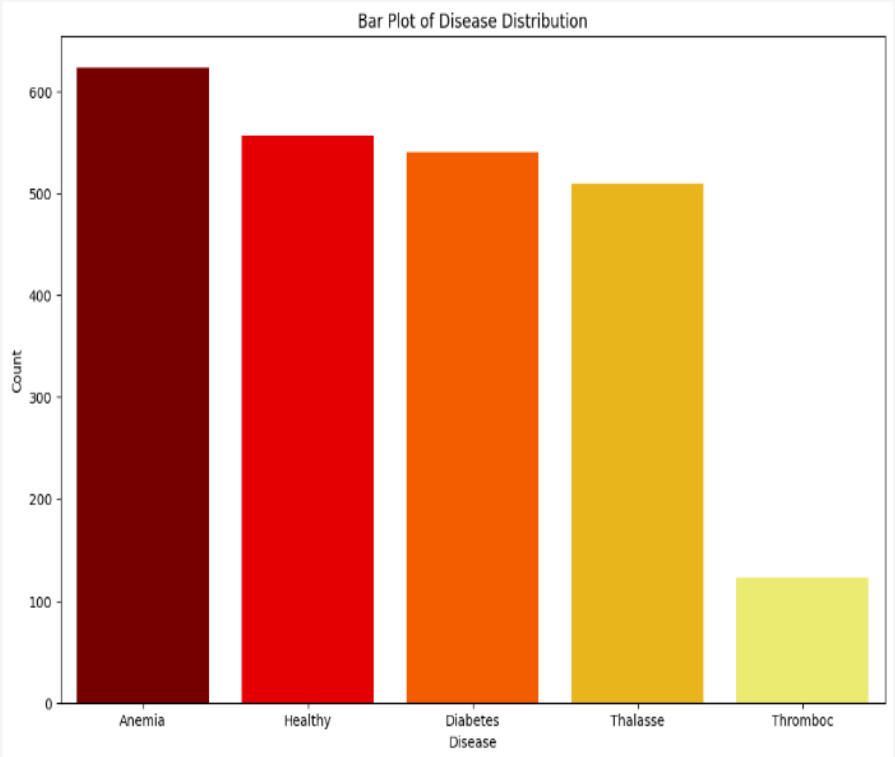
```python
# Pie Chart
disease_counts = train_data['Disease'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(disease_counts, labels=disease_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Pie Chart of Disease Distribution')
plt.show()
```

✓ 0.6s



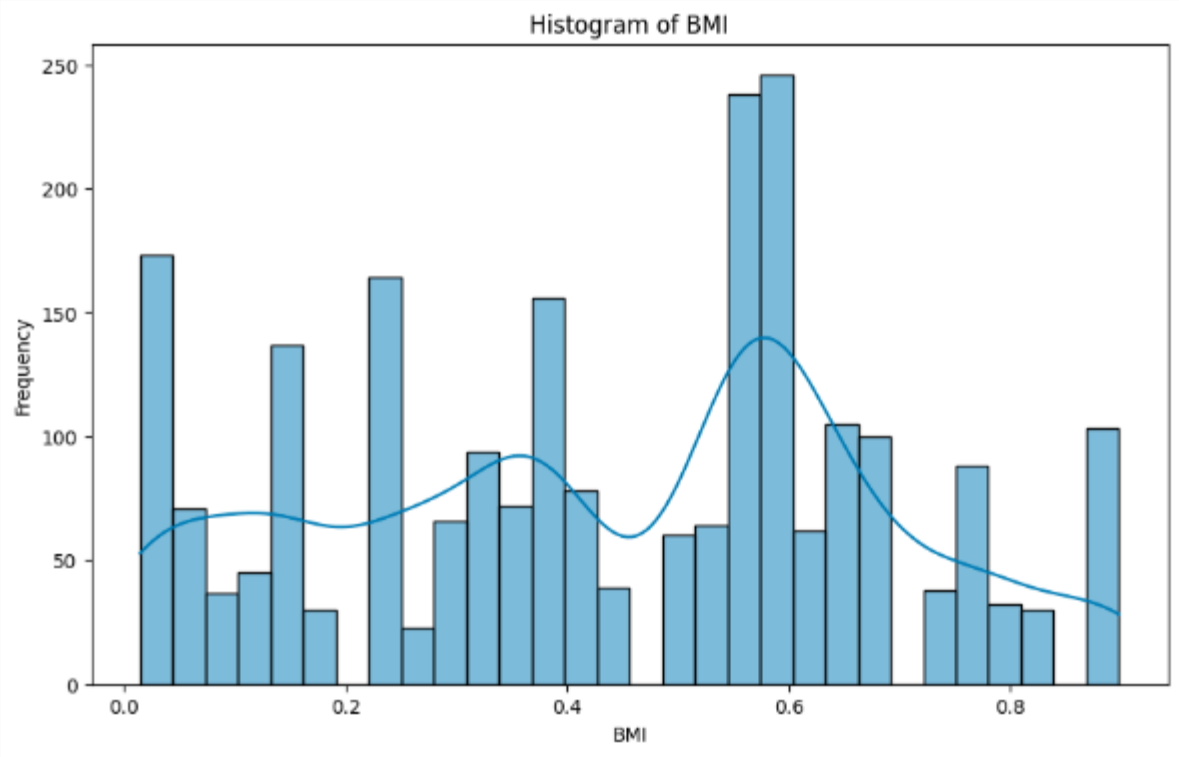

```python
# Bar Plot
plt.figure(figsize=(12, 8))
sns.barplot(x=disease_counts.index, y=disease_counts.values, palette="hot")
plt.title('Bar Plot of Disease Distribution')
plt.xlabel('Disease')
plt.ylabel('Count')
plt.show()
```

✓ 1.0s



```python
# Histogram
plt.figure(figsize=(10, 6))
sns.histplot(train_data['BMI'], kde=True, bins=30)
plt.title('Histogram of BMI')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.show()
```

✓ 1.0s



```
# Example data for Sunburst Chart
sunburst_data = train_data.copy()
sunburst_data['BMI_Category'] = pd.cut(train_data['BMI'], bins=[0, 18.5, 25, 30, 100], labels=['Underweight', 'Normal', 'Overweight', 'Obese'])
sunburst_data['Blood_Pressure_Category'] = pd.cut(train_data['Systolic Blood Pressure'], bins=[0, 120, 130, 140, 200], labels=['Normal', 'Elevated', 'Hypertension Stage 1', 'Hypertension Stage 2'])

fig = px.sunburst(sunburst_data, path=['Disease', 'BMI_Category', 'Blood_Pressure_Category'],
                  values='BMI', color='Disease', color_continuous_scale='RdBu')
fig.update_layout(title='Sunburst Chart of Disease, BMI, and Blood Pressure Categories')
fig.show()
```

✓ 57.3s

Sunburst Chart of Disease, BMI, and Blood Pressure Categories

Conclusion

- **Summary:** Recapitulation of steps undertaken for predictive modeling.
- **Implications:** Potential applications of the developed model in healthcare settings.

Question and Answer

- Inviting questions and discussions from the audience.


ThankYou
IMAGINED WITH AI