assignment-16-09-02-2024

February 11, 2024

1 Gunjan Chakraborty

1.1 22MSRDS007

1.1.1 09/02/2024

[1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.simplefilter(action='ignore')
```

[2]:
```
# Load the dataset
df = pd.read_csv('D:/Chools/Day_10/diabetes.csv')
```

1.1.2 1. Exploratory Data Analysis (EDA):

[3]:
```
print(df.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
None
```

[4]:
```
print(df.describe())
```

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

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[5]:

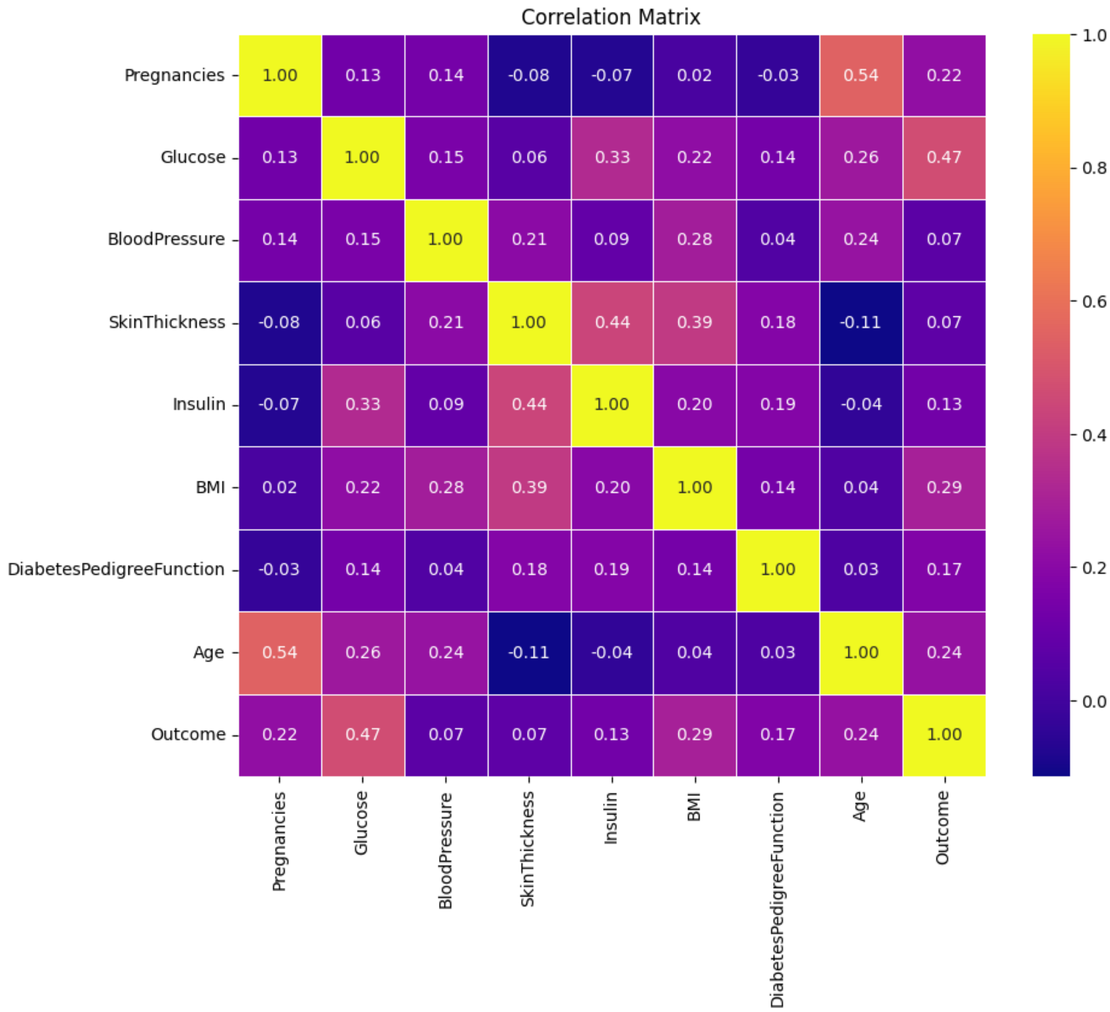
```python
print(df.isnull().sum())
```

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

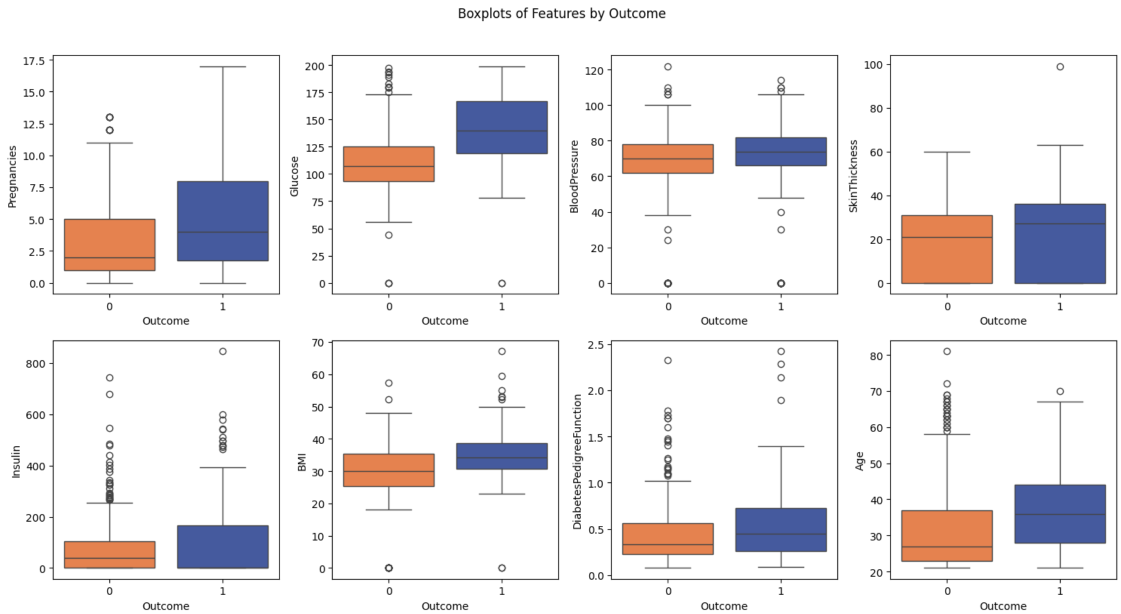
[6]:

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

sns.pairplot(df, hue='Outcome')
plt.suptitle('Pairplot of Features by Outcome', y=1.02)
plt.show()
```

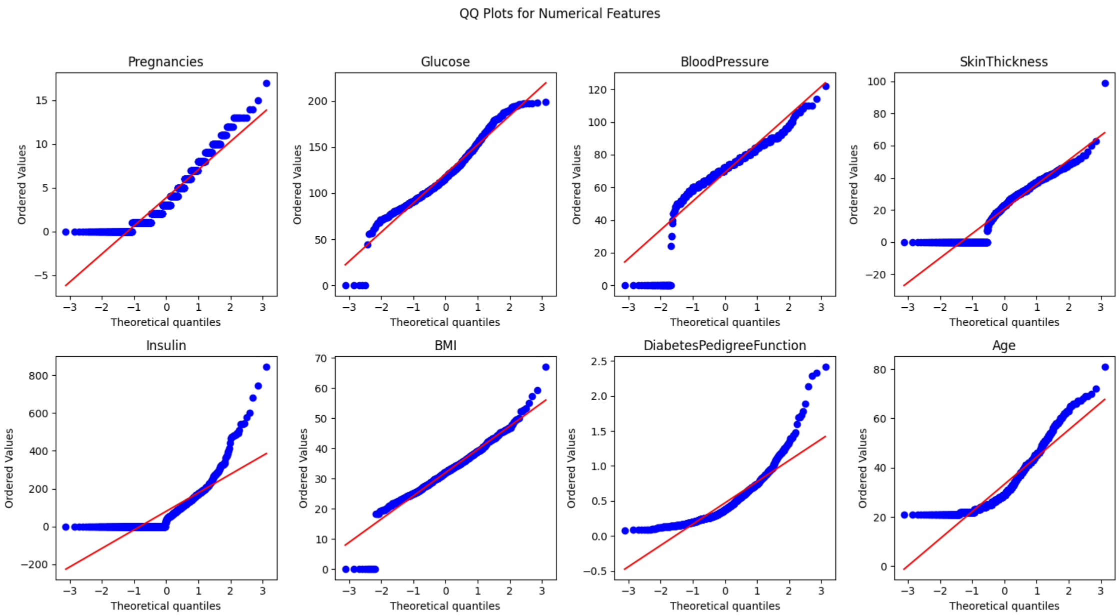
[7]:
```python
# Correlation matrix heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='plasma', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



[8]:
```python
# Define a colorful palette
colors = ["#FE7A36", "#3652AD"]

# Boxplots for each feature by Outcome
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 8))
fig.suptitle('Boxplots of Features by Outcome', y=1.02)

for i, column in enumerate(df.columns[:-1]):
    sns.boxplot(data=df, x='Outcome', y=column, ax=axes[i // 4, i % 4],
 ↪palette=colors)

plt.tight_layout()
plt.show()
```



[9]:
```python
# Q-Q plot
import statsmodels.api as sm
from scipy.stats import probplot

# QQ plot for each numerical feature
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 8))
fig.suptitle('QQ Plots for Numerical Features', y=1.02)

for i, column in enumerate(df.columns[:-1]):
    probplot(df[column], plot=axes[i // 4, i % 4])
    axes[i // 4, i % 4].set_title(column)

plt.tight_layout()
plt.show()
```



[10]:
```python
import numpy as np

# Calculate Z-scores for each column
z_scores = np.abs((df - df.mean()) / df.std())

# Define a threshold for outliers (e.g., Z-score greater than 3)
outlier_threshold = 3

# Identify outliers for each column
outliers = (z_scores > outlier_threshold).sum()

# Display the count of outliers for each column
print("Number of outliers for each column:")
print(outliers)
```

```
Number of outliers for each column:
Pregnancies                  4
Glucose                      5
BloodPressure               35
SkinThickness                1
Insulin                     18
BMI                         14
DiabetesPedigreeFunction    11
Age                          5
Outcome                      0
dtype: int64
```

[11]:
```
# Remove outliers using z-score or IQR method
from scipy.stats import zscore

z_scores = zscore(df)
df_no_outliers = df[(z_scores < 3).all(axis=1)]
```

[12]:
```
df_no_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 729 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               729 non-null    int64
 1   Glucose                   729 non-null    int64
 2   BloodPressure             729 non-null    int64
 3   SkinThickness             729 non-null    int64
 4   Insulin                   729 non-null    int64
 5   BMI                       729 non-null    float64
 6   DiabetesPedigreeFunction  729 non-null    float64
 7   Age                       729 non-null    int64
 8   Outcome                   729 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 57.0 KB
```

1.1.3 3. Model Fitting:

[13]:
```
X = df_no_outliers.drop('Outcome', axis=1)
y = df_no_outliers['Outcome']
```

[14]:
```
import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

```
WARNING:tensorflow:From
c:\Users\cgunj\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From
C:\Users\cgunj\AppData\Local\Temp\ipykernel_7088\292850708.py:2: The name
tf.logging.set_verbosity is deprecated. Please use
tf.compat.v1.logging.set_verbosity instead.
```

[15]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
from keras import models
from keras import layers


# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Data preprocessing using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building a simple neural network model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(X_train_scaled.
↪shape[1],)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))  # Assuming binary␣
↪classification

# Compiling the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Training the model
model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_split=0.
↪1)

# Evaluating the model
test_loss, test_acc = model.evaluate(X_test_scaled, y_test)
print('Test accuracy:', test_acc)
```

```
Epoch 1/10
17/17 [==============================] - 1s 10ms/step - loss: 0.6662 - accuracy:
0.6298 - val_loss: 0.6483 - val_accuracy: 0.6271
Epoch 2/10
17/17 [==============================] - 0s 2ms/step - loss: 0.5822 - accuracy:
0.7366 - val_loss: 0.6022 - val_accuracy: 0.6271
Epoch 3/10
17/17 [==============================] - 0s 2ms/step - loss: 0.5300 - accuracy:
0.7672 - val_loss: 0.5779 - val_accuracy: 0.6441
Epoch 4/10
17/17 [==============================] - 0s 2ms/step - loss: 0.4998 - accuracy:
0.7634 - val_loss: 0.5707 - val_accuracy: 0.6610
Epoch 5/10
```

```
17/17 [==============================] - 0s 2ms/step - loss: 0.4787 - accuracy:
0.7691 - val_loss: 0.5700 - val_accuracy: 0.6610
Epoch 6/10
17/17 [==============================] - 0s 2ms/step - loss: 0.4671 - accuracy:
0.7863 - val_loss: 0.5701 - val_accuracy: 0.6949
Epoch 7/10
17/17 [==============================] - 0s 2ms/step - loss: 0.4581 - accuracy:
0.7844 - val_loss: 0.5682 - val_accuracy: 0.6610
Epoch 8/10
17/17 [==============================] - 0s 2ms/step - loss: 0.4527 - accuracy:
0.7863 - val_loss: 0.5741 - val_accuracy: 0.6949
Epoch 9/10
17/17 [==============================] - 0s 2ms/step - loss: 0.4483 - accuracy:
0.7939 - val_loss: 0.5729 - val_accuracy: 0.6949
Epoch 10/10
17/17 [==============================] - 0s 2ms/step - loss: 0.4438 - accuracy:
0.7996 - val_loss: 0.5699 - val_accuracy: 0.6780
5/5 [==============================] - 0s 1ms/step - loss: 0.3849 - accuracy:
0.8425
Test accuracy: 0.8424657583236694
```

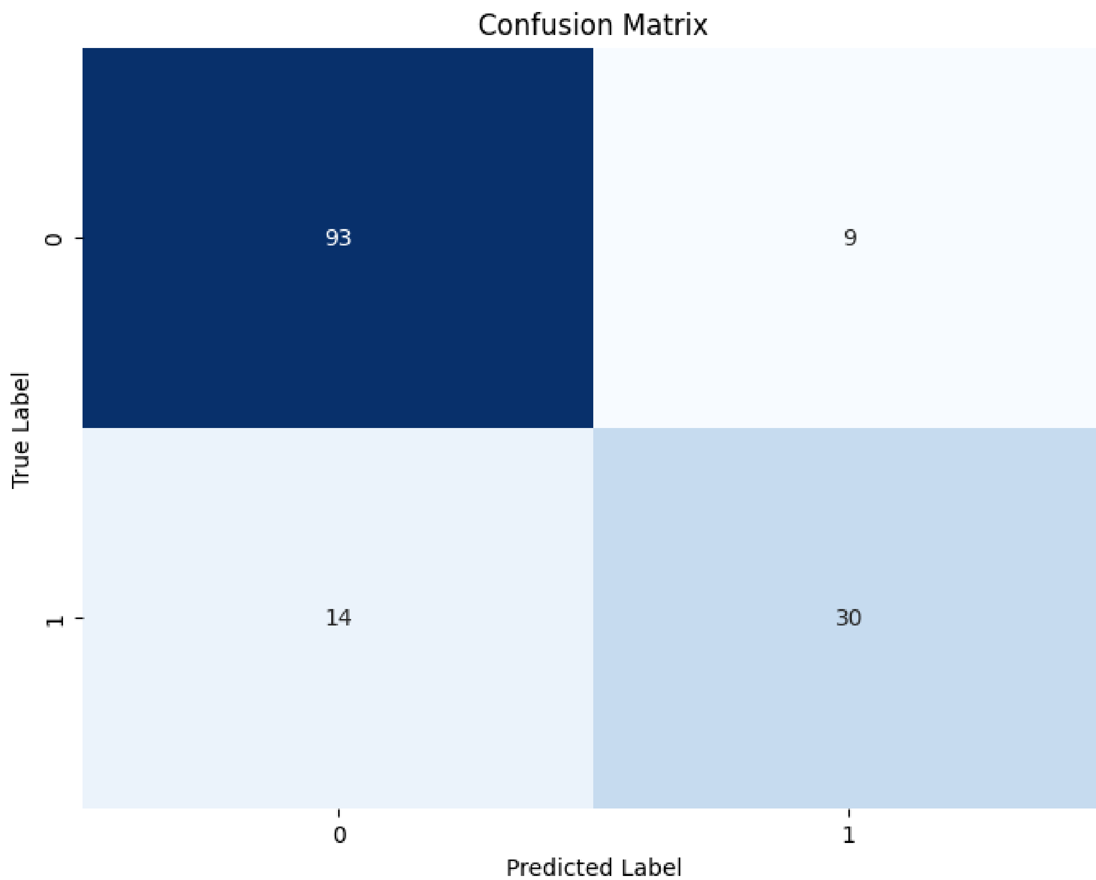
[16]:

```
from sklearn.metrics import confusion_matrix, roc_curve, auc

# Predict probabilities for the test set
y_pred_probs = model.predict(X_test_scaled)

# Convert probabilities to binary predictions
y_pred = (y_pred_probs > 0.5).astype(int)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

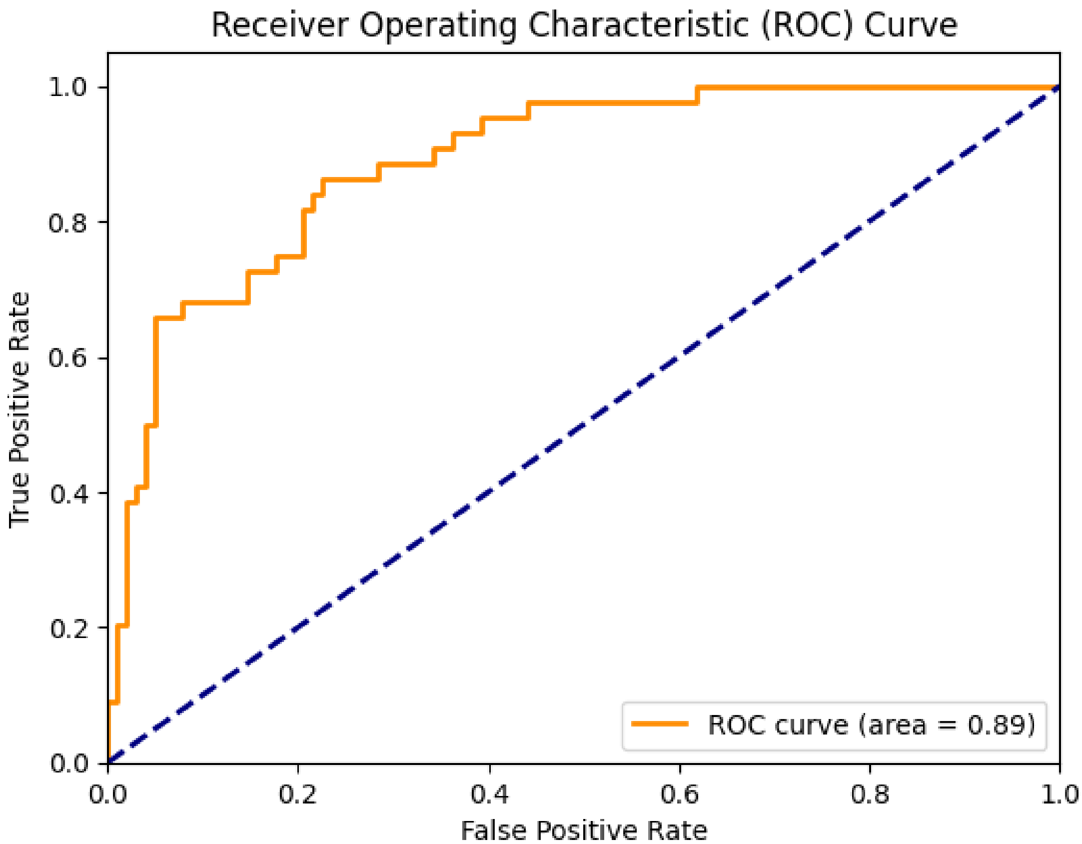
```
5/5 [==============================] - 0s 1ms/step
Confusion Matrix:
[[93  9]
 [14 30]]
```

[17]:

```
# Plot confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



[18]:
```
# ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



1.2 Regression

[19]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Data preprocessing using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building a simple regression model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(X_train_scaled.
↪shape[1],)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1))  # No activation function for regression

# Compiling the model
```

```
model.compile(optimizer='adam', loss='mean_squared_error')  # Using mean␣
↪squared error loss for regression

# Training the model
model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_split=0.
↪1)

# Evaluating the model
test_loss = model.evaluate(X_test_scaled, y_test)
print('Test loss:', test_loss)
```

```
Epoch 1/10
17/17 [==============================] - 1s 7ms/step - loss: 0.3272 - val_loss:
0.2566
Epoch 2/10
17/17 [==============================] - 0s 3ms/step - loss: 0.1944 - val_loss:
0.2263
Epoch 3/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1709 - val_loss:
0.2110
Epoch 4/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1617 - val_loss:
0.2137
Epoch 5/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1547 - val_loss:
0.2143
Epoch 6/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1514 - val_loss:
0.2182
Epoch 7/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1486 - val_loss:
0.2198
Epoch 8/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1465 - val_loss:
0.2198
Epoch 9/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1441 - val_loss:
0.2248
Epoch 10/10
17/17 [==============================] - 0s 2ms/step - loss: 0.1433 - val_loss:
0.2194
5/5 [==============================] - 0s 1ms/step - loss: 0.1287
Test loss: 0.12872956693172455
```

Interpretation of the results:

- **Training Loss (Epochs 1-10):** The training loss decreases steadily over the epochs, indicating that the model is improving its performance on the training data.

- **Validation Loss (Epochs 1-10):** The validation loss also decreases initially, indicating that the model generalizes well to unseen data. However, there is a slight increase in validation loss towards the end of training, which suggests that the model may be starting to overfit the training data slightly.
- **Test Loss:** The test loss (or evaluation loss) is calculated after training the model and evaluating it on the test set. In this case, the test loss is 0.128, which represents the mean squared error between the predicted and actual values on the test set. Lower test loss indicates better performance of the model on unseen data.

Overall, the model seems to perform reasonably well on the given regression task, with the test loss being relatively low. However, it's essential to monitor for overfitting, especially if the validation loss starts to increase significantly while the training loss continues to decrease. Regularization techniques or model adjustments may be necessary to address overfitting if observed.

[20]:
```
# Predictions on test set
y_pred = model.predict(X_test_scaled).flatten()  # Flatten the predictions to␣
↪make them 1D

# Calculate Mean Squared Error (MSE)
mse = np.mean((y_pred - y_test)**2)

print('Mean Squared Error (MSE):', mse)
```

```
5/5 [==============================] - 0s 1ms/step
Mean Squared Error (MSE): 0.12872956359812804
```

The Mean Squared Error (MSE) value of approximately 0.128 suggests that, on average, the squared difference between the predicted values and the actual values in the test set is 0.128.

Since the MSE is a measure of the average squared deviation of predictions from the actual values, a lower MSE indicates better performance of the regression model. In this case, the MSE value of 0.128 indicates that the model's predictions are relatively close to the actual values in the test set.