assignment-11-02-02-2024

February 4, 2024

1 Gunjan Chakraborty

1.1 22MSRDS007

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

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

1.1.1 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
```

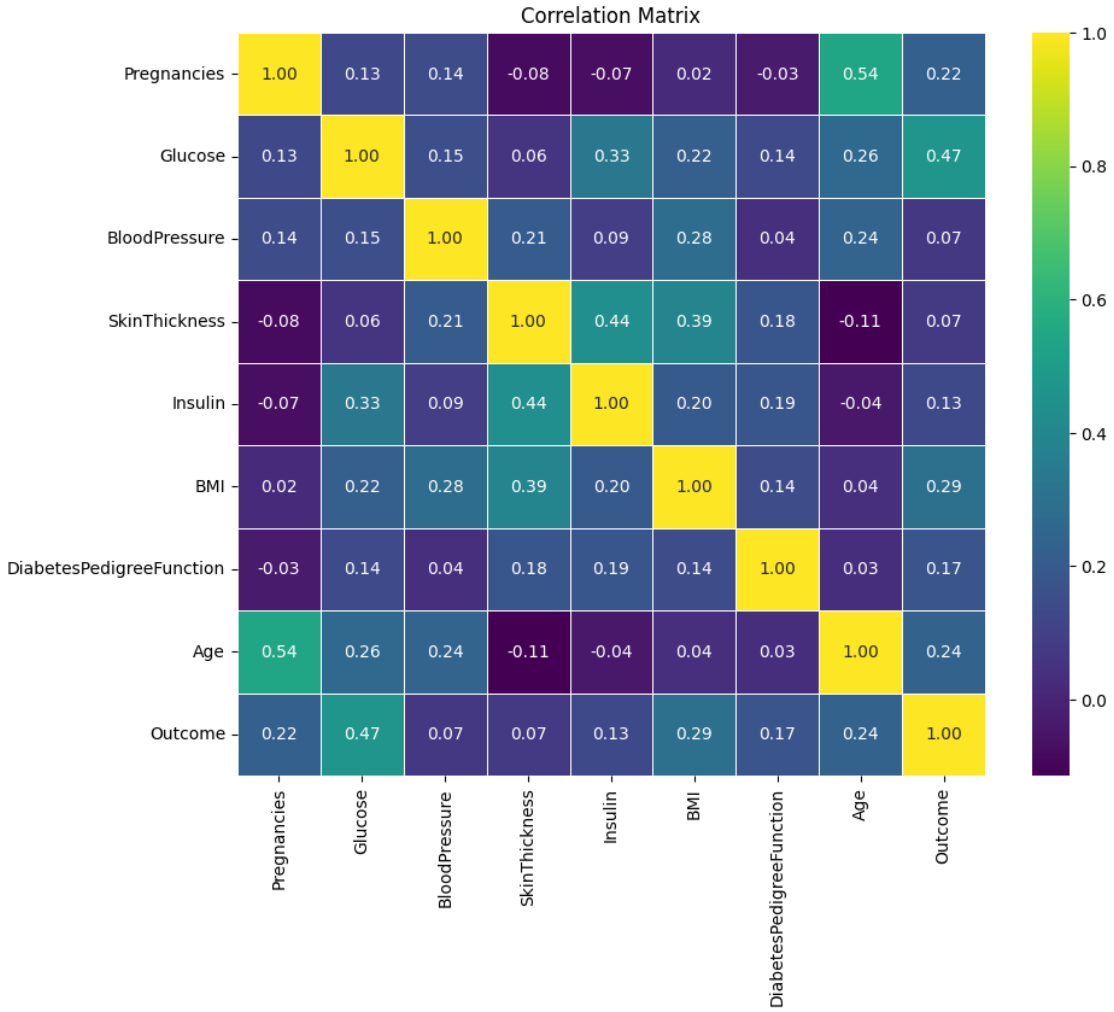
```
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       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]:
```
print(df.isnull().sum())
```

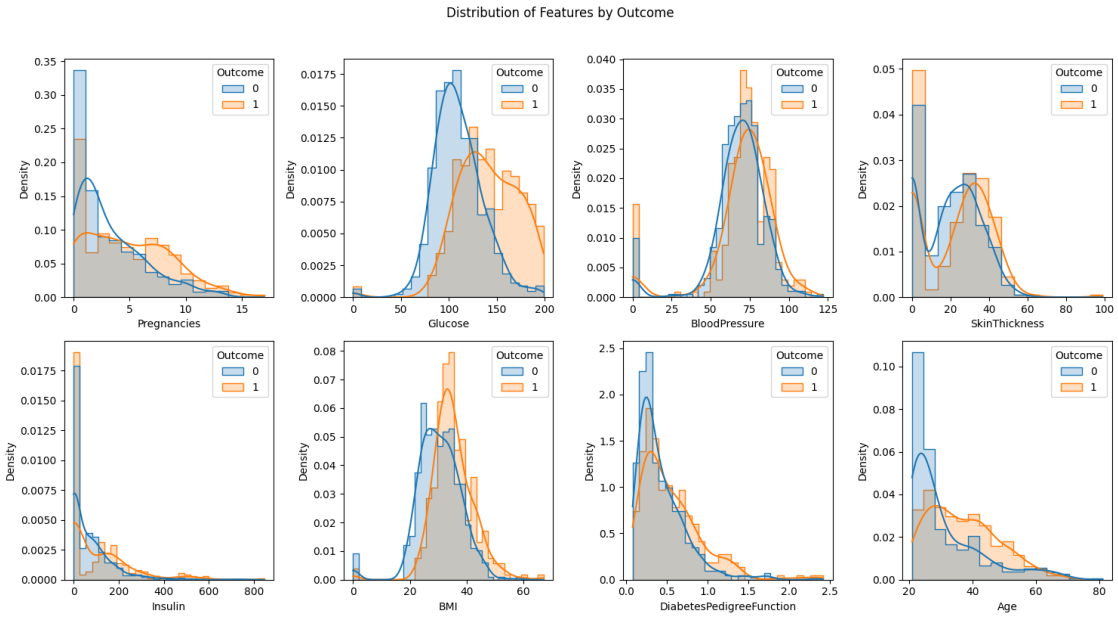
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[6]:
```
import seaborn as sns
import matplotlib.pyplot as plt

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

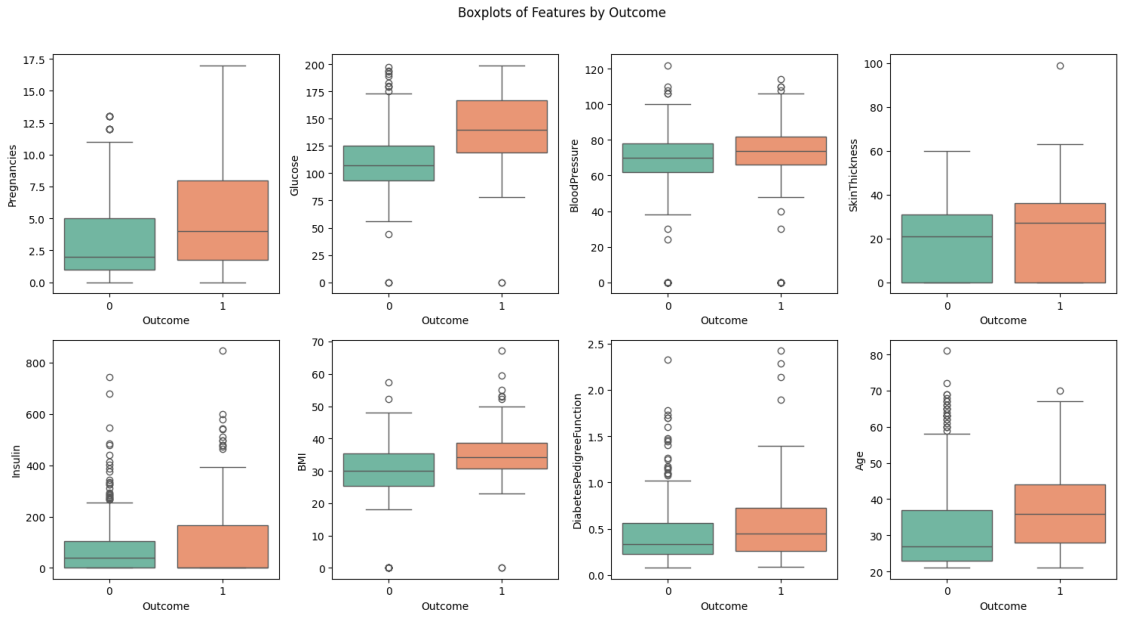
[7]:
```
# Correlation matrix heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt=".2f",␣
 ↪linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



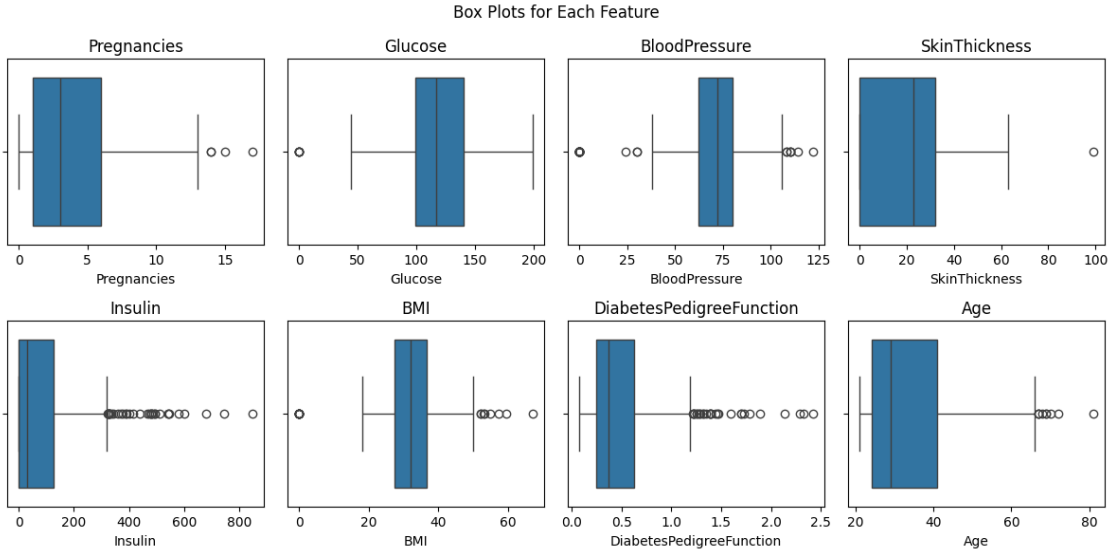
[8]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Distribution of each feature by Outcome with a unified histogram
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 8))
fig.suptitle('Distribution of Features by Outcome', y=1.02)

for i, column in enumerate(df.columns[:-1]):
    sns.histplot(data=df, x=column, hue='Outcome', kde=True, element="step",␣
 ↪stat="density", common_norm=False, ax=axes[i // 4, i % 4])

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



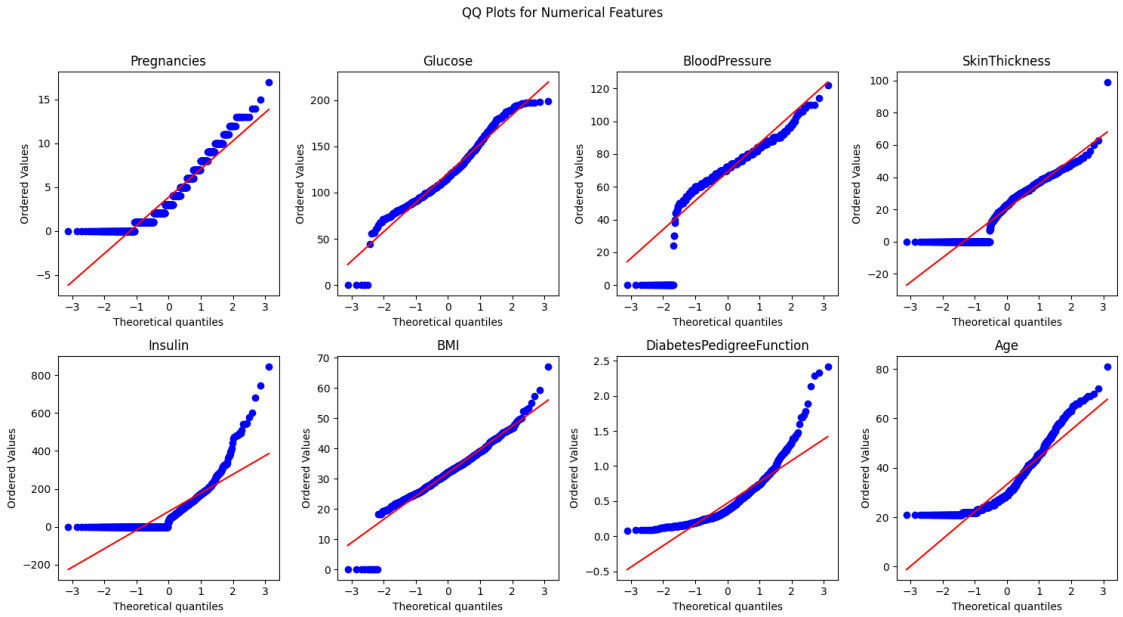
[9]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define a colorful palette
colors = ["#66c2a5", "#fc8d62"]

# 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()
```



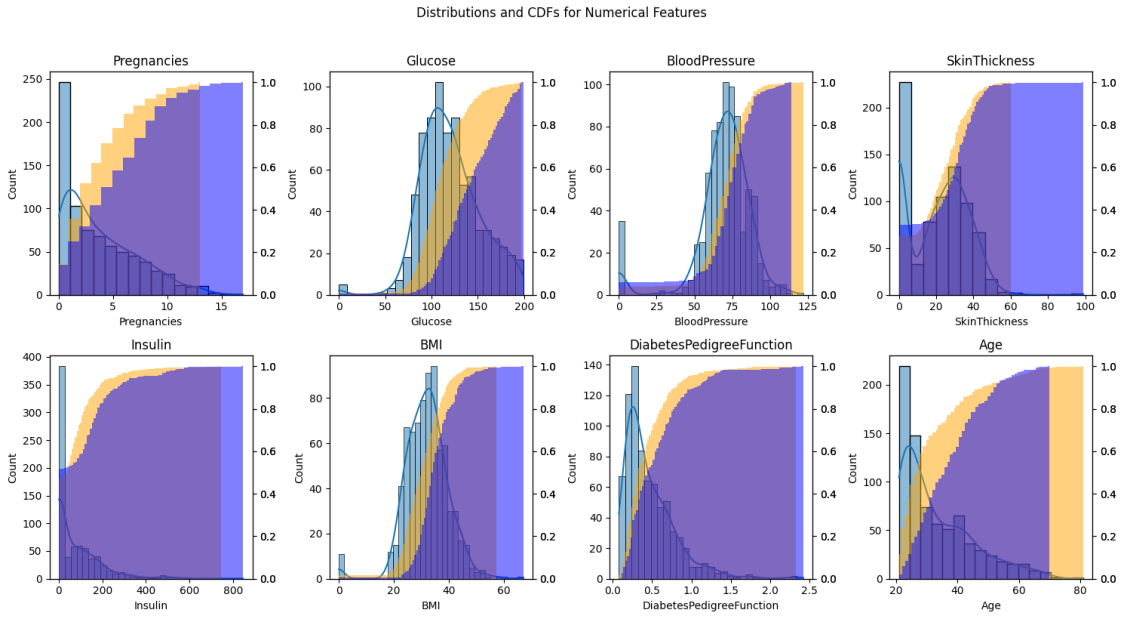
[10]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Select columns excluding 'Outcome'
columns_to_plot = df.iloc[:, :-1]

# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(12, 6))
fig.suptitle('Box Plots for Each Feature')

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Create box plots for each column
for i, column in enumerate(columns_to_plot.columns):
    sns.boxplot(x=columns_to_plot[column], ax=axes[i])
    axes[i].set_title(column)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



[11]:
```
# 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()
```



[12]:
```
# Individual feature distributions and CDFs
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15, 8))
fig.suptitle('Distributions and CDFs for Numerical Features', y=1.02)

for i, column in enumerate(df.columns[:-1]):
    sns.histplot(data=df, x=column, kde=True, ax=axes[i // 4, i % 4])
    axes[i // 4, i % 4].set_title(column)
    axes[i // 4, i % 4].twinx().hist(df[df['Outcome'] == 0][column],␣
 ↪cumulative=True, bins=100, color='orange', alpha=0.5, density=True)
    axes[i // 4, i % 4].twinx().hist(df[df['Outcome'] == 1][column],␣
 ↪cumulative=True, bins=100, color='blue', alpha=0.5, density=True)

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



[13]:
```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assuming X contains the predictor variables
VIF = df.drop('Outcome', axis=1)

# Calculate VIF for each predictor variable
vif_data = pd.DataFrame()
vif_data["Variable"] = VIF.columns
vif_data["VIF"] = [variance_inflation_factor(VIF.values, i) for i in range(VIF.
  ↪shape[1])]

# Display the VIF values
print(vif_data)
```

	Variable	VIF
0	Pregnancies	3.275748
1	Glucose	16.725078
2	BloodPressure	14.619512
3	SkinThickness	4.008696
4	Insulin	2.063689
5	BMI	18.408884
6	DiabetesPedigreeFunction	3.195626
7	Age	13.492985

1.2 Interpretation

1. **Glucose (VIF = 16.72):**
   - A VIF of 16.72 for 'Glucose' suggests a high level of multicollinearity with other variables in the dataset. This may indicate that 'Glucose' has a strong correlation with other predictor variables, potentially affecting the stability and reliability of the regression model.
2. **BloodPressure (VIF = 14.61):**
   - Similar to 'Glucose', a VIF of 14.61 for 'BloodPressure' indicates a high level of multicollinearity. This variable seems to have a strong correlation with other predictors.
3. **BMI (VIF = 18.40):**
   - 'BMI' has a very high VIF, suggesting a substantial degree of multicollinearity with other variables. This should be carefully considered, as it may affect the interpretation of the regression coefficients.
4. **Age (VIF = 13.49):**
   - 'Age' also has a high VIF, indicating multicollinearity. The presence of strong correlations between 'Age' and other predictors may impact the reliability of the regression model.

Variables with VIF values above a certain threshold (commonly 5-10) are considered to have problematic levels of multicollinearity. In this case, 'Glucose,' 'BloodPressure,' 'BMI,' and 'Age' exhibit high VIF values, suggesting potential issues.

To address multicollinearity, we may consider: - Removing one of the highly correlated variables. - Combining correlated variables into a composite feature. - Applying regularization techniques (e.g., Ridge or Lasso regression) that automatically penalize the influence of highly correlated predictors.

[14]:
```
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
```

[15]:
```
# 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)]
```

[16]:
```
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
```

1.2.1 3. Model Fitting:

[17]:
```
X = df_no_outliers.drop('Outcome', axis=1)
y = df_no_outliers['Outcome']
```

[18]:
```
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)
```

2 decision Tree

[19]:
```
from sklearn.tree import DecisionTreeClassifier

# Initialize the model
model = DecisionTreeClassifier()
```

```
# Fit the model
model.fit(X_train, y_train)
```

[19]:
```
DecisionTreeClassifier()
```

[20]:
```
model = DecisionTreeClassifier(max_depth=5, min_samples_split=2,␣
 ↪min_samples_leaf=1)
model.fit(X_train, y_train)
```

[20]:
```
DecisionTreeClassifier(max_depth=5)
```

[21]:
```
# Perform post-pruning using cost-complexity pruning
path = model.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Fit a series of models with different alpha values
models = []
for ccp_alpha in ccp_alphas:
    pruned_model = DecisionTreeClassifier(ccp_alpha=ccp_alpha)
    pruned_model.fit(X_train, y_train)
    models.append(pruned_model)

# Choose the model with the best alpha
best_model = models[np.argmax(ccp_alphas)]
```

[22]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Make predictions
y_pred = best_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Use zero_division=1 to set precision to 0 if there are no predicted samples
classification_rep = classification_report(y_test, y_pred, zero_division=1)

print(f"Accuracy(Decision Tree): {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{classification_rep}")
```

```
Accuracy(Decision Tree): 0.773972602739726
Confusion Matrix:
[[84 18]
 [15 29]]
Classification Report:
```

	precision	recall	f1-score	support
0	0.85	0.82	0.84	102
1	0.62	0.66	0.64	44
accuracy			0.77	146
macro avg	0.73	0.74	0.74	146
weighted avg	0.78	0.77	0.78	146

2.1 Random Forest

[23]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the Random Forest model
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy (Random Forest): {accuracy_rf}")
```

```
Accuracy (Random Forest): 0.7808219178082192
```

[24]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the Random Forest model with pre-pruning parameters
rf_pre_pruned_model = RandomForestClassifier(
    max_depth=5,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features='sqrt',
    random_state=42
)

# Fit the pre-pruned Random Forest model
rf_pre_pruned_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf_pre_pruned = rf_pre_pruned_model.predict(X_test)

# Evaluate the pre-pruned Random Forest model
accuracy_rf_pre_pruned = accuracy_score(y_test, y_pred_rf_pre_pruned)
```

```
print(f"Accuracy (Random Forest - Pre-Pruned): {accuracy_rf_pre_pruned}")
```

```
Accuracy (Random Forest - Pre-Pruned): 0.8082191780821918
```

[25]:
```
# Fit the model with the best hyperparameters
rf_model_tuned = RandomForestClassifier(
    n_estimators=120,
    max_depth=3,
    min_samples_split=4,
    min_samples_leaf=2,
    max_features='log2',
    random_state=42
)

rf_model_tuned.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf_tuned = rf_model_tuned.predict(X_test)

# Evaluate the tuned Random Forest model
accuracy_rf_tuned = accuracy_score(y_test, y_pred_rf_tuned)
print(f"Accuracy (Random Forest - Tuned): {accuracy_rf_tuned}")
```

```
Accuracy (Random Forest - Tuned): 0.821917808219178
```

2.1.1 Gradient Boosting

[26]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Assuming X contains the predictor variables and y contains the target variable
X = df_no_outliers.drop('Outcome', axis=1)
y = df_no_outliers['Outcome']

# Split 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)

# Initialize the Gradient Boosting model
gradient_boost_model = GradientBoostingClassifier(n_estimators=150,␣
 ↪learning_rate=0.01, random_state=42)

# Fit the model
gradient_boost_model.fit(X_train, y_train)
```

```python
# Make predictions on the test set
y_pred_gradient_boost = gradient_boost_model.predict(X_test)

# Evaluate the model
accuracy_gradient_boost = accuracy_score(y_test, y_pred_gradient_boost)
print(f"Accuracy (Gradient Boosting): {accuracy_gradient_boost:.4f}")

# Classification Report and Confusion Matrix
print("\nClassification Report:")
print(classification_report(y_test, y_pred_gradient_boost))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_gradient_boost))
```

```
Accuracy (Gradient Boosting): 0.8288

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89       102
           1       0.83      0.55      0.66        44

    accuracy                           0.83       146
   macro avg       0.83      0.75      0.77       146
weighted avg       0.83      0.83      0.82       146


Confusion Matrix:
[[97  5]
 [20 24]]
```

3 Conclusion

In conclusion, we evaluated the performance of three different machine learning models on the diabetes dataset:

1. **Gradient Boosting:**
   - Achieved an accuracy of 82.88%.
   - Demonstrated a relatively higher predictive accuracy compared to other models.
2. **Random Forest (Tuned):**
   - Achieved an accuracy of 82.19%.
   - Fine-tuning hyperparameters led to improved performance compared to the initial Random Forest model.
3. **Decision Tree:**
   - Achieved an accuracy of 77.40%.
   - Displayed lower accuracy compared to the Gradient Boosting and Tuned Random Forest models.

Overall, the Gradient Boosting model showed the highest accuracy, indicating its effectiveness in capturing complex relationships within the dataset. The fine-tuned Random Forest model also performed well, showcasing the importance of hyperparameter optimization. The Decision Tree, while providing reasonable accuracy, fell slightly behind the other models.