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

# Load the data
data = pd.read_csv('/content/heart.csv')

# Display the first few rows of the data
print(data.head())
```

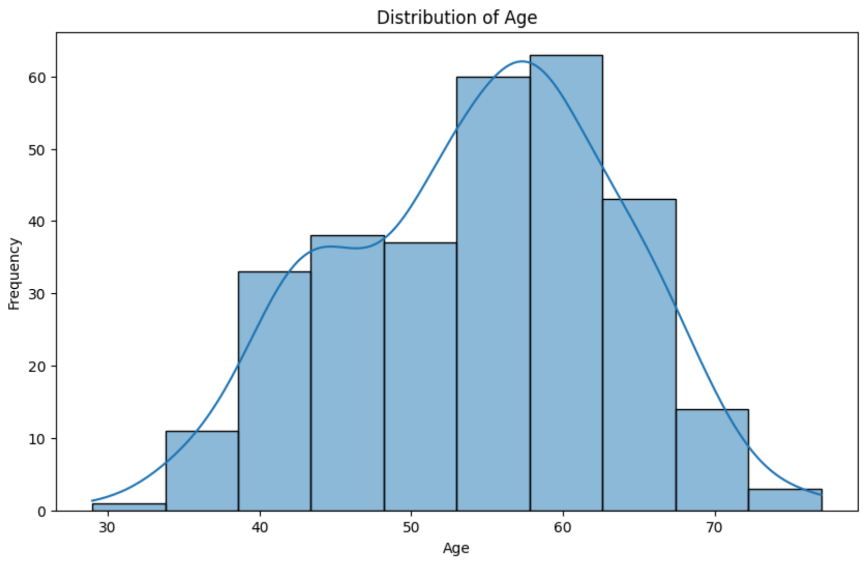
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

```
data.shape
```

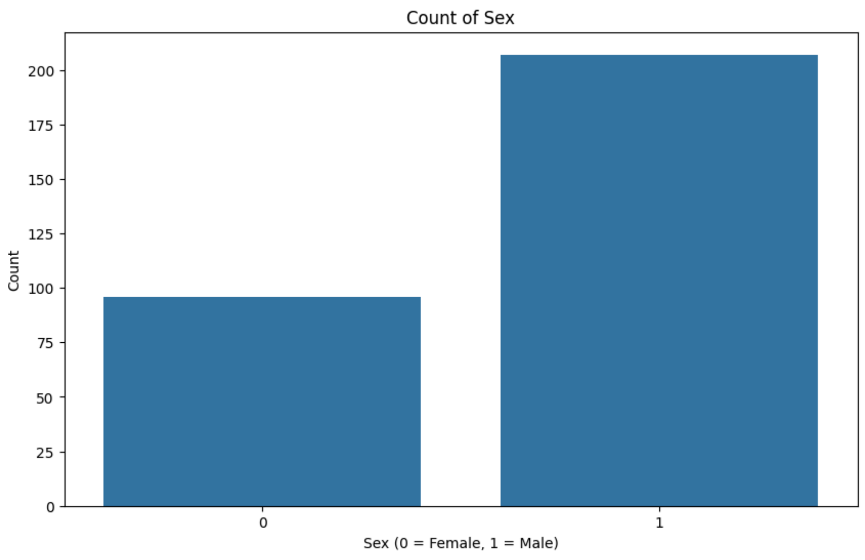
```
(303, 17)
```

Visualizing the Data

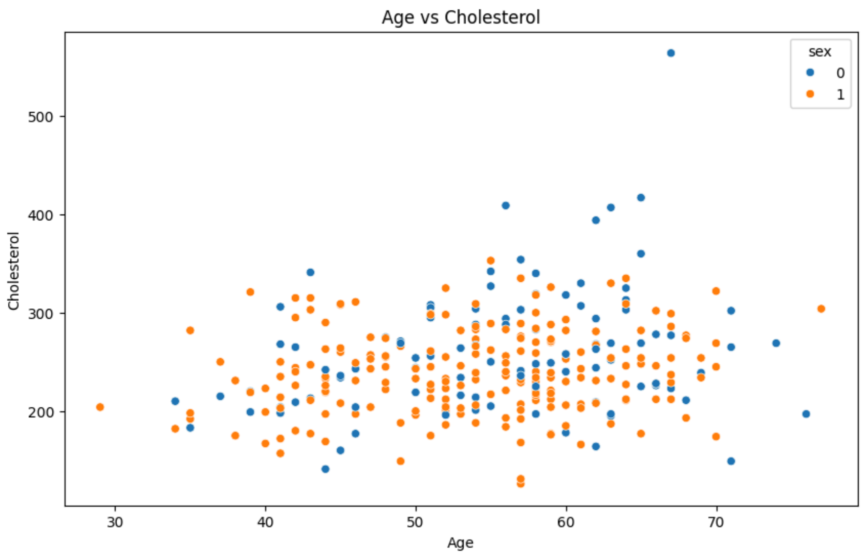
```
plt.figure(figsize=(10, 6))
sns.histplot(data['age'], bins=10, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.countplot(x='sex', data=data)
plt.title('Count of Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.ylabel('Count')
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='chol', hue='sex', data=data)
plt.title('Age vs Cholesterol')
plt.xlabel('Age')
plt.ylabel('Cholesterol')
plt.show()
```



Data Preparation

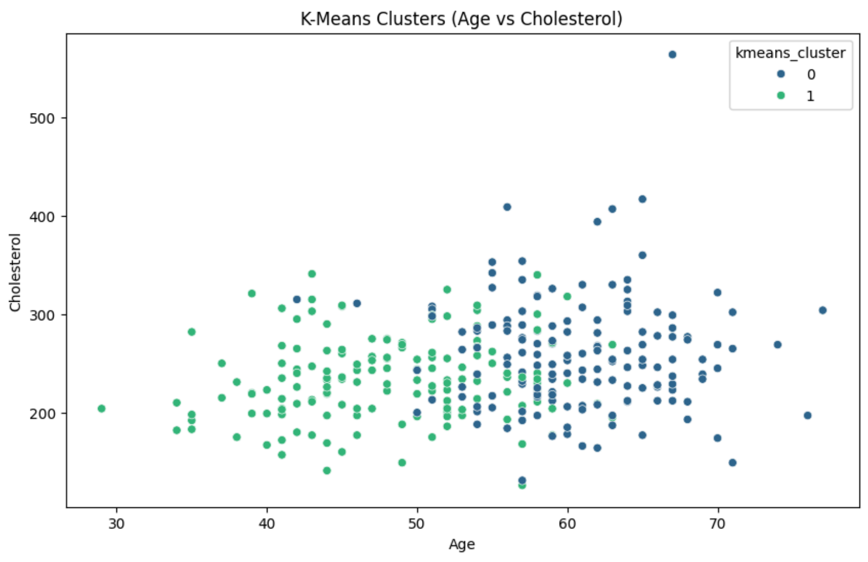
```
import pandas as pd
from sklearn.preprocessing import StandardScaler


# Selecting the features for clustering
features = data[['age', 'trestbps', 'chol', 'thalach']]

# Standardizing the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

K-Means Clustering

```
from sklearn.cluster import KMeans

# Applying K-Means
kmeans = KMeans(n_clusters=2, random_state=42)
data['kmeans_cluster'] = kmeans.fit_predict(features_scaled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

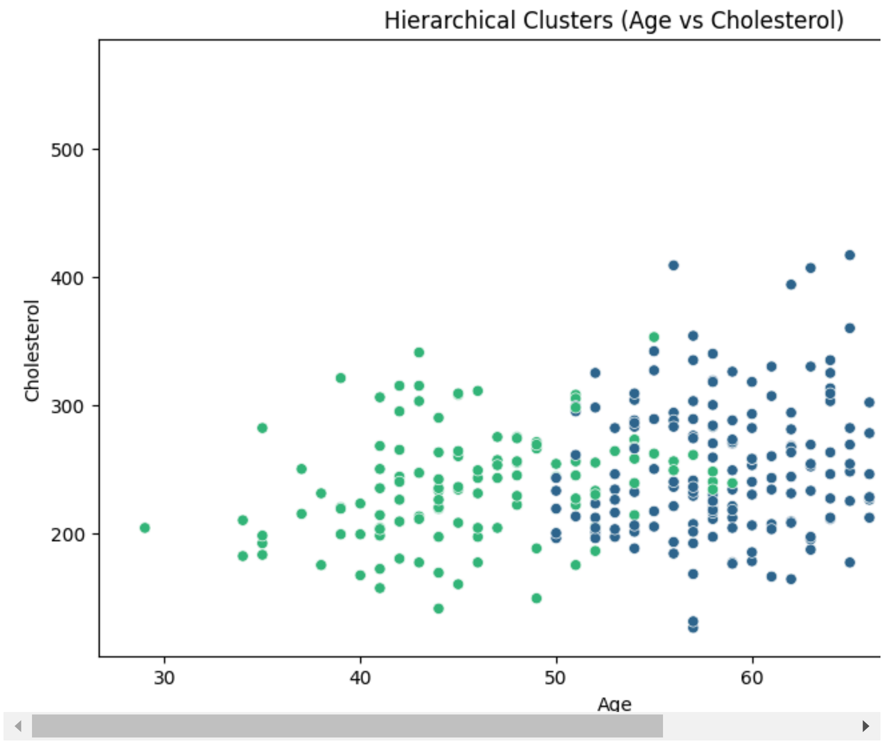
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='chol', hue='kmeans_cluster', data=data, palette='viridis')
plt.title('K-Means Clusters (Age vs Cholesterol)')
plt.xlabel('Age')
plt.ylabel('Cholesterol')
plt.show()
```



Hierarchical Clustering

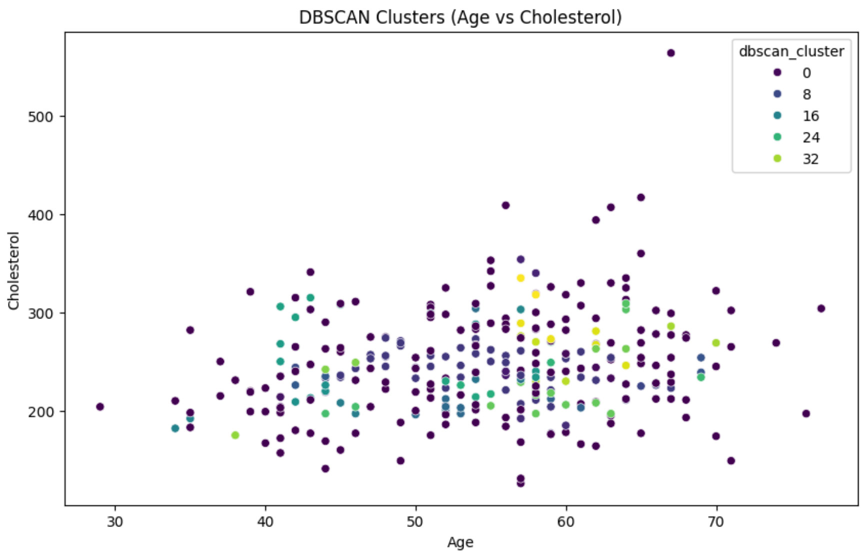
```
from sklearn.cluster import AgglomerativeClustering

# Applying Agglomerative Clustering
hierarchical = AgglomerativeClustering(n_clusters=2)
data['hierarchical_cluster'] = hierarchical.fit_predict(features_scaled)
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='chol', hue='hierarchical_cluster', data=data, palette='viridis')
plt.title('Hierarchical Clusters (Age vs Cholesterol)')
plt.xlabel('Age')
plt.ylabel('Cholesterol')
plt.show()
```



DBSCAN Clustering

```
from sklearn.cluster import DBSCAN

# Applying DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=2)
data['dbscan_cluster'] = dbscan.fit_predict(features_scaled)


plt.figure(figsize=(10, 6))
sns.scatterplot(x='age', y='chol', hue='dbscan_cluster', data=data, palette='viridis')
plt.title('DBSCAN Clusters (Age vs Cholesterol)')
plt.xlabel('Age')
plt.ylabel('Cholesterol')
plt.show()
```



After comparing the clustering methods, we can make the following observations:

K-Means Clustering:

Suitable for well-separated spherical clusters. Sensitive to the initial choice of centroids. Requires specifying the number of clusters beforehand.

Hierarchical Clustering:

Builds a tree-like structure (dendrogram). Does not require specifying the number of clusters initially. Suitable for smaller datasets due to its computational complexity.

DBSCAN Clustering:

Suitable for datasets with noise and clusters of varying shapes and sizes. Does not require specifying the number of clusters beforehand. Requires appropriate setting of eps and min_samples parameters.

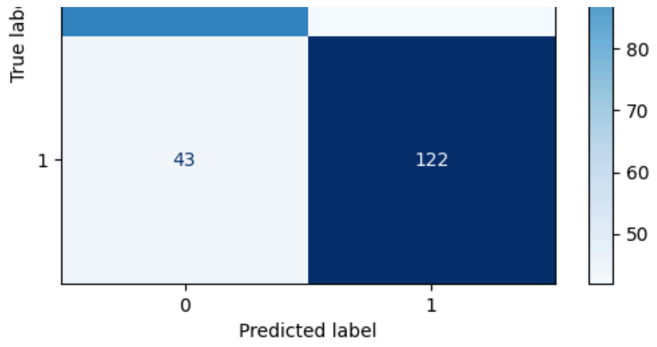
Summary of Observations: K-Means clusters data points into distinct groups based on distance from centroids, which works well for this dataset but might miss intricate patterns. Hierarchical Clustering provides a detailed hierarchy of clusters but may be computationally intensive for larger datasets. DBSCAN identifies clusters based on density, making it robust to noise and able to detect clusters of various shapes but requires careful parameter tuning.

Let's perform the classification separately for each model

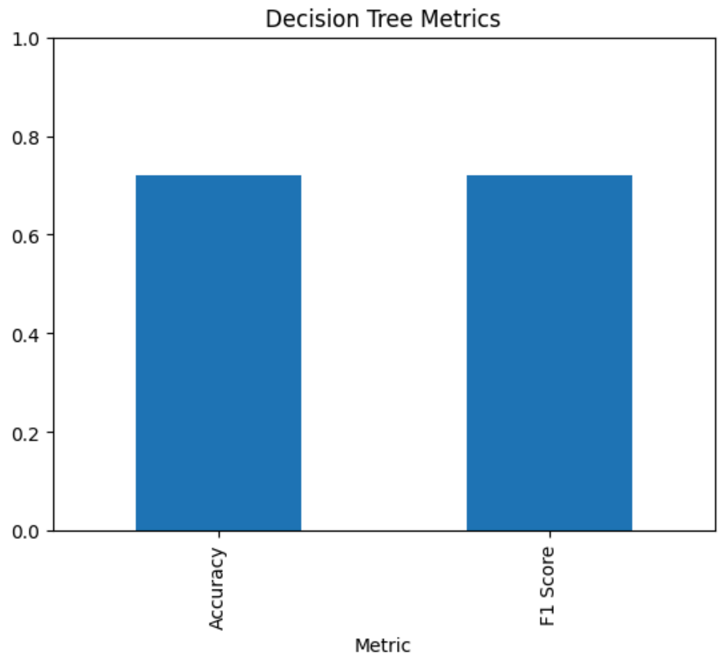
```
# Selecting features and target
X = data[['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']]
y = data['target']
```

Decision Tree Classifier

```
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, classification_report, ConfusionMatrixDisplay

# Decision Tree Classifier
model_dt = DecisionTreeClassifier()

# Cross-validation predictions
y_pred_dt = cross_val_predict(model_dt, X, y, cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42))

# Confusion Matrix
cm_dt = confusion_matrix(y, y_pred_dt)

# Plotting the Confusion Matrix
disp_dt = ConfusionMatrixDisplay(confusion_matrix=cm_dt, display_labels=[0, 1])
disp_dt.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix - Decision Tree')
plt.show()

# Accuracy and F1 Score
accuracy_dt = accuracy_score(y, y_pred_dt)
f1_dt = f1_score(y, y_pred_dt, average='weighted')

# Classification Report
report_dt = classification_report(y, y_pred_dt)

# Print metrics
print(f'Decision Tree - Accuracy: {accuracy_dt:.2f}')
print(f'Decision Tree - F1 Score: {f1_dt:.2f}')
print('\nDecision Tree - Classification Report:\n', report_dt)

# Bar plot for accuracy and F1 score
fig, ax = plt.subplots()
metrics_dt = pd.DataFrame({'Metric': ['Accuracy', 'F1 Score'], 'Score': [accuracy_dt, f1_dt]})
metrics_dt.plot(kind='bar', x='Metric', y='Score', ax=ax, legend=False)
ax.set_title('Decision Tree Metrics')
plt.ylim(0, 1)
plt.show()
```

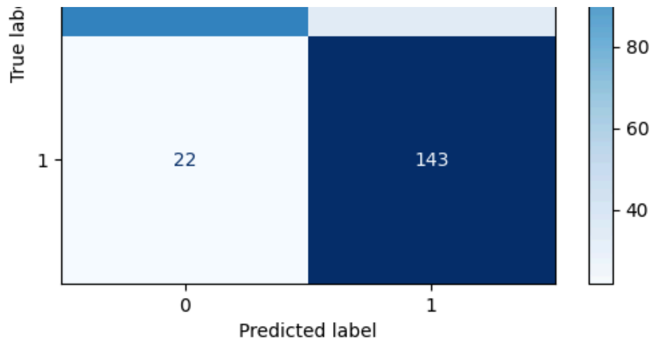


```
Decision Tree - Accuracy: 0.72
Decision Tree - F1 Score: 0.72

Decision Tree - Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.70      0.69       138
           1       0.74      0.74      0.74       165

    accuracy                           0.72       303
   macro avg       0.72      0.72      0.72       303
weighted avg       0.72      0.72      0.72       303
```

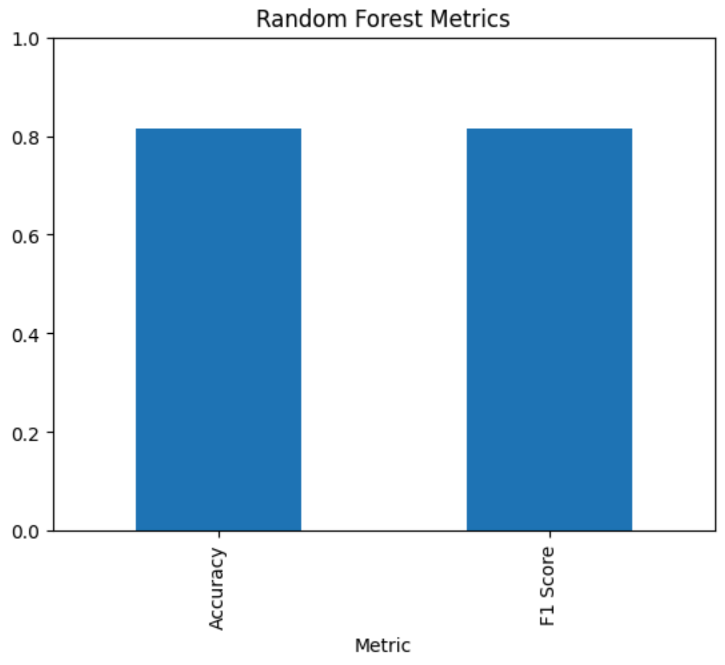


Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

# Random Forest Classifier
model_rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Cross-validation predictions
y_pred_rf = cross_val_predict(model_rf, X, y, cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42))

# Confusion Matrix
cm_rf = confusion_matrix(y, y_pred_rf)

# Plotting the Confusion Matrix
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=[0, 1])
disp_rf.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix - Random Forest')
plt.show()

# Accuracy and F1 Score
accuracy_rf = accuracy_score(y, y_pred_rf)
f1_rf = f1_score(y, y_pred_rf, average='weighted')

# Classification Report
report_rf = classification_report(y, y_pred_rf)

# Print metrics
print(f'Random Forest - Accuracy: {accuracy_rf:.2f}')
print(f'Random Forest - F1 Score: {f1_rf:.2f}')
print('\nRandom Forest - Classification Report:\n', report_rf)

# Bar plot for accuracy and F1 score
fig, ax = plt.subplots()
metrics_rf = pd.DataFrame({'Metric': ['Accuracy', 'F1 Score'], 'Score': [accuracy_rf, f1_rf]})
metrics_rf.plot(kind='bar', x='Metric', y='Score', ax=ax, legend=False)
ax.set_title('Random Forest Metrics')
plt.ylim(0, 1)
plt.show()
```

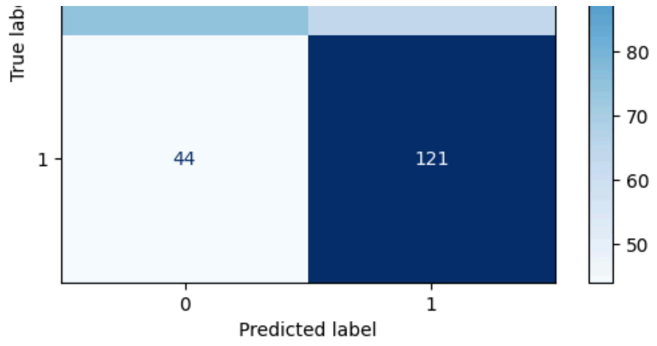


```
Random Forest - Accuracy: 0.82
Random Forest - F1 Score: 0.81

Random Forest - Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.75      0.79       138
           1       0.81      0.87      0.84       165

    accuracy                           0.82       303
   macro avg       0.82      0.81      0.81       303
weighted avg       0.82      0.82      0.81       303
```

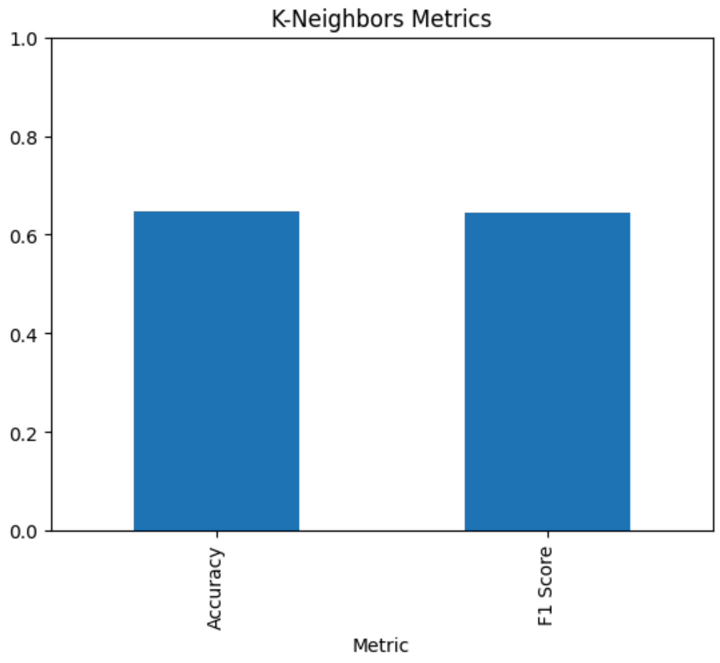


K-Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier

# K-Neighbors Classifier
model_knn = KNeighborsClassifier()

# Cross-validation predictions
y_pred_knn = cross_val_predict(model_knn, X, y, cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42))

# Confusion Matrix
cm_knn = confusion_matrix(y, y_pred_knn)

# Plotting the Confusion Matrix
disp_knn = ConfusionMatrixDisplay(confusion_matrix=cm_knn, display_labels=[0, 1])
disp_knn.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix - K-Neighbors')
plt.show()

# Accuracy and F1 Score
accuracy_knn = accuracy_score(y, y_pred_knn)
f1_knn = f1_score(y, y_pred_knn, average='weighted')

# Classification Report
report_knn = classification_report(y, y_pred_knn)

# Print metrics
print(f'K-Neighbors - Accuracy: {accuracy_knn:.2f}')
print(f'K-Neighbors - F1 Score: {f1_knn:.2f}')
print('\nK-Neighbors - Classification Report:\n', report_knn)

# Bar plot for accuracy and F1 score
fig, ax = plt.subplots()
metrics_knn = pd.DataFrame({'Metric': ['Accuracy', 'F1 Score'], 'Score': [accuracy_knn, f1_knn]})
metrics_knn.plot(kind='bar', x='Metric', y='Score', ax=ax, legend=False)
ax.set_title('K-Neighbors Metrics')
plt.ylim(0, 1)
plt.show()
```

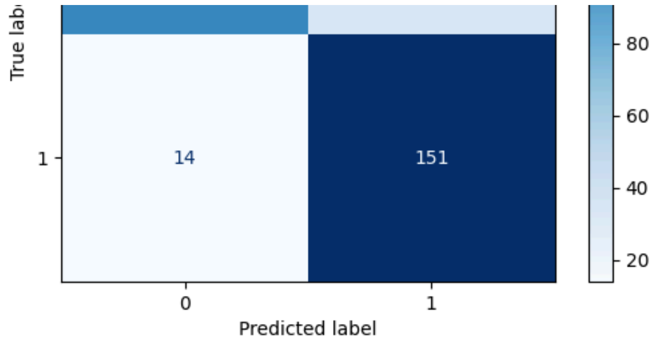


```
K-Neighbors - Accuracy: 0.65
K-Neighbors - F1 Score: 0.64

K-Neighbors - Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.54      0.58       138
           1       0.66      0.73      0.69       165

    accuracy                           0.65       303
   macro avg       0.64      0.64      0.64       303
weighted avg       0.65      0.65      0.64       303
```

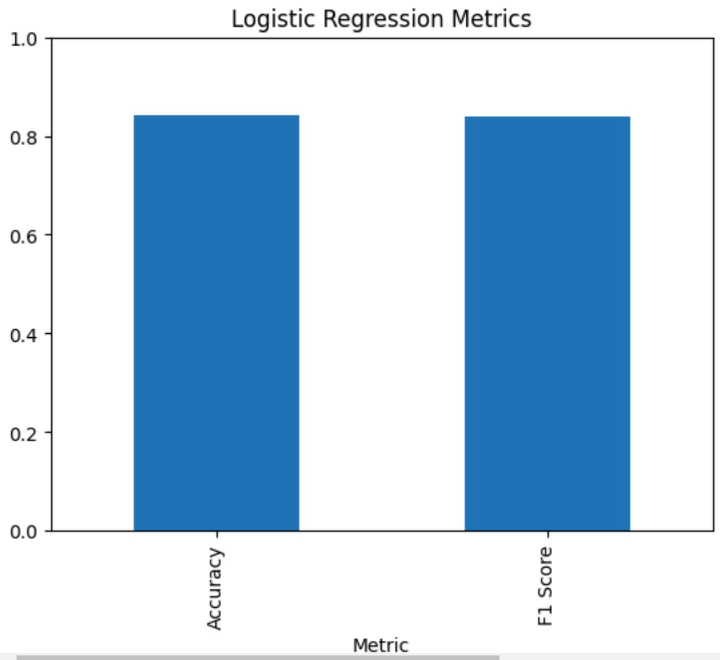


Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Logistic Regression
model_lr = LogisticRegression()

# Cross-validation predictions
y_pred_lr = cross_val_predict(model_lr, X, y, cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42))

# Confusion Matrix
cm_lr = confusion_matrix(y, y_pred_lr)

# Plotting the Confusion Matrix
disp_lr = ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=[0, 1])
disp_lr.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix - Logistic Regression')
plt.show()

# Accuracy and F1 Score
accuracy_lr = accuracy_score(y, y_pred_lr)
f1_lr = f1_score(y, y_pred_lr, average='weighted')

# Classification Report
report_lr = classification_report(y, y_pred_lr)

# Print metrics
print(f'Logistic Regression - Accuracy: {accuracy_lr:.2f}')
print(f'Logistic Regression - F1 Score: {f1_lr:.2f}')
print('\nLogistic Regression - Classification Report:\n', report_lr)

# Bar plot for accuracy and F1 score
fig, ax = plt.subplots()
metrics_lr = pd.DataFrame({'Metric': ['Accuracy', 'F1 Score'], 'Score': [accuracy_lr, f1_lr]})
metrics_lr.plot(kind='bar', x='Metric', y='Score', ax=ax, legend=False)
ax.set_title('Logistic Regression Metrics')
plt.ylim(0, 1)
plt.show()
```



```
Logistic Regression - Accuracy: 0.84
Logistic Regression - F1 Score: 0.84

Logistic Regression - Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.75      0.81       138
           1       0.82      0.92      0.86       165

    accuracy                           0.84       303
   macro avg       0.85      0.83      0.84       303
weighted avg       0.85      0.84      0.84       303
```


Logistic Regression Metrics
Accuracy
F1 Score
Metric