Certainly! Here are updated versions of the code snippets that include printing metrics, confusion matrices, and ROC curves for each technique:

Decision Tree for Classification:

python

Copy code

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc

import matplotlib.pyplot as plt

import pandas as pd


# Load data

data = pd.read_csv("your_data.csv")


# Preprocessing

# Fill null values if any

data.fillna(data.mean(), inplace=True)


# Split data into features and target

X = data.drop('target_column', axis=1)

y = data['target_column']


# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Build model

decision_tree = DecisionTreeClassifier()

decision_tree.fit(X_train, y_train)


# Predictions
```

```
y_pred = decision_tree.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Decision Tree Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

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

# ROC curve
y_probs = decision_tree.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_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('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

Support Vector Machine (SVM) for Classification:

python

Copy code

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import pandas as pd

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Split data into features and target
X = data.drop('target_column', axis=1)
y = data['target_column']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Predictions
y_pred = svm_classifier.predict(X_test)

# Evaluate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)
print("SVM Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

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

# ROC curve
y_probs = svm_classifier.decision_function(X_test)
fpr, tpr, _ = roc_curve(y_test, y_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('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

Linear Regression:

python

Copy code

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

import pandas as pd

import matplotlib.pyplot as plt


# Load data

data = pd.read_csv("your_data.csv")


# Preprocessing

# Fill null values if any

data.fillna(data.mean(), inplace=True)


# Split data into features and target

X = data.drop('target_column', axis=1)

y = data['target_column']


# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Build model

linear_regression = LinearRegression()

linear_regression.fit(X_train, y_train)


# Predictions

y_pred = linear_regression.predict(X_test)


# Calculate Mean Squared Error
```

```
mse = mean_squared_error(y_test, y_pred)
print("Linear Regression Mean Squared Error:", mse)

# Calculate R^2 score
r2 = r2_score(y_test, y_pred)
print("Linear Regression R^2 Score:", r2)

# Plot predicted vs actual
plt.scatter(y_test, y_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Linear Regression Predicted vs Actual")
plt.show()
```

Random Forest for Regression:

python

Copy code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Split data into features and target
X = data.drop('target_column', axis=1)
```

```
y = data['target_column']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)

# Predictions
y_pred = random_forest.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Random Forest Mean Squared Error:", mse)

# Plot predicted vs actual
plt.scatter(y_test, y_pred)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Random Forest Predicted vs Actual")
plt.show()
```

Hierarchical Clustering:

python

Copy code

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
```

```
# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Features
X = data.drop('target_column', axis=1)

# Build model
hierarchical_clustering = AgglomerativeClustering(n_clusters=3)
clusters = hierarchical_clustering.fit_predict(X)

# Evaluate Silhouette Score (higher is better)
silhouette_avg = silhouette_score(X, clusters)
print("Hierarchical Clustering Silhouette Score:", silhouette_avg)

# Plot dendrogram
plt.figure(figsize=(10, 7))
plt.title("Dendrogram")
dendrogram(hierarchical_clustering.children_)
plt.show()
```

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

python

Copy code

```
from sklearn.cluster import DBSCAN
import pandas as pd
import matplotlib.pyplot as plt

# Load data
```

```
data = pd.read_csv("your_data.csv")

# Preprocessing
# Fill null values if any
data.fillna(data.mean(), inplace=True)

# Features
X = data.drop('target_column', axis=1)

# Build model
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(X)

# Plot clusters
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=clusters, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('DBSCAN Clustering')
plt.show()
```

Logistic Regression for Classification:

python

Copy code

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt
import pandas as pd

# Load data
```

```
data = pd.read_csv("your_data.csv")

# Preprocessing, model building, and predictions are similar to the previous example

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

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

# ROC Curve
y_pred_proba = logistic_regression.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.plot(fpr, tpr, label='ROC Curve (AUC = %0.2f)' % roc_auc_score(y_test, y_pred_proba))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

Random Forest Regression:

python

Copy code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing, model building, and predictions are similar to the previous example

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Random Forest Mean Squared Error:", mse)

# R-squared (coefficient of determination)
r2 = r2_score(y_test, y_pred)
print("Random Forest R-squared (R2):", r2)
```

K-Means Clustering:

python

Copy code

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Load data
data = pd.read_csv("your_data.csv")

# Preprocessing and model building are similar to the previous example

# Evaluate Silhouette Score (higher is better)
```

```
silhouette_avg = silhouette_score(X, clusters)

print("K-Means Silhouette Score:", silhouette_avg)


# Visualize Clusters (optional)

plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=clusters, cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('K-Means Clustering')

plt.show()
```