```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean, cityblock

# Load dataset
data = pd.read_csv(r"C:\Users\gowri\Desktop\Iris.csv")  # Replace with your actual
dataset

data_numeric = data.select_dtypes(include=[np.number])  # Numeric columns only

data_cleaned = data.dropna()  # Remove rows with missing values

# Replace missing values with mean (for numeric columns)
data_filled = data.copy()
numeric_cols = data.select_dtypes(include=[np.number]).columns
data_filled[numeric_cols] =
data_filled[numeric_cols].fillna(data_filled[numeric_cols].mean())

# Normalization
scaler = StandardScaler()
data_standardized = pd.DataFrame(scaler.fit_transform(data_filled[numeric_cols]),
columns=numeric_cols)

# Correlation and Covariance
correlation_matrix = data_standardized.corr()
print("Corrlation Matrix:")
print(correlation_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()

# ----------------- Covariance -----------------
# Calculate covariance matrix
covariance_matrix = data_standardized.cov()

# Display covariance matrix
print("Covariance Matrix:")
print(covariance_matrix)

# Visualizing Covariance Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(covariance_matrix, annot=True, cmap='coolwarm')
```

```
plt.title("Covariance Matrix")
plt.show()

# Cosine Similarity
cosine_sim = cosine_similarity(data_standardized.iloc[:2, :])
print("Cosine Similarity:\n", cosine_sim)

# Proximal Analysis
print("Euclidean Distance:", euclidean(data_standardized.iloc[0],
data_standardized.iloc[1]))
print("Manhattan Distance:", cityblock(data_standardized.iloc[0],
data_standardized.iloc[1]))
print("Supremum:", np.max(data_numeric.to_numpy()))

# KNN Classification
X = data_standardized.iloc[:, :-1]
y = data_filled.iloc[:, -1]

if not pd.api.types.is_numeric_dtype(y):
    y = pd.factorize(y)[0]

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

# Find the best k using cross-validation
k_values = range(1, 21)
scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    score = cross_val_score(knn, X_train, y_train, cv=5).mean()
    scores.append(score)

best_k = k_values[np.argmax(scores)]  # Get k with the highest accuracy
print("Best k:", best_k)

# Train KNN model with best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)

y_pred = knn_best.predict(X_test)
print("KNN Accuracy with best k:", knn_best.score(X_test, y_test))

# Plot accuracy vs k
plt.plot(k_values, scores, marker='o')
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("Cross-Validated Accuracy")
plt.title("Choosing the Best k")
plt.show()
```

```
#OUTLIERS
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
# Outlier detection
data['IF'] = IsolationForest(contamination=0.1,
random_state=42).fit_predict(data_standardized)
data['EE'] = EllipticEnvelope(contamination=0.09,
random_state=42).fit_predict(data_standardized)

# Cook's Distance
X = sm.add_constant(data_standardized)
y = np.random.rand(len(data))  # Placeholder target
influence = sm.OLS(y, X).fit().get_influence()
data['CC'] = influence.cooks_distance[0]

# Heatmap
sns.heatmap(data[['IF', 'EE', 'CC']], cmap='coolwarm', cbar=True)
plt.show()

# Summary
print("Outliers (IF):", (data['IF'] == -1).sum())
print("Outliers (EE):", (data['EE'] == -1).sum())
print("Top 5 Cook's Distance:\n", data.nlargest(5, 'CC'))


#train_test_split
from sklearn.model_selection import train_test_split

def split_data(X, y, test_size=0.2, random_state=42):
    return train_test_split(X, y, test_size=test_size, random_state=random_state)


#kmeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate synthetic 2D data
X, y = make_blobs(n_samples=300, centers=3, cluster_std=0.6, random_state=42)

# Fit KMeans
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_
```

```
# Display Results
print(f"\n Number of Clusters: {n_clusters}")
print("Cluster Labels:\n", clusters)
print("\n Cluster Centroids:\n", centroids)

# Plot Clusters and Centroids
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=clusters, palette='viridis', s=100)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X',
label='Centroids')
plt.title("KMeans Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()



#naive bayes classification
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate synthetic 2D data
X, y = make_blobs(n_samples=300, centers=3, cluster_std=0.6, random_state=42)

# Fit KMeans
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_

# Display Results
print(f"\n Number of Clusters: {n_clusters}")
print("Cluster Labels:\n", clusters)
print("\n Cluster Centroids:\n", centroids)

# Plot Clusters and Centroids
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=clusters, palette='viridis', s=100)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X',
label='Centroids')
plt.title("KMeans Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

```
#naive bayes regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import make_regression

# Generate a regression dataset (you can replace this with your own data)
X, y = make_regression(n_samples=200, n_features=1, noise=0.1, random_state=42)

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

# Initialize Gaussian Naive Bayes (Note: it is a classification model, so we'll
have to adapt it)
model = GaussianNB()

# Since GaussianNB is a classifier, we need to treat the regression task as
classification
# Binning continuous target values into discrete classes
y_train_binned = np.digitize(y_train, bins=np.linspace(np.min(y_train),
np.max(y_train), 10))  # Binning the target variable
y_test_binned = np.digitize(y_test, bins=np.linspace(np.min(y_train),
np.max(y_train), 10))

# Train the model
model.fit(X_train, y_train_binned)

# Predict the binned classes
y_pred_binned = model.predict(X_test)

# Map the predicted classes back to continuous values by averaging the bins
bin_centers = (np.linspace(np.min(y_train), np.max(y_train), 10)[:-1] +
np.linspace(np.min(y_train), np.max(y_train), 10)[1:]) / 2
y_pred_continuous = bin_centers[y_pred_binned - 1]

# Evaluate the model
mse = mean_squared_error(y_test, y_pred_continuous)
r2 = r2_score(y_test, y_pred_continuous)

# Display results
print(f"✅ Naive Bayes Regression MSE: {mse:.2f}")
print(f"✅ Naive Bayes Regression R²: {r2:.2f}")

# Plot the predicted vs actual values
plt.scatter(y_test, y_pred_continuous)
```

```
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('Naive Bayes Regression - Predictions vs Actual Values')
plt.show()



#SVM
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

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

# SVM
svm_model = SVC(kernel='rbf')
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
acc_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {acc_svm:.2f}")

# Confusion Matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)
ConfusionMatrixDisplay(cm_svm,
display_labels=iris.target_names).plot(cmap="Purples")
plt.title("SVM - Confusion Matrix")
plt.show()


#LR
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error

# Load regression dataset
housing = fetch_california_housing()
X, y = housing.data, housing.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=42)

# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
preds_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, preds_lr)
print(f"Linear Regression MSE: {mse_lr:.2f}")

import matplotlib.pyplot as plt
import numpy as np

# Plot Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, preds_lr, alpha=0.5, color='blue', edgecolors='k')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--', linewidth=2)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Linear Regression: Actual vs Predicted")
plt.grid(True)
plt.show()



#CATBOOST AND GRADIENT
from catboost import CatBoostClassifier

# Load classification dataset
iris = load_iris()
X = iris.data
y = iris.target

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

# CatBoost
cat_model = CatBoostClassifier(verbose=0)
cat_model.fit(X_train, y_train)
y_pred_cat = cat_model.predict(X_test)
acc_cat = accuracy_score(y_test, y_pred_cat)
print(f"CatBoost Accuracy: {acc_cat:.2f}")

# Confusion Matrix
cm_cat = confusion_matrix(y_test, y_pred_cat)
ConfusionMatrixDisplay(cm_cat,
display_labels=iris.target_names).plot(cmap="Oranges")
plt.title("CatBoost - Confusion Matrix")
plt.show()
```

```
#DECISION TREE CLASSI
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Train the model
dtc_model = DecisionTreeClassifier()
dtc_model.fit(X_train, y_train)

# Predict
y_pred_dtc = dtc_model.predict(X_test)

# Accuracy
acc_dtc = accuracy_score(y_test, y_pred_dtc)
print("Decision Tree Classifier Accuracy:", acc_dtc)

# Confusion Matrix
cm_dtc = confusion_matrix(y_test, y_pred_dtc)
disp = ConfusionMatrixDisplay(confusion_matrix=cm_dtc)
disp.plot(cmap='Blues')
plt.title("Decision Tree - Confusion Matrix")
plt.show()


#DECISION TREE REGRE
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Train the model
dtr_model = DecisionTreeRegressor()
dtr_model.fit(X_train, y_train)

# Predict
y_pred_dtr = dtr_model.predict(X_test)

# Mean Squared Error
mse_dtr = mean_squared_error(y_test, y_pred_dtr)
print("Decision Tree Regressor MSE:", mse_dtr)

# Actual vs Predicted plot
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_dtr, alpha=0.5, color='green', edgecolors='k')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--', linewidth=2)
plt.xlabel("Actual Values")
```

```
plt.ylabel("Predicted Values")
plt.title("Decision Tree Regressor: Actual vs Predicted")
plt.grid(True)
plt.show()


#KMEDOIDS
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Create and fit the model
agg_model = AgglomerativeClustering(n_clusters=3, linkage='ward')  # 'affinity'
deprecated in newer versions
labels_agg = agg_model.fit_predict(X)

# Show cluster labels
print("Agglomerative Clustering Labels:", labels_agg)

# Visualize the clusters (assuming 2D or reduced 2D features)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_agg, palette='Set2', s=100,
edgecolor='k')
plt.title("Agglomerative Clustering Results")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.legend(title="Cluster")
plt.show()

#DBSCAN

from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN

# Apply PCA to reduce to 2 components for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Run DBSCAN
model = DBSCAN(eps=0.5, min_samples=5)
labels = model.fit_predict(X)

# Prepare DataFrame with PCA components
df = pd.DataFrame(X_pca, columns=['PCA1', 'PCA2'])
df['Cluster'] = labels
```

```python
# Plotting
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='PCA1', y='PCA2', hue='Cluster', palette='tab10')
plt.title("DBSCAN Clustering (PCA-reduced)")
plt.grid(True)
plt.show()
```