```
# EDA HISTOGRAM – CALIFORNIA HOUSE DATASET
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Load dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Display the first few rows
print(df.head())

# Display the shape and info of the dataset
print(df.shape)
print(df.info())

# Display the number of unique values for each column
print(df.nunique())

# Check for missing values
print(df.isnull().sum())

# Check for duplicates
print(df.duplicated().sum())
```

```
# Handle missing values by filling with the median (if any column has missing values)
df.fillna(df.median(), inplace=True)

# Display summary statistics
print(df.describe().T)

# Select only numerical columns
numerical_cols = df.select_dtypes(include=[np.number]).columns
print("Numerical columns:", numerical_cols)

# Plot histograms for each numerical column
for col in numerical_cols:
   plt.figure(figsize=(10, 6))
   df[col].plot(kind='hist', title=col, bins=60, edgecolor='black')
   plt.ylabel('Frequency')
   plt.show()

# Plot boxplots for each numerical column
for col in numerical_cols:
   plt.figure(figsize=(6, 6))
   sns.boxplot(x=df[col], color='blue')
   plt.title(col)
   plt.ylabel(col)
   plt.show()

# Compute and visualize the correlation matrix
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
```

```python
plt.show()

# Create a pair plot to visualize pairwise relationships
sns.pairplot(df)
plt.show()
```

```
# EDA HEATMAP-CALIFORNIA HOUSE DATASET

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing

# Load California Housing dataset
data = fetch_california_housing()

# Convert to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target  # Adding the target variable (median house value)

# Table of Meaning of Each Variable
variable_meaning = {
    "MedInc": "Median income in block group",
    "HouseAge": "Median house age in block group",
    "AveRooms": "Average number of rooms per household",
    "AveBedrms": "Average number of bedrooms per household",
    "Population": "Population of block group",
    "AveOccup": "Average number of household members",
    "Latitude": "Latitude of block group",
    "Longitude": "Longitude of block group",
    "Target": "Median house value (in $100,000s)"
}

variable_df = pd.DataFrame(list(variable_meaning.items()), columns=["Feature",
"Description"])
```

```
# Display the meaning of variables
print("\nVariable Meaning Table:")
print(variable_df)

# Basic Data Exploration
print("\nBasic Information about Dataset:")
print(df.info())  # Overview of dataset
print("\nFirst Five Rows of Dataset:")
print(df.head())  # Display first few rows

# Check for missing values
print("\nMissing Values in Each Column:")
print(df.isnull().sum())  # Count of missing values

# Histograms for distribution of features
plt.figure(figsize=(12, 8))
df.hist(bins=30, edgecolor='black')
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()

# Boxplots for outlier detection
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.title("Boxplots of Features to Identify Outliers")
plt.show()

# Correlation Matrix Heatmap
plt.figure(figsize=(10, 6))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
```

```python
plt.title("Feature Correlation Heatmap")
plt.show()

# Pairplot for feature relationships (only a subset for clarity)
sns.pairplot(df[['MedInc', 'HouseAge', 'AveRooms', 'Target']], diag_kind='kde')
plt.show()

# Key Insights
print("\nKey Insights:")
print("1. The dataset has", df.shape[0], "rows and", df.shape[1], "columns.")
print("2. No missing values were found in the dataset.")
print("3. Histograms show skewed distributions in some features like 'MedInc'.")
print("4. Boxplots indicate potential outliers in 'AveRooms' and 'AveOccup'.")
print("5. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.")
```

PCA- IRIS DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Step 1: Load the Iris Dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Step 2: Standardizing the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Calculating Covariance Matrix and Eigenvalues/Eigenvectors
cov_matrix = np.cov(X_scaled.T)
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)

# Step 4: Visualizing Data in 3D before PCA
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
colors = ['red', 'green', 'blue']
labels = iris.target_names
for i in range(len(colors)):
```

```
 ax.scatter(X_scaled[y == i, 0], X_scaled[y == i, 1], X_scaled[y == i, 2], color=colors[i],
label=labels[i])
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('3D Visualization of Iris Data Before PCA')
plt.legend()
plt.show()

# Step 5: Applying PCA using SVD (Singular Value Decomposition)
U, S, Vt = np.linalg.svd(X_scaled, full_matrices=False)
print("Singular Values:", S)

# Step 6: Applying PCA to Reduce Dimensionality to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 7: Understanding Variance Explained
explained_variance = pca.explained_variance_ratio_
print(f"Explained Variance by PC1: {explained_variance[0]:.2f}")
print(f"Explained Variance by PC2: {explained_variance[1]:.2f}")

# Step 8: Visualizing the Transformed Data
plt.figure(figsize=(8, 6))
for i in range(len(colors)):
 plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], color=colors[i], label=labels[i])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Iris Dataset (Dimensionality Reduction)')
plt.legend()
plt.grid()
```

```
plt.show()

# Step 9: Visualizing Eigenvectors Superimposed on 3D Data
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
for i in range(len(colors)):
 ax.scatter(X_scaled[y == i, 0], X_scaled[y == i, 1], X_scaled[y == i, 2], color=colors[i],
label=labels[i])
for i in range(3):
 ax.quiver(0, 0, 0, eigenvectors[i, 0], eigenvectors[i, 1], eigenvectors[i, 2], color='black',
length=1)
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('3D Data with Eigenvectors')
plt.legend()
plt.show()
```

```python
#  LWR-LWR DATASET
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist

# Load dataset
df_lwr = pd.read_csv("lwr_dataset.csv")

# Gaussian Kernel for weights
def gaussian_kernel(x, X, tau):
    return np.exp(-cdist([[x]], X, 'sqeuclidean') / (2 * tau**2))

# Locally Weighted Regression function
def locally_weighted_regression(X_train, y_train, tau=0.5):
    # Add intercept term to X
    X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train])

    # Generate test points (for plotting curve)
    x_min, x_max = X_train[:, 1].min(), X_train[:, 1].max()
    X_range = np.linspace(x_min, x_max, 200)
    y_pred = []

    # Perform LWR prediction for each point in X_range
    for x in X_range:
        x_vec = np.array([1, x])  # Intercept + feature
        weights = gaussian_kernel(x, X_train[:, 1:], tau).flatten()
        W = np.diag(weights)
        theta = np.linalg.pinv(X_train.T @ W @ X_train) @ (X_train.T @ W @ y_train)
        y_pred.append(x_vec @ theta)
```

```
    # Plot
    plt.scatter(X_train[:, 1], y_train, label='Data', alpha=0.7)
    plt.plot(X_range, y_pred, color='red', label=f'LWR (tau={tau})')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title("Locally Weighted Regression")
    plt.legend()
    plt.grid(True)
    plt.show()

# Run LWR
X = df_lwr[['X']].values
y = df_lwr['Y'].values
locally_weighted_regression(X, y, tau=0.5)
```

```
#  POLYNOMIAL REGRESSION-AUTO MPG DATASET

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Load the dataset
df = pd.read_csv('Auto_MPG.csv')

# Normalize column names
df.columns = df.columns.str.strip().str.lower()

# Check available columns
print("Columns:", df.columns)

# Replace missing values (e.g., '?') with NaN and drop those rows
df.replace('?', np.nan, inplace=True)
df.dropna(inplace=True)

# Convert horsepower to float (after removing '?')
df['horsepower'] = df['horsepower'].astype(float)

# Select one feature and target for polynomial regression
X = df[['horsepower']].values
y = df['mpg'].values

# Create polynomial features (degree = 2 for quadratic)
degree = 2
```

```
poly = PolynomialFeatures(degree=degree)
X_poly = poly.fit_transform(X)

# Fit Linear Regression on polynomial features
model = LinearRegression()
model.fit(X_poly, y)

# Predict using the model
y_pred = model.predict(X_poly)

# Evaluate the model
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error (Degree {degree}):", mse)

# Sort values for smooth plotting
sort_idx = X.flatten().argsort()
X_sorted = X[sort_idx]
y_pred_sorted = y_pred[sort_idx]

# Plotting
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X_sorted, y_pred_sorted, color='red', label=f'Polynomial Degree {degree}')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.title('Polynomial Regression: MPG vs Horsepower')
plt.legend()
plt.grid(True)
plt.show()
```

```
# SIMPLE LINEAR REGRESSION-BOSTON HOUSE DATASET

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the Boston Housing dataset (CSV format)
# Make sure the file path is correct
df = pd.read_csv('Boston House.csv')

# Display first few rows to understand the dataset
print("Dataset Preview:")
print(df.head())

# Select one feature for Simple Linear Regression
# We'll use 'RM' (average number of rooms per dwelling) to predict 'MEDV' (Median
house value)
X = df[['RM']]  # Independent variable (feature)
y = df['MEDV']  # Dependent variable (target)

# Split the dataset into training and testing sets (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear Regression model
model = LinearRegression()

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

```
# Make predictions
y_pred = model.predict(X_test)

# Print model coefficients
print("\nModel Coefficient (slope):", model.coef_[0])
print("Model Intercept:", model.intercept_)

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

print("\nMean Squared Error:", mse)
print("R-squared Score:", r2)

# Plot the regression line with test data
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.title('Simple Linear Regression (RM vs MEDV)')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('Median Home Value (MEDV)')
plt.legend()
plt.grid(True)
plt.show()
```

```
# DECISION TREE-BREAST CANCER DATASET

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

# 1. Load the dataset
df = pd.read_csv('Breast_cancer.csv')

# 2. Preprocess the data
# Drop columns if there is an ID or unnamed column
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

# Check for missing values
df.dropna(inplace=True)

# Encode categorical variables if necessary (example: diagnosis column 'M'/'B')
if df['diagnosis'].dtype == 'object':
    df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# 3. Split features and target
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# 4. Train/Test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. Train Decision Tree
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
```

```python
clf.fit(X_train, y_train)


# 6. Predict and Evaluate
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))


# 7. Visualize the decision tree
plt.figure(figsize=(20,10))
plot_tree(clf, feature_names=X.columns, class_names=['Benign', 'Malignant'],
filled=True)
plt.title("Decision Tree - Breast Cancer")
plt.show()
```

```
#K means-ANY RANDOM DATASET(MALL_CUSTOMER.CSV)



#without built in function
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Mall_Customers.csv')



dataset.head(5)

#we are extracting only 3rd and 4th feature
x = dataset.iloc[:, [3, 4]].values
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS



#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
 kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
 kmeans.fit(x)
 wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elbow Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
```

```
mtp.show()

#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)

#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')

mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label
= 'Centroid')

mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

```
# DBSCAN- ANY RANDOM DATASET(MALL_CUSTOMER.CSV)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('Mall_Customers.csv')

# Display first few rows
print("First 5 rows of the dataset:")
print(df.head())

# Select relevant features (you can change depending on your goal)
X = df[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Feature Scaling (important for DBSCAN)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)  # You may tune these parameters
labels = dbscan.fit_predict(X_scaled)

# Add the cluster labels to the original data
df['Cluster'] = labels

# Print unique cluster labels
```

```
print(f"\nUnique cluster labels: {set(labels)}")

# Plot the clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)',
        hue='Cluster', palette='Set1', data=df, legend='full')
plt.title('DBSCAN Clustering on Mall Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```

```
# SVM- ANY RANDOM DATASET(MALL_CUSTOMER.CSV)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.cluster import KMeans

# Load dataset
data = pd.read_csv('Mall_Customers.csv')

# Preprocess: Select features (for example, income and score)
X = data[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Optional: Create labels using clustering (unsupervised to supervised workaround)
kmeans = KMeans(n_clusters=3, random_state=42)
y = kmeans.fit_predict(X)

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

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Fit SVM classifier
classifier = SVC(kernel='rbf', random_state=42)
classifier.fit(X_train, y_train)

# Predict
y_pred = classifier.predict(X_test)

# Evaluation
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Optional: Visualization
def visualize_results(X_set, y_set, title):
    from matplotlib.colors import ListedColormap
    X1, X2 = np.meshgrid(
        np.arange(start=X_set[:, 0].min() - 1, stop=X_set[:, 0].max() + 1, step=0.01),
        np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1, step=0.01)
    )
    plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
                 alpha=0.75, cmap=ListedColormap(('red', 'green', 'blue')))
    plt.scatter(X_set[:, 0], X_set[:, 1], c=y_set, cmap=ListedColormap(('red', 'green', 'blue')))
    plt.title(title)
    plt.xlabel('Annual Income (scaled)')
    plt.ylabel('Spending Score (scaled)')
    plt.show()
```

```
visualize_results(X_train, y_train, "SVM (Training set)")
visualize_results(X_test, y_test, "SVM (Test set)")
```