1. data preprocessing

**Handling miss value**

import pandas as pd

import numpy as np

# Step 1: Load Dataset (Replace 'data.csv' with your actual file)

df = pd.read\_csv("data.csv")

# Step 2: Display missing values count

print("Missing Values Before Handling:\n", df.isnull().sum())

# Step 3: Handling Missing Values

# 3.1 Fill missing values with Mean (for numerical columns)

df["Column1"] = df["Column1"].fillna(df["Column1"].mean())

print("\nMissing Values After Handling:\n", df.isnull().sum())

**label encoding**

label\_encoder = LabelEncoder()

df["Category\_LabelEncoded"] = label\_encoder.fit\_transform(df["Category"])

print("\nDataset after Label Encoding:\n", df)

**Normalization**

# Step 2: Apply Normalization (Min-Max Scaling)

minmax\_scaler = MinMaxScaler()

df\_normalized = pd.DataFrame(minmax\_scaler.fit\_transform(df), columns=df.columns)

2) logistic regfression

import pandas as pd

import numpy as np

from sklearn.preprocessing

nd Living Area (GrLivArea)")

plt.ylabel("Sale Price")

plt.title("Simple Linear Regression")

# Residual Plot for Multiple Linear Regression

plt.subplot(1, 2, 2)

sns.residplot(x=y\_pred\_multiple, y=(y\_test - y\_pred\_multiple), lowess=True, color="green")

plt.xlabel("Predicted Price")

plt.ylabel("Residuals")

plt.title("Multiple Linear Regression Residual Plot")

plt.tight\_layout()

plt.show()

3) X = df[['GrLivArea', 'TotalBsmtSF', 'GarageArea']] # Features

y = df['PriceCategory'] # Binary Target (0 = Low Price, 1 = High Price)

# Step 4: Handle Missing Values

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

# Step 5: Split Data into Training & Testing Sets (80% Train, 20% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 6: Standardize Features (for better performance)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Step 7: Train Logistic Regression Model

log\_reg = LogisticRegression()

log\_reg.fit(X\_train, y\_train)

y\_pred = log\_reg.predict(X\_test)

y\_prob = log\_reg.predict\_proba(X\_test)[:, 1] # Probabilities for class 1

# Step 8: Model Evaluation - Accuracy & Confusion Matrix

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Model Accuracy: {accuracy:.2f}")

print("Confusion Matrix:\n", conf\_matrix)

# Step 9: ROC Curve & AUC Score

fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

# Step 10: Plot ROC Curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color="blue", label=f"ROC Curve (AUC = {roc\_auc:.2f})")

plt.plot([0, 1], [0, 1], color="grey", linestyle="--") # Diagonal line

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver Operating Characteristic (ROC) Curve")

plt.legend(loc="lower right")

plt.show()

4)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Step 1: Load Dataset

file\_path = "/mnt/data/House-Price.xlsx" # Update the path if needed

df = pd.read\_excel(file\_path)

# Step 2: Convert Problem into Binary Classification

# Predict whether SalePrice is high (above median) or low (below median)

df['PriceCategory'] = (df['SalePrice'] > df['SalePrice'].median()).astype(int)

# Step 3: Select Features & Target Variable

X = df[['GrLivArea', 'TotalBsmtSF', 'GarageArea']] # Features

y = df['PriceCategory'] # Binary Target (0 = Low Price, 1 = High Price)

# Step 4: Handle Missing Values

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

# Step 5: Split Data into Training & Testing Sets (80% Train, 20% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 6: Standardize Features (Important for KNN)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Step 7: Find Best k using Cross-Validation

k\_values = range(1, 21)

cv\_scores = []

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=5, scoring='accuracy')

cv\_scores.append(scores.mean())

# Best k value

best\_k = k\_values[np.argmax(cv\_scores)]

print(f"Best k value: {best\_k}")

# Step 8: 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)

# Step 9: Model Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Model Accuracy: {accuracy:.2f}")

print("Confusion Matrix:\n", conf\_matrix)

# Step 10: Plot k vs Accuracy

plt.figure(figsize=(8, 5))

plt.plot(k\_values, cv\_scores, marker='o', linestyle='dashed', color='blue', label="Cross-Validation Accuracy")

plt.xlabel("Number of Neighbors (k)")

plt.ylabel("Accuracy")

plt.title("Optimizing k in KNN Classification")

plt.legend()

plt.show()

5) import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Step 1: Load Dataset

file\_path = "/mnt/data/House-Price.xlsx" # Update the path if needed

df = pd.read\_excel(file\_path)

# Step 2: Select Features for Clustering

X = df[['GrLivArea', 'TotalBsmtSF']] # Using two features for 2D visualization

# Step 3: Handle Missing Values

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

# Step 4: Standardize Features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 5: Find Optimal k using Elbow Method

wcss = [] # Within-cluster sum of squares

k\_values = range(1, 11)

for k in k\_values:

kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

kmeans.fit(X\_scaled)

wcss.append(kmeans.inertia\_)

# Plot Elbow Method

plt.figure(figsize=(8, 5))

plt.plot(k\_values, wcss, marker='o', linestyle='dashed', color='blue')

plt.xlabel("Number of Clusters (k)")

plt.ylabel("WCSS (Within-Cluster Sum of Squares)")

plt.title("Elbow Method for Optimal k")

plt.show()

# Step 6: Apply K-Means Clustering with Optimal k (Assume k=3 from elbow method)

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42, n\_init=10)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Step 7: Visualize Clusters in 2D

plt.figure(figsize=(8, 6))

sns.scatterplot(x=X\_scaled[:, 0], y=X\_scaled[:, 1], hue=df['Cluster'], palette="viridis", s=100)

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], c='red', marker='X', s=200, label='Centroids')

plt.xlabel("GrLivArea (Standardized)")

pl

6)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, confusion\_matrix

import graphviz

from IPython.display import display

# Step 1: Load Dataset

file\_path = "/mnt/data/House-Price.xlsx" # Update the path if needed

df = pd.read\_excel(file\_path)

# Step 2: Convert Problem into Binary Classification

df['PriceCategory'] = (df['SalePrice'] > df['SalePrice'].median()).astype(int)

# Step 3: Select Features & Target Variable

X = df[['GrLivArea', 'TotalBsmtSF', 'GarageArea']] # Features

y = df['PriceCategory'] # Binary Target (0 = Low Price, 1 = High Price)

# Step 4: Handle Missing Values

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

# Step 5: Split Data into Training & Testing Sets (80% Train, 20% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 6: Train Decision Tree Model

dtree = DecisionTreeClassifier(max\_depth=3, random\_state=42) # Limit depth for better visualization

dtree.fit(X\_train, y\_train)

y\_pred\_tree = dtree.predict(X\_test)

# Step 7: Train Random Forest Model

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

# Step 8: Model Evaluation

accuracy\_tree = accuracy\_score(y\_test, y\_pred\_tree)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

print(f"Decision Tree Accuracy: {accuracy\_tree:.2f}")

print(f"Random Forest Accuracy: {accuracy\_rf:.2f}")

# Step 9: Visualize Decision Tree using Graphviz

dot\_data = export\_graphviz(dtree, out\_file=None, feature\_names=X.columns, class\_names=["Low Price", "High Price"],

filled=True, rounded=True, special\_characters=True)

graph = graphviz.Source(dot\_data)

display(graph) # Displays the decision tree visualization

7)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Step 1: Load Dataset

file\_path = "/mnt/data/House-Price.xlsx" # Update the path if needed

df = pd.read\_excel(file\_path)

# Step 2: Convert Problem into Binary Classification

df['PriceCategory'] = (df['SalePrice'] > df['SalePrice'].median()).astype(int)

# Step 3: Select Features & Target Variable

X = df[['GrLivArea', 'TotalBsmtSF']] # Using 2 features for visualization

y = df['PriceCategory'] # Binary Target (0 = Low Price, 1 = High Price)

# Step 4: Handle Missing Values

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

# Step 5: Split Data into Training & Testing Sets (80% Train, 20% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 6: Standardize Features (Important for SVM)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Step 7: Train SVM with Linear Kernel

svm\_linear = SVC(kernel='linear', random\_state=42)

svm\_linear.fit(X\_train, y\_train)

y\_pred\_linear = svm\_linear.predict(X\_test)

# Step 8: Train SVM with RBF Kernel

svm\_rbf = SVC(kernel='rbf', gamma='scale', random\_state=42)

svm\_rbf.fit(X\_train, y\_train)

y\_pred\_rbf = svm\_rbf.predict(X\_test)

# Step 9: Model Evaluation

accuracy\_linear = accuracy\_score(y\_test, y\_pred\_linear)

accuracy\_rbf = accuracy\_score(y\_test, y\_pred\_rbf)

print(f"SVM Linear Kernel Accuracy: {accuracy\_linear:.2f}")

print(f"SVM RBF Kernel Accuracy: {accuracy\_rbf:.2f}")

# Step 10: Plot Decision Boundaries

def plot\_decision\_boundary(model, X, y, title):

h = 0.02 # Step size

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

np.arange(y\_min, y\_max, h))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6))

plt.contourf(xx, yy, Z, alpha=0.3, cmap="coolwarm")

sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y, palette="coolwarm", edgecolor="black")

plt.xlabel("GrLivArea (Standardized)")

plt.ylabel("TotalBsmtSF (Standardized)")

plt.title(title)

plt.show()

# Plot Decision Boundaries for both models

plot\_decision\_boundary(svm\_linear, X\_train, y\_train, "SVM Linear Kernel Decision Boundary")

plot\_decision\_boundary(svm\_rbf, X\_train, y\_train, "SVM RBF Kernel Decision Boundary")

8)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from mpl\_toolkits.mplot3d import Axes3D

# Step 1: Load Dataset

file\_path = "/mnt/data/House-Price.xlsx" # Update path if needed

df = pd.read\_excel(file\_path)

# Step 2: Select Numerical Features for PCA

features = ['GrLivArea', 'TotalBsmtSF', 'GarageArea', 'LotArea', 'YearBuilt']

X = df[features]

# Step 3: Handle Missing Values

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

# Step 4: Standardize Data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 5: Apply PCA to Reduce to 3 Components

pca = PCA(n\_components=3)

X\_pca = pca.fit\_transform(X\_scaled)

# Step 6: Convert PCA Output to DataFrame

df\_pca = pd.DataFrame(X\_pca, columns=['PC1', 'PC2', 'PC3'])

# Step 7: 3D Visualization

fig = plt.figure(figsize=(10, 7))

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(df\_pca['PC1'], df\_pca['PC2'], df\_pca['PC3'], c='blue', marker='o', alpha=0.6)

# Labels & Title

ax.set\_xlabel('Principal Component 1')

ax.set\_ylabel('Principal Component 2')

ax.set\_zlabel('Principal Component 3')

ax.set\_title('3D Visualization of PCA-transformed Data')

plt.show()

# Step 8: Explained Variance Ratio

print("Explained Variance Ratio:", pca.explained\_variance\_ratio\_)

9)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import string

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Step 1: Load Sample Sentiment Dataset

data = {

"text": [

"I love this product! It's amazing.",

"This is the worst experience I've ever had.",

"Absolutely fantastic! I would buy it again.",

"I hate this so much. Waste of money!",

"Not bad, but could be better.",

"The quality is terrible. Never again!",

"I'm very happy with my purchase.",

"Awful experience, totally disappointed.",

"Decent product for the price.",

"Horrible! I regret buying this."

],

"sentiment": [1, 0, 1, 0, 1, 0, 1, 0, 1, 0] # 1 = Positive, 0 = Negative

}

df = pd.DataFrame(data)

# Step 2: Text Preprocessing Function

def clean\_text(text):

text = text.lower() # Lowercase

text = re.sub(f"[{string.punctuation}]", "", text) # Remove punctuation

return text

df["clean\_text"] = df["text"].apply(clean\_text)

# Step 3: Convert Text to Numerical Features (TF-IDF Vectorization)

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(df["clean\_text"])

y = df["sentiment"]

# Step 4: Split Data into Training & Testing Sets (80% Train, 20% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 5: Train Naïve Bayes Model

nb\_classifier = MultinomialNB()

nb\_classifier.fit(X\_train, y\_train)

# Step 6: Make Predictions

y\_pred = nb\_classifier.predict(X\_test)

# Step 7: Model Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f"Naïve Bayes Classifier Accuracy: {accuracy:.2f}")

print("\nConfusion Matrix:\n", conf\_matrix)

print("\nClassification Report:\n", report)

10)

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, confusion\_matrix

# Step 1: Load MNIST Dataset

(X\_train, y\_train), (X\_test, y\_test) = keras.datasets.mnist.load\_data()

# Step 2: Normalize Pixel Values (Scale between 0 and 1)

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Step 3: Build ANN Model

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28)), # Flatten 28x28 images to 1D

keras.layers.Dense(128, activation='relu'), # Hidden Layer (128 neurons, ReLU activation)

keras.layers.Dense(10, activation='softmax') # Output Layer (10 neurons for digits 0-9, Softmax activation)

])

# Step 4: Compile Model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Step 5: Train the Model

history = model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

# Step 6: Evaluate Model Performance

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

print(f"\nTest Accuracy: {test\_acc:.2f}")

# Step 7: Predictions

y\_pred = np.argmax(model.predict(X\_test), axis=1)

# Step 8: Classification Report

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Step 9: Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(10, 8))

plt.imshow(conf\_matrix, cmap='Blues', interpolation='nearest')

plt.colorbar()

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# Step 10: Visualize Some Predictions

def plot\_images(X, y\_true, y\_pred, num\_images=10):

plt.figure(figsize=(10, 5))

for i in range(num\_images):

plt.subplot(2, 5, i + 1)

plt.imshow(X[i], cmap='gray')

plt.title(f"True: {y\_true[i]}\nPred: {y\_pred[i]}")

plt.axis("off")

plt.show()

plot\_images(X\_test[:10], y\_test[:10], y\_pred[:10])

11)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import IsolationForest

from sklearn.svm import OneClassSVM

from sklearn.preprocessing import StandardScaler

# Step 1: Load Sample Data (Simulating Normal & Anomalous Data)

np.random.seed(42)

normal\_data = np.random.normal(loc=50, scale=10, size=(200, 2)) # Normal Data

anomalous\_data = np.random.normal(loc=80, scale=5, size=(10, 2)) # Anomalous Data

data = np.vstack((normal\_data, anomalous\_data))

# Convert to DataFrame

df = pd.DataFrame(data, columns=["Feature1", "Feature2"])

# Step 2: Normalize Data

scaler = StandardScaler()

df\_scaled = scaler.fit\_transform(df)

# Step 3: Train Isolation Forest Model

iso\_forest = IsolationForest(n\_estimators=100, contamination=0.05, random\_state=42)

iso\_preds = iso\_forest.fit\_predict(df\_scaled)

# Step 4: Train One-Class SVM Model

oc\_svm = OneClassSVM(kernel="rbf", nu=0.05)

oc\_preds = oc\_svm.fit\_predict(df\_scaled)

# Step 5: Convert Predictions (-1: Anomaly, 1: Normal)

df["IsoForest\_Label"] = iso\_preds

df["OneClassSVM\_Label"] = oc\_preds

# Step 6: Visualize Anomalies Detected by Isolation Forest

plt.figure(figsize=(10, 6))

sns.scatterplot(x=df["Feature1"], y=df["Feature2"], hue=df["IsoForest\_Label"], palette={1: 'blue', -1: 'red'})

plt.title("Anomaly Detection using Isolation Forest")

plt.legend(["Normal", "Anomaly"])

plt.show()

# Step 7: Visualize Anomalies Detected by One-Class SVM

plt.figure(figsize=(10, 6))

sns.scatterplot(x=df["Feature1"], y=df["Feature2"], hue=df["OneClassSVM\_Label"], palette={1: 'blue', -1: 'red'})

plt.title("Anomaly Detection using One-Class SVM")

plt.legend(["Normal", "Anomaly"])

plt.show()