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
```

```python
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')
```

```python
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)

```python
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
```

```python
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"])
```

```python
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()
```