ML Practical

***** SLIP 1 *****

**** Q1) Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25**
==>
1. Keep groceries.csv in the same folder as your python file.
2. Install mlxtend (if needed):
```
                pip install mlxtend
```
3. Run the program:
```
                python apriori_groceries.py
```

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Load dataset
data = pd.read_csv("groceries.csv", header=None)

# Convert dataset to list of lists
transactions = data.apply(lambda row: row.dropna().tolist(), axis=1).tolist()

# Transaction Encoder
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori Algorithm
frequent_items = apriori(df, min_support=0.25, use_colnames=True)

print("Frequent Itemsets (Support >= 0.25):")
print(frequent_items)

# Generate association rules
rules = association_rules(frequent_items, metric="lift", min_threshold=1)
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

**** Q2) Write a Python program to prepare Scatter Plot for Iris Dataset. Convert Categorical values in numeric format for a dataset.**
==>
1. Install required libraries (if needed):
```
                pip install seaborn matplotlib pandas scikit-learn
```
2. Run the Python file:
```
                python iris_scatter.py
```

```
import pandas as pd
from sklearn import datasets
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load iris dataset
```

```
iris = datasets.load_iris()

# Create DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target_names[iris.target]

# Convert categorical values to numeric
le = LabelEncoder()
df['species_num'] = le.fit_transform(df['species'])

# Scatter plot
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'], c=df['species_num'])
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("Scatter Plot of Iris Dataset")
plt.show()
```

*** SLIP 2 ***

** Q1) Write a python program to implement simple Linear Regression for predicting house price. First find all null values in a given dataset and remove them.

==>

1. Keep your dataset file in the same folder (example: house.csv).
2. Install required libraries if needed:
   ```
   pip install pandas scikit-learn matplotlib
   ```
3. Run your file:
   ```
   python simple_lr_house.py
   ```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("house.csv")

# 1. Find null values
print("Null values in dataset:")
print(df.isnull().sum())

# 2. Remove null rows
df = df.dropna()

# Select features and target
# (Assuming dataset has columns: "Area" and "Price")
X = df[['Area']]          # Independent variable
y = df['Price']           # Dependent variable

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

# Create model
model = LinearRegression()
```

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

# Predict
predicted = model.predict(X_test)

print("\nPredicted Prices:")
print(predicted)

print("\nModel Coefficient:", model.coef_)
print("Model Intercept:", model.intercept_)
```

** Q2) The data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units on diverse product categories. Using data Wholesale customer dataset compute agglomerative clustering to find out annual spending clients in the same region.

==>

1. Keep dataset file in same folder (example: Wholesale_customers.csv)
2. Install required libraries:

```
pip install pandas scikit-learn matplotlib scipy
```

3. Run:

```
python agglomerative_wholesale.py
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# Load dataset
df = pd.read_csv("Wholesale_customers.csv")

# Select only annual spending columns (drop region/channel if present)
X = df.select_dtypes(include=['int64', 'float64'])

# Handle missing values if any
X = X.dropna()

# Standardize data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ------------------- Dendrogram (for understanding clusters) -------------------
linked = linkage(X_scaled, method='ward')

plt.figure(figsize=(10, 6))
dendrogram(linked)
plt.title("Dendrogram for Wholesale Customers")
plt.xlabel("Customers")
plt.ylabel("Distance")
plt.show()

# ------------------- Agglomerative Clustering -------------------
```

```
model = AgglomerativeClustering(n_clusters=3, linkage='ward')
labels = model.fit_predict(X_scaled)

df['Cluster'] = labels

print("Clustered Data:")
print(df[['Cluster']].head())

# Visualize
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels)
plt.title("Agglomerative Clustering on Wholesale Customers")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

***** SLIP 3 *****

**** Q1) Write a python program to implement multiple Linear Regression for a house price dataset. Divide the dataset into training and testing data.**
==>
1. Keep your dataset file in the same folder. Example name: house.csv
2. Install required libraries:
```
pip install pandas scikit-learn
```
3. Run:
```
python multiple_lr_house.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("house.csv")

# Check null values
print("Null Values:\n", df.isnull().sum())

# Remove null rows
df = df.dropna()

# --------- Select features (X) and target (y) ---------
# Example: using Area, Bedrooms, Age to predict Price
X = df[['Area', 'Bedrooms', 'Age']]
y = df['Price']

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

# --------- Create & Train model ---------
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# --------- Predict on test data ---------
predicted = model.predict(X_test)

print("\nPredicted Prices:")
print(predicted)

print("\nModel Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
```

**** Q2) Use dataset crash.csv is an accident survivor's dataset portal for USA hosted by data.gov. The dataset contains passengers age and speed of vehicle (mph) at the time of impact and fate of passengers (1 for survived and 0 for not survived) after a crash. use logistic regression to decide if the age and speed can predict the survivability of the passengers.**

==>

1. Keep dataset in same folder. Example file name: crash.csv
2. Install required libraries:

```
        pip install pandas scikit-learn
```

3. Run:

```
        python logistic_crash.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("crash.csv")

# Show first few rows
print(df.head())

# Select features and target
# Assume columns are: 'age', 'speed', 'survived'
X = df[['age', 'speed']]
y = df['survived']   # 1 = survived, 0 = not survived

# Handle missing values
X = X.dropna()
y = y.loc[X.index]

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

# Create logistic regression model
model = LogisticRegression()

# Train model
model.fit(X_train, y_train)

# Predict survivability
```

```
y_pred = model.predict(X_test)

print("\nPredicted survivability (1=survived, 0=dead):")
print(y_pred)

# Accuracy
acc = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", acc)

# Model Coefficients
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

*** SLIP 4 ***

** Q1) Write a python program to implement k-means algorithm on a mall_customers dataset.

==>
1. Keep dataset in same folder (example: mall_customers.csv)
2. Install required libraries:
```
		pip install pandas scikit-learn matplotlib
```
3. Run the file:
```
		python kmeans_mall.py
```

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

# Load dataset
df = pd.read_csv("mall_customers.csv")

print(df.head())

# Select features for clustering
# Commonly used: Annual Income and Spending Score
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]

# ----- K-Means model -----
kmeans = KMeans(n_clusters=5, random_state=42)
labels = kmeans.fit_predict(X)

# Add cluster labels to dataset
df['Cluster'] = labels

print("\nClustered Data:")
print(df[['Annual Income (k$)', 'Spending Score (1-100)', 'Cluster']].head())

# ----- Visualization -----
plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'], c=labels)
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1–100)")
plt.title("K-Means Clustering on Mall Customers")
plt.show()
```

**** Q2) Write a python program to Implement Simple Linear Regression for predicting house price.**
==>
1. Keep dataset in same folder (example: house.csv)
2. Install packages (if needed):

```
pip install pandas scikit-learn
```

3. Run:

```
python simple_lr_house.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("house.csv")

print(df.head())

# Select feature and target
# Example: Area -> Price
X = df[['Area']]      # independent variable
y = df['Price']       # dependent variable

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

# Create model
model = LinearRegression()

# Train model
model.fit(X_train, y_train)

# Predict
pred = model.predict(X_test)

print("\nPredicted Prices:")
print(pred)

print("\nModel Coefficient:", model.coef_)
print("Model Intercept:", model.intercept_)
```

*** SLIP 5 ***

**** Q1) Write a python program to implement Multiple Linear Regression for Fuel Consumption dataset.**
==>
1. Keep the dataset (example: FuelConsumption.csv) in the same folder.
2. Install required libraries:

```
pip install pandas scikit-learn
```

3. Run:

```
python multiple_lr_fuel.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("FuelConsumption.csv")

print("Dataset Loaded:")
print(df.head())

# Select features (example: Engine Size, Cylinders, Fuel Consumption)
X = df[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']]
y = df['CO2EMISSIONS']   # Target variable

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

# Create model
model = LinearRegression()

# Train model
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

print("\nPredictions:")
print(y_pred)

print("\nCoefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

**** Q2) Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use iris Dataset)**

==>

1. No dataset file needed — Iris is built into sklearn.
2. Install required libraries (if needed):

```
pip install pandas scikit-learn
```

3. Run:

```
python knn_iris.py
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import pandas as pd

# Load iris dataset
```

```
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print("First 5 rows:")
print(df.head())

# Features and target
X = df.iloc[:, :-1]   # All 4 flower measurements
y = df['target']      # Species

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

# KNN model (choose k=3)
knn = KNeighborsClassifier(n_neighbors=3)

# Train
knn.fit(X_train, y_train)

# Predict
pred = knn.predict(X_test)

print("\nPredictions:")
print(pred)

# Accuracy
acc = accuracy_score(y_test, pred)
print("\nAccuracy:", acc)
```

***** SLIP 6 *****

**** Q1) Write a python program to implement Polynomial Linear Regression for Boston Housing Dataset.**
==>
1. Keep dataset in same folder (example: BostonHousing.csv or boston.csv)
2. Install required libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python poly_reg_boston.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("BostonHousing.csv")   # use your actual dataset name
```

```
print(df.head())

# Choose feature (example: number of rooms 'RM')
X = df[['RM']]
y = df['MEDV']    # or 'PRICE' depending on dataset

# Split into training/testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Polynomial Transformation (degree 2)
poly = PolynomialFeatures(degree=2)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

# Train Polynomial Regression model
model = LinearRegression()
model.fit(X_poly_train, y_train)

# Predict
y_pred = model.predict(X_poly_test)

print("\nPredicted prices:")
print(y_pred)

# Plotting Polynomial Regression Curve
plt.scatter(X, y, color='blue')
X_all_poly = poly.fit_transform(X)
plt.plot(X, model.predict(X_all_poly), color='red')
plt.xlabel("RM - Number of Rooms")
plt.ylabel("House Price (MEDV)")
plt.title("Polynomial Regression (Boston Housing)")
plt.show()
```

**** Q2) Use K-means clustering model and classify the employees into various income groups or clusters. Preprocess data if require (i.e. drop missing or null values).**
==>
1. Keep dataset in same folder (example: employees.csv).
2. Install required libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python kmeans_employee_income.py
```

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

# Load dataset
df = pd.read_csv("employees.csv")

print("Dataset Loaded:")
print(df.head())

# ---------- Preprocessing: Handle Missing Values ----------
df = df.dropna()   # Removes null rows
print("\nNull values after cleaning:")
print(df.isnull().sum())

# ---------- Select features ----------
# Assume dataset contains: 'Age', 'Annual_Income'
X = df[['Annual_Income']]    # You can also use multiple features

# ---------- K-means Clustering ----------
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)

df['Cluster'] = labels  # Add cluster labels to dataset

print("\nClustered Employee Data:")
print(df[['Annual_Income', 'Cluster']].head())

# ---------- Visualization ----------
plt.scatter(df['Annual_Income'], [0]*len(df), c=df['Cluster'])
plt.xlabel("Annual Income")
plt.title("Employee Income Groups using K-Means")
plt.show()
```

***** SLIP 7 *****

**** Q1) Fit the simple linear regression model to Salary_positions.csv data. Predict the sa of level 11 and level 12 employees.**
==>
1. Keep Salary_positions.csv in same folder.
2. Install required libraries:
```
pip install pandas scikit-learn matplotlib
```
3. Run:
```
python salary_simple_lr.py
```

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("Salary_positions.csv")

print("Dataset Loaded:")
print(df.head())

# Separate feature and target
X = df[['Level']]       # Independent variable
y = df['Salary']        # Dependent variable

# Create model
model = LinearRegression()
```

```
# Train model
model.fit(X, y)

# Predictions for Level 11 and Level 12
pred_11 = model.predict([[11]])
pred_12 = model.predict([[12]])

print("\nPredicted Salary for Level 11:", pred_11[0])
print("Predicted Salary for Level 12:", pred_12[0])

# -------- Optional: Plot --------
plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.xlabel("Level")
plt.ylabel("Salary")
plt.title("Simple Linear Regression (Salary vs Level)")
plt.show()
```

**** Q2) Write a python program to implement Naive Bayes on weather forecast dataset.**
==>
1. Keep the dataset in same folder. Example: weather.csv
2. Install required libraries:

```
pip install pandas scikit-learn
```

3. Run:

```
python naive_weather.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("weather.csv")

print("Dataset Loaded:")
print(df.head())

# Encode categorical columns
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Features and target
X = df.drop('Play', axis=1)
y = df['Play']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# Create Naive Bayes model
model = GaussianNB()

# Train model
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

print("\nPredictions:")
print(y_pred)

# Accuracy
acc = accuracy_score(y_test, y_pred)
print("\nAccuracy:", acc)
```

*** SLIP 8 ***

** Q1) Write a python program to categorize the given news text into one of the available 20 categories of news groups, using multinomial Naïve Bayes machine learning model.
==>
1. Install required libraries:

```
pip install scikit-learn
```

2. Run:

```
python news_nb.py
```

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

# Load dataset (training and test)
train_data = fetch_20newsgroups(subset='train', shuffle=True)
test_data = fetch_20newsgroups(subset='test', shuffle=True)

print("Training samples:", len(train_data.data))
print("Testing samples:", len(test_data.data))

# Create a pipeline for text processing + model
text_clf = Pipeline([
    ('vect', CountVectorizer()),        # Convert text to word counts
    ('tfidf', TfidfTransformer()),      # TF-IDF transformation
    ('clf', MultinomialNB())            # Multinomial Naive Bayes
])

# Train the model
text_clf.fit(train_data.data, train_data.target)

# Predict on test data
predicted = text_clf.predict(test_data.data)

# Accuracy
```

```
acc = accuracy_score(test_data.target, predicted)
print("\nAccuracy:", acc)

# Example prediction
sample_text = ["The GPU performance of the new NVIDIA card is amazing"]
category = text_clf.predict(sample_text)
print("\nSample Prediction Category:", train_data.target_names[category[0]])
```

** Q2) Write a python program to implement Decision Tree whether or not to play Tennis.
==>
1. Keep dataset in same folder. Example file: play_tennis.csv
2. Install libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python dt_tennis.py
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("play_tennis.csv")

print("Dataset Loaded:")
print(df.head())

# Encode categorical variables
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Features and target
X = df.drop('Play', axis=1)
y = df['Play']

# Create model
model = DecisionTreeClassifier(criterion='entropy')

# Train model
model.fit(X, y)

# Predict (example)
pred = model.predict([[2, 1, 0, 1]])  # sample encoded input
print("\nSample Prediction:", pred)

# Plot Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.columns, filled=True)
plt.title("Decision Tree - Play Tennis")
plt.show()
```

***** SLIP 9 *****

**** Q1) Implement Ridge Regression and Lasso regression model using boston_houses.csv and take only 'RM' and 'Price' of the houses. Divide the data as training and testing data. Fit line using Ridge regression and to find price of a house if it contains 5 rooms and compare results.**

==>

1. Keep dataset: boston_houses.csv in same folder

2. Install libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python ridge_lasso_boston.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("boston_houses.csv")   # dataset must contain RM and Price

print("Dataset Loaded:")
print(df.head())

# Select only RM (rooms) and Price
X = df[['RM']]
y = df['Price']

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

# -------- Ridge Regression --------
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

pred_ridge_5 = ridge.predict([[5]])

print("\nRidge Regression:")
print("Coefficient:", ridge.coef_)
print("Intercept:", ridge.intercept_)
print("Predicted Price for RM = 5:", pred_ridge_5[0])

# -------- Lasso Regression --------
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)

pred_lasso_5 = lasso.predict([[5]])

print("\nLasso Regression:")
print("Coefficient:", lasso.coef_)
print("Intercept:", lasso.intercept_)
print("Predicted Price for RM = 5:", pred_lasso_5[0])
```

```
# -------- Plot (optional for exam) --------
plt.scatter(X, y, color='blue')
plt.plot(X, ridge.predict(X), color='red', label='Ridge Line')
plt.plot(X, lasso.predict(X), color='green', label='Lasso Line')
plt.xlabel("RM (Number of Rooms)")
plt.ylabel("Price")
plt.title("Ridge vs Lasso Regression (Boston Housing)")
plt.legend()
plt.show()
```

**** Q2) Write a python program to implement Linear SVM using UniversalBank.csv**
==>
1. Put UniversalBank.csv in same folder.
2. Install libs (if needed):

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python linear_svm_universalbank.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
df = pd.read_csv("UniversalBank.csv")  # ensure the CSV is present

print("Columns:", df.columns.tolist())

# Typical columns in Universal Bank dataset:
# ['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg', 'Education', 'Mortgage',
'Personal Loan', 'Securities Account', 'CD Account', 'Online', 'CreditCard']
# Adjust if your CSV column names differ.

# Identify target column
target_candidates = ['Personal Loan', 'PersonalLoan', 'Personal_Loan',
'PersonalLoan?','PersonalLoan']
target_col = next((c for c in target_candidates if c in df.columns), None)
if target_col is None:
    # fallback: choose last column if it looks binary
    target_col = df.columns[-1]
    print(f"Using last column as target: {target_col}. Make sure it's 'Personal Loan' (0/1).")

# Drop ID and ZIP Code if present
drop_cols = []
for c in ['ID', 'Id', 'id', 'ZIP Code', 'ZIP', 'Zip', 'Zipcode', 'zip']:
    if c in df.columns:
        drop_cols.append(c)

df = df.drop(columns=drop_cols, errors='ignore')
```

```
# Handle missing values
df = df.dropna()

# If any categorical columns exist, convert them (Education often numeric, Family numeric)
# If there are true categorical strings, use get_dummies
for col in df.select_dtypes(include=['object']).columns:
    df[col] = pd.get_dummies(df[col], drop_first=True)

# Separate features and target
X = df.drop(columns=[target_col])
y = df[target_col]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# Linear SVM (use linear kernel)
svm = SVC(kernel='linear', C=1.0, random_state=42)

# Train
svm.fit(X_train, y_train)

# Predict
y_pred = svm.predict(X_test)

# Evaluation
acc = accuracy_score(y_test, y_pred)
print("\nAccuracy:", acc)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Linear SVM (UniversalBank)')
plt.show()
```

*** SLIP 10 ***

**** Q1) Write a python program to transform data with Principal Component Analysis (PCA). Use iris dataset.**

==>

1. No CSV needed — Iris dataset is built-in.

2. Install if needed:

```
pip install scikit-learn pandas matplotlib
```

3. Run:

```
python pca_iris.py
```

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

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

# Standardize data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA (reduce 4D → 2D)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("PCA Transformed Data (first 5 rows):")
print(X_pca[:5])

# Plot 2D PCA
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("PCA on Iris Dataset")
plt.show()
```

** Q2) Write a Python program to prepare Scatter Plot for Iris Dataset. Convert Categorical values in to numeric.
==>
1. No dataset needed (Iris is built-in).
2. Install required libraries (only if needed):

```
pip install pandas scikit-learn matplotlib
```

3. Save code as:

```
iris_scatter.py
```

4. Run:

```
python iris_scatter.py
```

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

# Load iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
# Add categorical column (species) and convert to numeric
df['species'] = iris.target   # numeric encoding (0,1,2)

print("First 5 rows after converting categorical to numeric:")
print(df.head())

# Scatter Plot (Sepal Length vs Sepal Width)
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'], c=df['species'])
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("Iris Dataset Scatter Plot")
plt.show()
```

*** SLIP 11 ***

**** Q1) Write a python program to implement Polynomial Regression for Boston Housing Dataset.**

==>

1. Keep the dataset in same folder. Example name: BostonHousing.csv
2. Install required libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python poly_boston.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("BostonHousing.csv")   # use your actual file name

print(df.head())

# Feature and target (RM -> MEDV or PRICE)
X = df[['RM']]
y = df['MEDV']     # or df['PRICE'] if your dataset uses PRICE

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

# Polynomial transformation (degree 2)
poly = PolynomialFeatures(degree=2)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

# Train polynomial regression model
model = LinearRegression()
model.fit(X_poly_train, y_train)
```

```
# Predictions
y_pred = model.predict(X_poly_test)
print("\nPredicted values:")
print(y_pred)

# Plot polynomial curve
plt.scatter(X, y, color='blue')
X_poly_all = poly.fit_transform(X)
plt.plot(X, model.predict(X_poly_all), color='red')
plt.xlabel("RM (Number of Rooms)")
plt.ylabel("House Price")
plt.title("Polynomial Regression - Boston Housing")
plt.show()
```

** Q2) Write a python program to Implement Decision Tree classifier model on Data which is extracted from images that were taken from genuine and forged banknote-like specimens. (refer UCI dataset https://archive.ics.uci.edu/dataset/267/banknote+authentication)
==>
1. Download dataset from UCI (CSV format)
2. Save the file as: banknote.csv
3. Install requirements:
```
pip install pandas scikit-learn matplotlib
```
4. Run your Python file:
```
python dt_banknote.py
```

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

# Load dataset
df = pd.read_csv("banknote.csv")   # use the dataset you downloaded

print(df.head())

# Features and target
X = df.drop('class', axis=1)
y = df['class']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Decision Tree model
model = DecisionTreeClassifier(criterion='entropy')

# Train model
model.fit(X_train, y_train)

# Predict on test data
pred = model.predict(X_test)
print("\nPredictions:", pred)
```

```
# Accuracy
print("\nAccuracy:", model.score(X_test, y_test))

# Plot the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.columns, class_names=['Forged','Genuine'], filled=True)
plt.title("Decision Tree - Banknote Authentication")
plt.show()
```

***** SLIP 12 *****

**** Q1) Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use iris Dataset).**

==>

1. No dataset needed — Iris comes built-in.
2. Install required libraries (if needed):

```
pip install pandas scikit-learn
```

3. Save the program as:

```
knn_iris.py
```

4. Run it:

```
python knn_iris.py
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import pandas as pd

# Load iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print("First 5 rows:")
print(df.head())

# Features and target
X = df.iloc[:, :-1]   # 4 measurements
y = df['target']      # species

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

# KNN model (k=3)
knn = KNeighborsClassifier(n_neighbors=3)

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

```
# Predict
pred = knn.predict(X_test)

print("\nPredictions:")
print(pred)

# Accuracy
acc = accuracy_score(y_test, pred)
print("\nAccuracy:", acc)
```

** Q2) Fit the simple linear regression and polynomial linear regression models to Salary_positions.csv data. Find which one is more accurately fitting to the given data. Also predict the salaries of level 11 and level 12 employees.
==>
1. Keep Salary_positions.csv in the same folder.
2. Install packages (if needed):
```
pip install pandas scikit-learn matplotlib
```
3. Run:
```
python salary_compare.py
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Load dataset
df = pd.read_csv("Salary_positions.csv")
X = df[['Level']]
y = df['Salary']

# ---------------- Simple Linear Regression ----------------
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# ---------------- Polynomial Linear Regression (degree 4) ----------------
poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X)

poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

# ---------------- Predictions for Level 11 and 12 ----------------
pred_11_simple = lin_reg.predict([[11]])
pred_12_simple = lin_reg.predict([[12]])

pred_11_poly = poly_reg.predict(poly.transform([[11]]))
pred_12_poly = poly_reg.predict(poly.transform([[12]]))

print("Simple Regression - Level 11 Salary:", pred_11_simple[0])
print("Simple Regression - Level 12 Salary:", pred_12_simple[0])

print("\nPolynomial Regression - Level 11 Salary:", pred_11_poly[0])
print("Polynomial Regression - Level 12 Salary:", pred_12_poly[0])

# ---------------- Compare Accuracy (Visual) ----------------
plt.scatter(X, y, color='blue')

# simple line
plt.plot(X, lin_reg.predict(X), color='red', label='Simple Regression')

# polynomial curve
plt.plot(X, poly_reg.predict(X_poly), color='green', label='Polynomial Regression')

plt.xlabel("Level")
plt.ylabel("Salary")
plt.title("Simple vs Polynomial Regression")
plt.legend()
plt.show()
```

*** SLIP 13 ***

**** Q1) Create RNN model and analyze the Google stock price dataset. Find out increasing or decreasing trends of stock price for the next day.**

==>

1. Keep dataset in same folder → example name: Google_Stock_Price.csv
2. Install required libraries:

```
pip install pandas numpy matplotlib scikit-learn tensorflow
```

3. Run:

```
python rnn_google.py
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Load dataset
df = pd.read_csv("Google_Stock_Price.csv")

# Use only the "Open" price
data = df[['Open']].values

# Scale data 0–1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Create training data (60 timesteps)
X_train = []
y_train = []
for i in range(60, len(scaled_data)):
   X_train.append(scaled_data[i-60:i, 0])
   y_train.append(scaled_data[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)
```

```
# Reshape for RNN input
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# Build RNN (LSTM)
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=50))
model.add(Dense(1))

# Train model
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=5, batch_size=32)

# Predict next day
last_60 = scaled_data[-60:]
last_60 = np.reshape(last_60, (1, 60, 1))
next_day_scaled = model.predict(last_60)
next_day_price = scaler.inverse_transform(next_day_scaled)

print("\nPredicted price for next day:", next_day_price[0][0])

# Compare with last real value
last_real_price = data[-1][0]

if next_day_price > last_real_price:
   print("Trend: Increasing")
else:
   print("Trend: Decreasing")
```

**** Q2) Write a python program to implement simple Linear Regression for predicting house price.**

==>

1. Keep your dataset in same folder (example: house.csv)
2. Install required libraries:

```
pip install pandas scikit-learn
```

3. Run:

```
python simple_lr_house.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("house.csv")

print("Dataset Loaded:")
print(df.head())

# Feature and target
# Example: Area → Price
X = df[['Area']]        # independent variable
y = df['Price']         # dependent variable
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create model
model = LinearRegression()

# Train model
model.fit(X_train, y_train)

# Predictions
pred = model.predict(X_test)

print("\nPredicted Prices:")
print(pred)

print("\nCoefficient:", model.coef_)
print("Intercept:", model.intercept_)
```

***** SLIP 14 *****

**** Q1) Create a CNN model and train it on mnist handwritten digit dataset. Using model find out the digit written by a hand in a given image.**
Import mnist dataset from tensorflow.keras.datasets.
==>
1. Install required libraries:

```
pip install tensorflow matplotlib
```

2. Save this program as cnn_mnist.py
3. Run:

```
python cnn_mnist.py
```

4. For prediction, keep an image in same folder

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image

# ---------------- Load MNIST dataset ----------------
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Reshape for CNN (batch, height, width, channels)
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test  = x_test.reshape(-1, 28, 28, 1) / 255.0

# ---------------- Build CNN model ----------------
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')   # 10 digits
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# ---------------- Train model ----------------
print("Training CNN model...")
model.fit(x_train, y_train, epochs=5, batch_size=32)

# ---------------- Evaluate ----------------
loss, acc = model.evaluate(x_test, y_test)
print("Test Accuracy:", acc)

# ---------------- Predict digit from image ----------------
# Image must be 28x28 grayscale
img = image.load_img("test_digit.png", color_mode='grayscale', target_size=(28, 28))
img = image.img_to_array(img)
img = img.reshape(1, 28, 28, 1) / 255.0

prediction = model.predict(img)
digit = np.argmax(prediction)

print("\nPredicted Digit =", digit)
```

**** Q2) Write a python program to find all null values in a given dataset and remove them. Create your own dataset.**

==>

1. Create your own dataset file (example: mydata.csv), e.g.:

```
Name,Age,Salary
Rohan,25,50000
Mira,,60000
Amit,30,
Sara,28,55000
```

2. Install pandas if needed:

```
pip install pandas
```

3. Run:

```
python remove_nulls.py
```

```
import pandas as pd

# Load your own dataset
df = pd.read_csv("mydata.csv")

print("Original Dataset:")
print(df)

# 1. Find all null values
```

```
print("\nNull Values in Dataset:")
print(df.isnull().sum())

# 2. Remove rows with null values
df_cleaned = df.dropna()

print("\nDataset After Removing Null Values:")
print(df_cleaned)

# Optional: Save cleaned data
df_cleaned.to_csv("cleaned_data.csv", index=False)
```

***** SLIP 15 *****

**** Q1) Create an ANN and train it on house price dataset classify the house price is above average or below average.**

==>

1. Dataset is already provided (house.csv).

2. Install:

```
pip install pandas scikit-learn tensorflow
```

3. Run:

```
python ann_house_classification.py
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load dataset
df = pd.read_csv("house.csv")
print(df.head())

# ---------- Create binary target column ----------
avg_price = df['Price'].mean()
df['Price_Class'] = np.where(df['Price'] >= avg_price, 1, 0)   # 1 = Above Avg, 0 = Below Avg

# ---------- Features and target ----------
# Example features – change based on your dataset columns:
X = df[['Area', 'Bedrooms', 'Age']]
y = df['Price_Class']

# ---------- Train-test split ----------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ---------- Scaling ----------
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# ---------- ANN Model ----------
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # binary output

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# ---------- Train model ----------
model.fit(X_train, y_train, epochs=30, batch_size=10, verbose=1)

# ---------- Evaluate ----------
loss, acc = model.evaluate(X_test, y_test)
print("\nAccuracy:", acc)

# ---------- Predict sample ----------
sample = np.array([[2000, 3, 5]])  # example: Area=2000, Bedrooms=3, Age=5
sample = scaler.transform(sample)
pred = model.predict(sample)
print("\nPrediction (1=Above Avg, 0=Below Avg):", (pred > 0.5).astype(int))
```

** Q2) Write a python program to implement multiple Linear Regression for a house price dataset.
==>
1. Keep dataset in same folder (example: house.csv)
2. Install required libraries:
```
pip install pandas scikit-learn
```
3. Run:
```
python multiple_lr_house.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("house.csv")

print("Dataset Loaded:")
print(df.head())

# Select multiple features
# Example if dataset contains: Area, Bedrooms, Age, Price
X = df[['Area', 'Bedrooms', 'Age']]
y = df['Price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create model
model = LinearRegression()
```

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

# Predict
y_pred = model.predict(X_test)

print("\nPredicted Prices:")
print(y_pred)

print("\nCoefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

*** SLIP 16 ***

** Q1) Create a two layered neural network with relu and sigmoid activation function.
==>
1. Install TensorFlow (if needed)

```
pip install tensorflow
```

2. Run your file:

```
python two_layer_nn.py
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Create sample input data (you can replace with real dataset)
X = np.random.rand(100, 3)   # 100 samples, 3 features
y = np.random.randint(0, 2, 100)  # Binary output (0/1)

# Build two-layer neural network
model = Sequential()

# First hidden layer → ReLU activation
model.add(Dense(8, activation='relu', input_shape=(3,)))

# Output layer → Sigmoid activation
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train model
model.fit(X, y, epochs=10, batch_size=5)

# Test prediction
sample = np.array([[0.4, 0.6, 0.8]])
pred = model.predict(sample)

print("Prediction (0 = No, 1 = Yes):", pred)
```

** Q2) Write a python program to implement Simple Linear Regression for Boston housing

dataset.
==>
1. Keep file: boston_houses.csv in same folder
2. Install:
```
pip install pandas scikit-learn
```
3. Run:
```
python simple_boston_lr.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset (they will give boston_houses.csv)
df = pd.read_csv("boston_houses.csv")

print(df.head())

# Use RM (average number of rooms) to predict PRICE
X = df[['RM']]
y = df['PRICE']

# Train–test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create and train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
pred = model.predict(X_test)

print("\nPredicted Prices:")
print(pred)

print("\nCoefficient:", model.coef_)
print("Intercept:", model.intercept_)
```

*** SLIP 17 ***

** Q1) Implement Ensemble ML algorithm on Pima Indians Diabetes Database with bagging (random forest), boosting, voting and Stacking methods and display analysis accordingly. Compare result.
==>
1. Put the dataset file in the same folder as the script.
2. Install required libs (if not already):
```
pip install pandas scikit-learn matplotlib
```
3. Run:
```
python ensemble_pima.py
```

```
# ensemble_pima.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
VotingClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")

# --------- 1. Load dataset ----------
# Expected columns (common pima dataset):
#
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFu
nction','Age','Outcome']
df = pd.read_csv("pima.csv")   # change filename if necessary

print("Dataset shape:", df.shape)
print(df.head())

# --------- 2. Basic preprocessing ----------
# If zeros represent missing values for some features (common in Pima),
# replace zeros with NaN for numeric columns where 0 is impossible or indicates
missingness
cols_with_zero_missing = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
for col in cols_with_zero_missing:
    if col in df.columns:
        df[col] = df[col].replace(0, np.nan)

# Show missing count
print("\nMissing values per column:")
print(df.isnull().sum())

# Option A (simple): drop rows with missing values
df = df.dropna()
# (Alternative: impute with median -- uncomment if you prefer)
# from sklearn.impute import SimpleImputer
# imp = SimpleImputer(strategy='median')
# df[cols_with_zero_missing] = imp.fit_transform(df[cols_with_zero_missing])

print("\nAfter dropping missing rows:", df.shape)

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

# Standardize features (important for some estimators)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# --------- 4. Define models ----------
# Bagging (Random Forest)
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Boosting (Gradient Boosting)
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)

# Strong base learners for voting/stacking
svc = SVC(probability=True, kernel='rbf', random_state=42)
lr = LogisticRegression(max_iter=1000, random_state=42)

# Voting classifier (soft voting)
voting = VotingClassifier(
   estimators=[('rf', rf), ('gb', gb), ('lr', lr)],
   voting='soft'
)

# Stacking classifier
stack = StackingClassifier(
   estimators=[('rf', rf), ('svc', svc), ('gb', gb)],
   final_estimator=LogisticRegression(),
   passthrough=False
)

# Put all models in a list for convenient looping
models = [
   ('RandomForest (Bagging)', rf),
   ('GradientBoosting (Boosting)', gb),
   ('VotingEnsemble', voting),
   ('StackingEnsemble', stack)
]

# --------- 5. Train, evaluate and compare ----------
results = {}
for name, model in models:
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)

   acc = accuracy_score(y_test, y_pred)
   results[name] = {
      'accuracy': acc,
      'report': classification_report(y_test, y_pred, zero_division=0),
      'confusion_matrix': confusion_matrix(y_test, y_pred)
   }
   print(f"\n---- {name} ----")
   print("Accuracy:", acc)
   print("Confusion matrix:\n", results[name]['confusion_matrix'])
   print("Classification report:\n", results[name]['report'])

# --------- 6. (Optional) Cross-validation for comparison ----------
print("\nCross-validation (5-fold) mean accuracies:")
for name, model in models:
    cv_scores = cross_val_score(model, X_scaled, y, cv=5, scoring='accuracy')
    print(f"{name}: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})")

# --------- 7. Summary ----------
best = max(results.items(), key=lambda x: x[1]['accuracy'])
print(f"\nBest model on test set: {best[0]} with accuracy {best[1]['accuracy']:.4f}")

# Save results to CSV (optional)
summary = [(name, info['accuracy']) for name, info in results.items()]
pd.DataFrame(summary, columns=['Model',
'Accuracy']).to_csv("ensemble_results_summary.csv", index=False)
print("\nSummary saved to ensemble_results_summary.csv")
```

** Q2) Write a python program to implement Multiple Linear Regression for a house price dataset.
==>
1. Dataset is already provided (example name: house.csv)
2. Install libraries (if needed):

```
pip install pandas scikit-learn
```

3. Run the program:

```
python multiple_house.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load dataset
df = pd.read_csv("house.csv")   # dataset already provided

print("Dataset Loaded:")
print(df.head())

# Select features and target
# (Example: Area, Bedrooms, Age → predict Price)
X = df[['Area', 'Bedrooms', 'Age']]
y = df['Price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create model
model = LinearRegression()

# Train model
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

print("\nPredicted Prices:")
print(y_pred)
print("\nModel Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
```

*** SLIP 18 ***

** Q1) Write a python program to implement k-means algorithm on a Diabetes dataset.
==>
1. Keep dataset in same folder
2. Install required libraries (if needed):
```
pip install pandas scikit-learn matplotlib
```
3. Run program:
```
python kmeans_diabetes.py
```

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

# Load dataset
df = pd.read_csv("diabetes.csv")

print("Dataset loaded:")
print(df.head())

# --------- Preprocessing ---------
# Remove missing/null rows if any
df = df.dropna()

# Select two features for clustering
# (You can choose any, commonly Glucose & BMI)
X = df[['Glucose', 'BMI']]

# --------- K-Means Model ---------
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)

# Add cluster labels to dataset
df['Cluster'] = labels

print("\nClustered Data (first 5 rows):")
print(df[['Glucose', 'BMI', 'Cluster']].head())

# --------- Visualization ---------
plt.scatter(df['Glucose'], df['BMI'], c=df['Cluster'])
plt.xlabel("Glucose")
plt.ylabel("BMI")
plt.title("K-Means Clustering on Diabetes Dataset")
plt.show()
```

** Q2) Write a python program to implement Polynomial Linear Regression for salary_positions dataset.
==>

1. Keep the dataset in same folder: Salary_positions.csv
2. Install required libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run your file:

```
python poly_salary.py
```

```
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("Salary_positions.csv")

print(df.head())

# Feature (Level) and Target (Salary)
X = df[['Level']]
y = df['Salary']

# Create polynomial features (degree 4 is common)
poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X)

# Train Polynomial Regression Model
model = LinearRegression()
model.fit(X_poly, y)

# Predict salary for Level 11 and Level 12
pred_11 = model.predict(poly.transform([[11]]))
pred_12 = model.predict(poly.transform([[12]]))

print("\nPredicted Salary for Level 11:", pred_11[0])
print("Predicted Salary for Level 12:", pred_12[0])

# Plot Polynomial Curve
plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X_poly), color='red')
plt.xlabel("Position Level")
plt.ylabel("Salary")
plt.title("Polynomial Regression on Salary Positions")
plt.show()
```

*** SLIP 19 ***

**** Q1) Fit the simple linear regression and polynomial linear regression models to Salary_positions.csv data. Find which one is more accurately fitting to the given data. Also predict the salaries of level 11 and level 12 employees.**

==>

1. Keep Salary_positions.csv in your folder.
2. Install needed libraries:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python salary_poly.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Load dataset
df = pd.read_csv("Salary_positions.csv")
X = df[['Level']]
y = df['Salary']

# ---------------- Simple Linear Regression ----------------
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# ---------------- Polynomial Linear Regression (degree 4) ----------------
poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X)

poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

# ---------------- Predict Level 11 & 12 ----------------
pred_lin_11 = lin_reg.predict([[11]])
pred_lin_12 = lin_reg.predict([[12]])

pred_poly_11 = poly_reg.predict(poly.transform([[11]]))
pred_poly_12 = poly_reg.predict(poly.transform([[12]]))

print("Simple LR Prediction - Level 11:", pred_lin_11[0])
print("Simple LR Prediction - Level 12:", pred_lin_12[0])

print("Polynomial LR Prediction - Level 11:", pred_poly_11[0])
print("Polynomial LR Prediction - Level 12:", pred_poly_12[0])

# ---------------- Plot ----------------
plt.scatter(X, y, color='blue')

# Simple LR line
plt.plot(X, lin_reg.predict(X), color='red')

# Polynomial curve
plt.plot(X, poly_reg.predict(X_poly), color='green')

plt.xlabel("Level")
plt.ylabel("Salary")
plt.title("Simple vs Polynomial Regression")
plt.show()
```

**** Q2) Write a python program to implement Naive Bayes on weather forecast dataset.**
==>
1. Keep dataset in same folder (example: weather.csv)

2. Install:

```
pip install pandas scikit-learn
```

3. Run:

```
python nb_weather.py
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("weather.csv")
print(df.head())

# Encode categorical values
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Features and target
X = df.drop('Play', axis=1)
y = df['Play']

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

# Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict
pred = model.predict(X_test)
print("\nPredictions:", pred)

# Accuracy
print("\nAccuracy:", accuracy_score(y_test, pred))
```

***** SLIP 20 *****

**** Q1) Implement Ridge Regression, Lasso regression model using boston_houses.csv and take only ‘RM’ and ‘Price’ of the houses. divide the data as training and testing data. Fit line using Ridge regression and to find price of a house if it contains 5 rooms. and compare results.**

==>

1. Keep dataset in same folder → boston_houses.csv
2. Install required packages:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python ridge_lasso_boston.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso

# Load dataset
df = pd.read_csv("boston_houses.csv")

print("Dataset loaded:")
print(df.head())

# Select only RM (rooms) and Price
X = df[['RM']]
y = df['Price']

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

# ----- Ridge Regression -----
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

# Predict price of house with 5 rooms
ridge_pred_5 = ridge.predict([[5]])

print("\nRidge Regression: Price for 5 rooms =", ridge_pred_5[0])

# ----- Lasso Regression -----
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)

# Predict price of house with 5 rooms
lasso_pred_5 = lasso.predict([[5]])

print("Lasso Regression: Price for 5 rooms =", lasso_pred_5[0])

# Compare model score
print("\nRidge Score:", ridge.score(X_test, y_test))
print("Lasso Score:", lasso.score(X_test, y_test))
```

** Q2) Write a python program to implement Decision Tree whether or not to play Tennis.
==>
1. Keep dataset in same folder (example: play_tennis.csv)
2. Install required packages:

```
pip install pandas scikit-learn matplotlib
```

3. Run:

```
python decisiontree_tennis.py
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
```

```
# Load dataset
df = pd.read_csv("play_tennis.csv")

print("Dataset Loaded:")
print(df.head())

# Encode categorical values
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Features & Target
X = df.drop("Play", axis=1)
y = df["Play"]

# Create Decision Tree Model
model = DecisionTreeClassifier(criterion="entropy")
model.fit(X, y)

# Example Prediction (encoded values)
sample = model.predict([[2, 1, 0, 1]])
print("\nSample Prediction (1 = Play, 0 = No):", sample[0])

# Plot Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.columns, filled=True)
plt.title("Decision Tree - Play Tennis")
plt.show()
```