1.Cleaning & Pre-Processing Data

import pandas as pd

try:

df = pd.read\_csv(input("CSV file path: "))

print("\n=> Data loaded.\n")

except Exception as e:

print("\n=> Error:", e)

exit()

df.fillna(0, inplace=True)

df.fillna("Unknown", inplace=True)

df.drop\_duplicates(inplace=True)

df.reset\_index(drop=True, inplace=True)

print("\n=> Data Cleaned.\n")

print("\n=> Adding Experience Level column.\n")

df['Experience\_Level'] = pd.cut(df['Experience'], [-1, 2, 5, float('inf')], labels=['Beginner', 'Intermediate', 'Expert'])

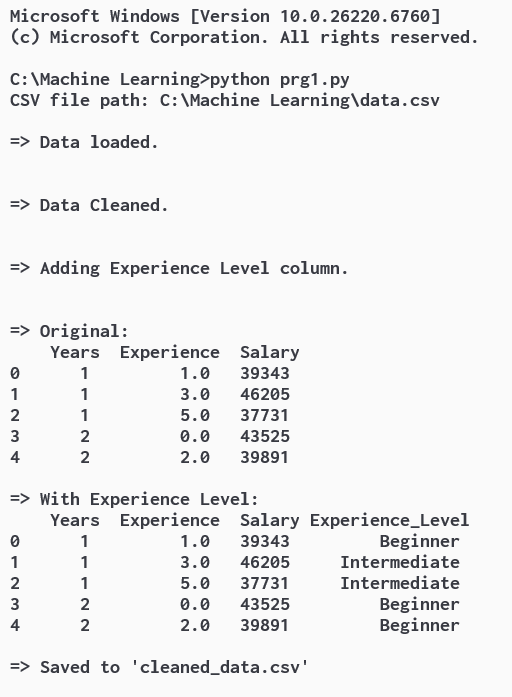
print("\n=> Original:\n", df[['Years','Experience','Salary']].head())

print("\n=> With Experience Level:\n", df.head())

df.to\_csv("cleaned\_data.csv", index=False)

print("\n=> Saved to 'cleaned\_data.csv'")

**Output:**

****

2.Feature Extraction

import pandas as pd

def get\_date\_from\_user(prompt):

while True:

try: return pd.to\_datetime(input(prompt))

except: print("Use YYYY-MM-DD format.")

start\_date = get\_date\_from\_user("\n =>Enter the start date (YYYY-MM-DD): ")

end\_date = get\_date\_from\_user("\n =>Enter the end date (YYYY-MM-DD): ")

df = pd.DataFrame({'date': pd.date\_range(start=start\_date, end=end\_date)})

print("\n Extracting features from date data...")

df["year"] = df["date"].dt.year

df["month"] = df["date"].dt.month

df["day\_of\_month"] = df["date"].dt.day

df["day\_of\_week"] = df["date"].dt.dayofweek

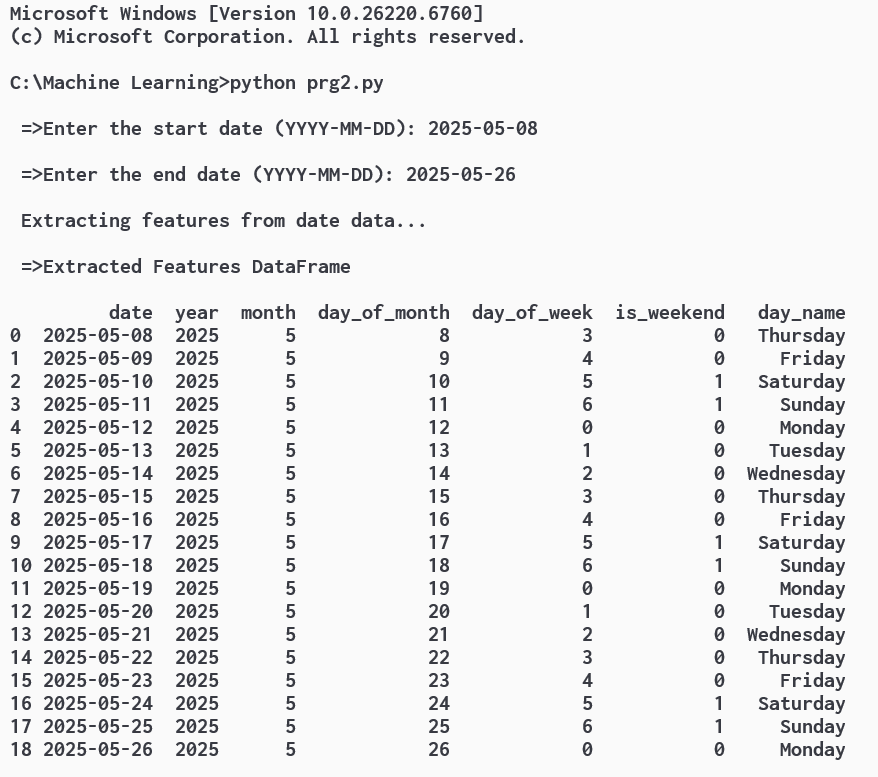
df["is\_weekend"] = df["day\_of\_week"].isin([5, 6]).astype(int)

df["day\_name"] = df["date"].dt.day\_name()

print("\n =>Extracted Features DataFrame\n ")

print(df)

**Output:**



3.Linear Regression

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

try:

data = pd.read\_csv(input("CSV file path: "))

print("\n=> Data loaded.\n")

except Exception as e:

print("\n=> Error:", e)

exit()

x = data.iloc[:, 0].values.reshape(-1, 1)

y = data.iloc[:, 1].values

model = LinearRegression().fit(x, y)

print("\n=> Training Finished By Given Data.\n")

print("\n=> Predicting By Linear Regression...\n")

target = 100

predicted\_x = (target - model.intercept\_) / model.coef\_[0]

print("\n=> Printing Predicted Data.\n")

print(f"\n{data.columns[1]} will reach {target} at {data.columns[0]} ≈ {predicted\_x:.2f}\n")

print("\n=> Plotting Data...\n")

plt.scatter(x, y)

plt.plot(x, model.predict(x), color='red')

plt.axhline(y=target, color='green', linestyle='--')

plt.axvline(x=predicted\_x, color='purple', linestyle='--')

plt.xlabel(data.columns[0])

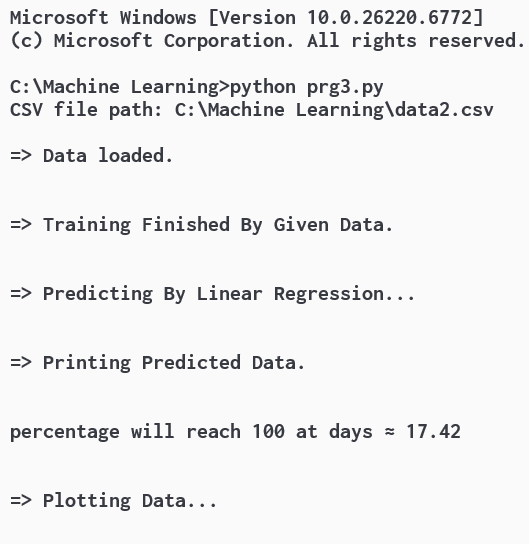
plt.ylabel(data.columns[1])

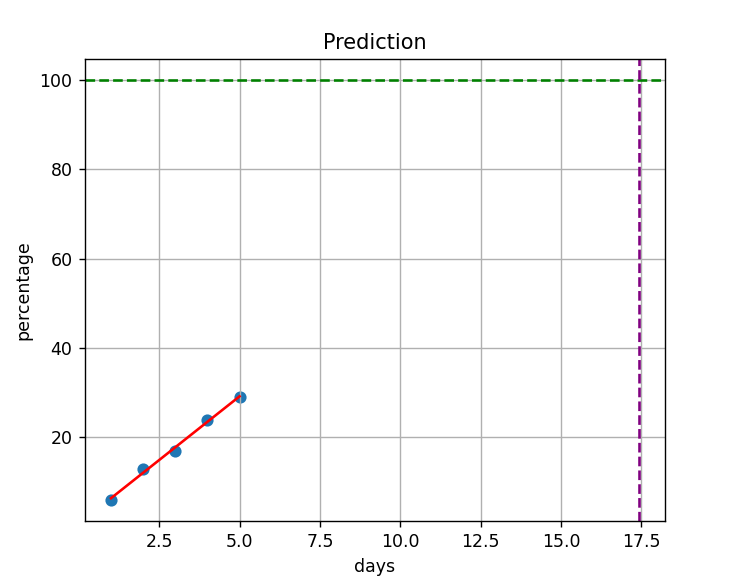
plt.title('Prediction')

plt.grid(True)

plt.show()

**Output:**





4.K Nearest Neighbours

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

# Create the dataset

data = {

'Data Point': ['A', 'B', 'C', 'D', 'E'],

'Age (years)': [25, 30, 35, 40, 45],

'Income ($)': [50000, 70000, 90000, 60000, 80000],

'Purchased': ['Yes', 'No', 'Yes', 'No', 'Yes']

}

df = pd.DataFrame(data)

# Prepare the features and target variable

X = df[['Age (years)', 'Income ($)']]

y = df['Purchased']

# Define the parameter grid

param\_grid = {'n\_neighbors': range(1, 11)}

# Create KNN classifier

knn = KNeighborsClassifier()

# Grid search

grid\_search = GridSearchCV(knn, param\_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data

grid\_search.fit(X, y)

# Print the best parameters and best score

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score:", grid\_search.best\_score\_)

5.K-Means Clustering

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import Kmeans

# Step 2: Generate Sample Data

np.random.seed(42)

X = np.array([[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])

# Step 3: Visualize the Data

plt.scatter(X[:, 0], X[:, 1], s=100, marker='.')

plt.title("Original Data")

plt.show()

# Step 4: Apply K-Means Clustering

k = 2 # You can choose the number of clusters

kmeans = KMeans(n\_clusters=k)

kmeans.fit(X)

# Step 5: Get Cluster Centers and Labels

centroids = kmeans.cluster\_centers\_

labels = kmeans.labels\_

# Step 6: Visualize Clusters

colors = ["g.", "r."]

for i in range(len(X)):

plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize=10)

plt.scatter(centroids[:, 0], centroids[:, 1], marker="x", s=150, linewidths=5, zorder=10)

plt.title("Clustered Data with Centroids")

plt.show()

6.Classification

from textblob import TextBlob

file\_path = input("Enter the full path to your text file: ")

try:

with open(file\_path, "r") as f:

content = f.read()

print("\n=>File loaded successfully.\n")

except IOError as e:

print(f"=>Could not open the file: {e}")

exit()

corrected = TextBlob(content).correct()

print("=>Original Text:\n")

print(content)

print("\n=>Corrected Text:\n")

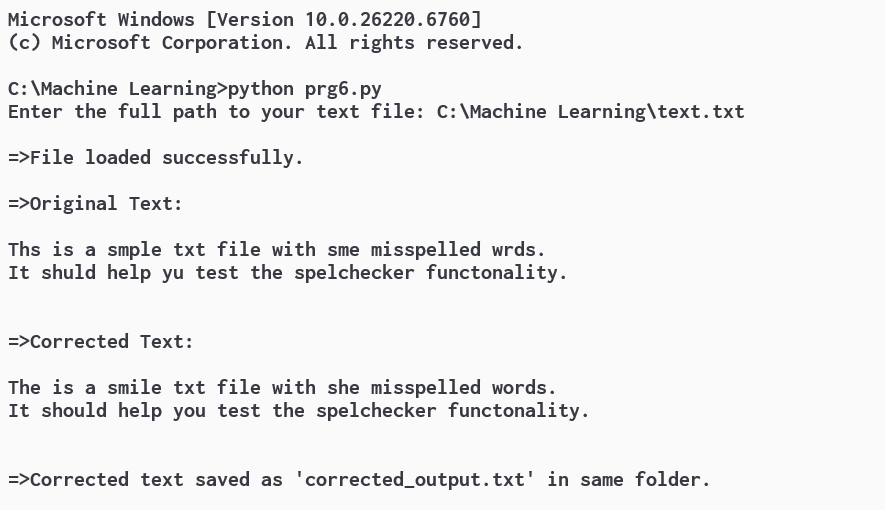
print(corrected)

with open("corrected\_output.txt", "w") as f:

f.write(str(corrected))

print("\n=>Corrected text saved as 'corrected\_output.txt' in same folder.")

**Output:**



7.Support Vector Machine

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

choice = input("\n Type 'E' to encrypt or 'D' to decrypt: ").strip().upper()

if choice == 'E':

msg = input("\n Enter the message to encrypt: ").upper()

encrypted\_nums = []

for char in msg:

if char in letters:

position = letters.index(char)

position = (position + 3) % 26

encrypted\_nums.append(position)

print("\n Encrypted numeric values: =>", encrypted\_nums, "<=")

elif choice == 'D':

nums\_str = input("\n Enter numeric values separated by spaces: ").strip()

nums = [int(n) for n in nums\_str.split()]

decrypted\_msg = ""

for num in nums:

pos = (num - 3) % 26

char = letters[pos]

decrypted\_msg += char

print("\n Decrypted message: =>", decrypted\_msg, "<=")

else:

print("\n Invalid choice. Please select 'E' or 'D'.")

**Output:**

