EXPERIMENT 1

Aim: Implement Linear and Logistic Regression on real-world datasets

1.Dataset Source: Custom dataset : Link

2. Dataset Description

The dataset consists of step-count data collected from a wearable fitness device (Fitbit/Dafit) and exported as CSV files. The raw data contains daily records with the following fields:

- timestamp – Date and time of recorded steps
- steps – Number of steps taken
- data source – Source of the recorded data

During preprocessing, daily step data was aggregated into **monthly average steps**.

Features:

- **MonthIndex** (numerical representation of month: 1, 2, 3, …)

Target Variable:

- **Monthly Average Steps**

Dataset Characteristics:

- Time-series dataset
- Small sample size (number of months available)
- Shows trend variation (increase followed by decrease)

3. Mathematical Formulation of the Algorithm

The model used is **Simple Linear Regression**.

Model Equation:

$y = \beta_0 + \beta_1 x$

Where:

- y = Predicted monthly average steps
- x = Month index
- β_0 = Intercept
- β_1 = Slope

The model parameters are estimated using the **Least Squares Method**, which minimizes the Mean Squared Error (MSE).

Mean Squared Error (MSE):

$MSE = (1 / n) \times \Sigma (y_i - \hat{y}_i)^2$
for $i = 1$ to n

Where:

- n = Number of observations
- y_i = Actual value
- $\hat{y}_i$ = Predicted value

4. Algorithm Limitations

1. Assumes linear relationship between variables.
2. Cannot capture non-linear or seasonal patterns.
3. Sensitive to outliers.
4. Performance decreases with very small datasets.
5. Long-term future predictions may be unreliable.

5. Methodology / Workflow

1. Data Export – CSV files obtained from fitness app.
2. Data Cleaning – Converted timestamp to datetime format.
3. Feature Engineering – Extracted month and computed monthly averages.

4. Model Training – Applied Linear Regression using MonthIndex as input.
5. Prediction – Predicted next month's average steps.
6. Visualization – Plotted actual values and regression line.
7. Evaluation – Computed R^2 score and error metrics.

Workflow summary:

Data Collection → Preprocessing → Monthly Aggregation
→ Model Training → Prediction → Evaluation

6. Performance Analysis

The model was evaluated using:

- **R^2 Score** – Measures how well the model explains variance in the data.
- **Mean Squared Error (MSE)** – Measures average squared prediction error.

Due to limited data, the model captures only the overall trend and may underfit if the data contains non-linear patterns.

7. Hyperparameter Tuning

Linear Regression has limited hyperparameters:

- fit_intercept (True/False)

Tuning impact is minimal for simple regression.

If Polynomial Regression is applied:

- degree becomes the main hyperparameter.
- Higher degree improves flexibility but increases risk of overfitting.

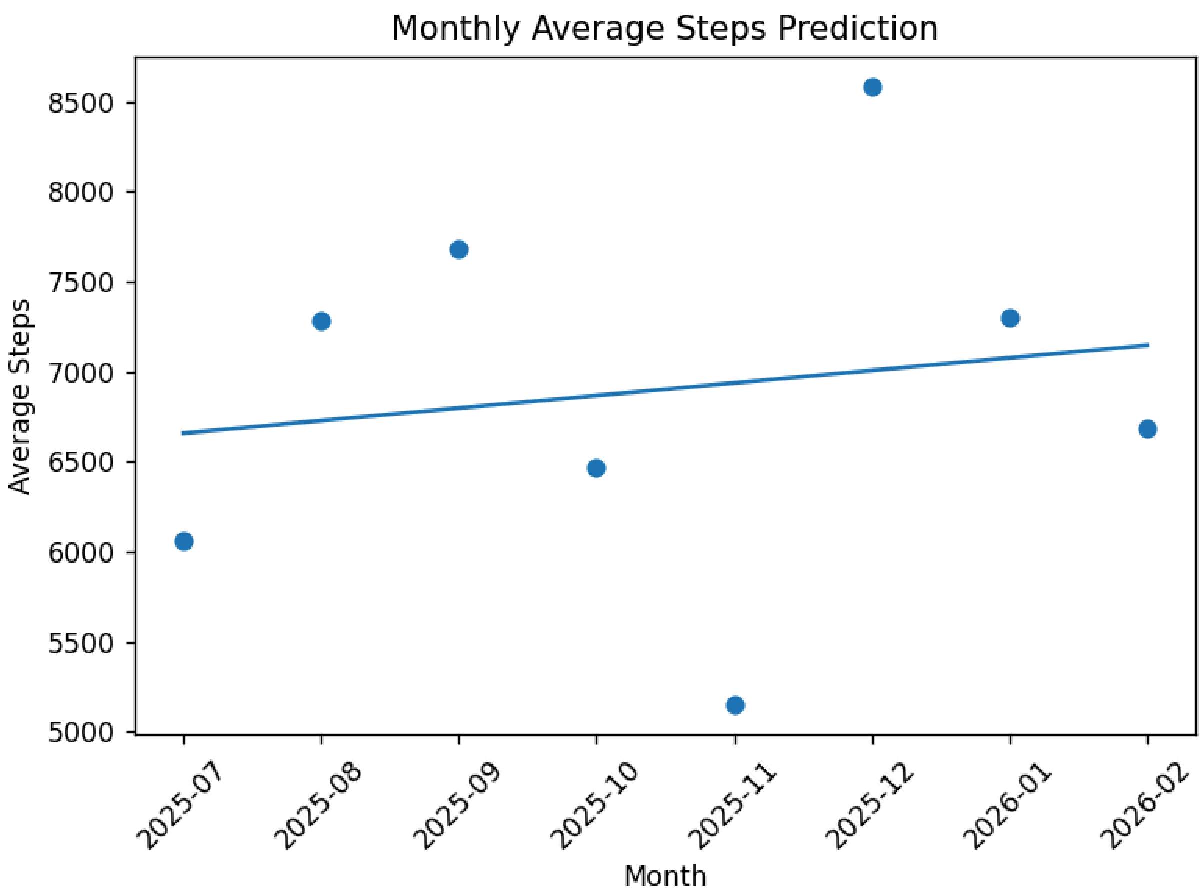
Linear Regression

Code:

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


folder_path = "."
csv_files = [f for f in os.listdir(folder_path) if f.endswith(".csv")]

df_list = []

for file in csv_files:
    df = pd.read_csv(file)
    df_list.append(df)

data = pd.concat(df_list, ignore_index=True)

print("Files loaded:", csv_files)
print("Columns:", data.columns)



data['timestamp'] = pd.to_datetime(data['timestamp'])
data = data.sort_values('timestamp')



data['Date'] = data['timestamp'].dt.date
daily_steps = data.groupby('Date')['steps'].sum().reset_index()



daily_steps['Date'] = pd.to_datetime(daily_steps['Date'])
daily_steps['YearMonth'] = daily_steps['Date'].dt.to_period('M')

monthly_avg = daily_steps.groupby('YearMonth')['steps'].mean().reset_index()
```

```python
print("\nMonthly Average Steps:")
print(monthly_avg)



monthly_avg['MonthLabel'] = monthly_avg['YearMonth'].astype(str)
monthly_avg['MonthIndex'] = range(len(monthly_avg))

X = monthly_avg[['MonthIndex']]
y = monthly_avg['steps']


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

print("\nIntercept:", model.intercept_)
print("Slope:", model.coef_[0])



next_index = monthly_avg['MonthIndex'].max() + 1
prediction = model.predict([[next_index]])

print("\nPredicted Average Steps Next Month:", int(prediction[0]))


plt.figure()
plt.scatter(monthly_avg['MonthLabel'], y)
plt.plot(monthly_avg['MonthLabel'], model.predict(X))

plt.xlabel("Month")
plt.ylabel("Average Steps")
plt.title("Monthly Average Steps Prediction")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Output:




```
Predicted Average Steps Next Month: 7214
```

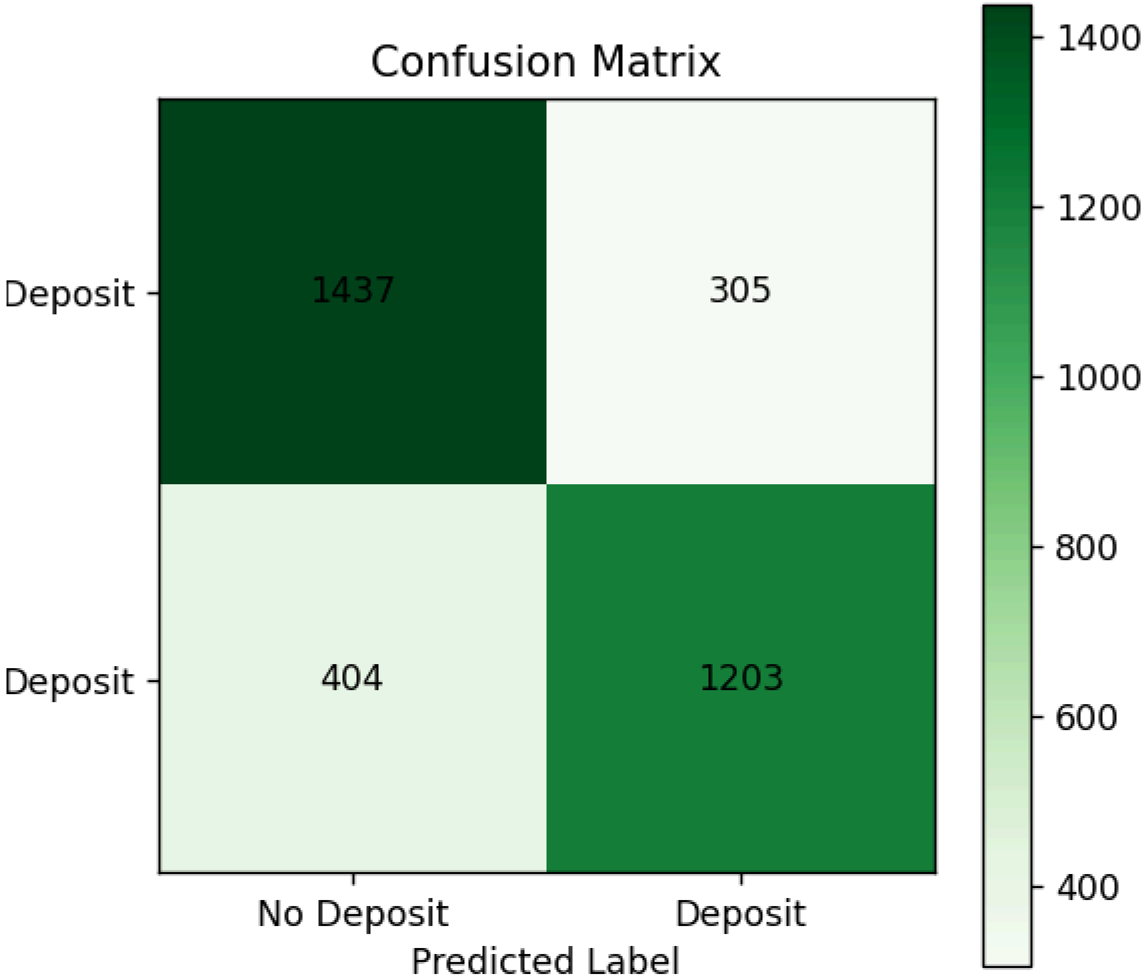
Logistic Regression

Code:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder, StandardScaler

df = pd.read_csv("bank.csv")

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

df['deposit'] = df['deposit'].map({'yes': 1, 'no': 0})

le = LabelEncoder()

for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])

X = df.drop('deposit', axis=1)
y = df['deposit']


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```python


y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print("\nAccuracy:", round(accuracy * 100, 2), "%")
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))


plt.figure(figsize=(5,5))
plt.imshow(cm, cmap='Greens')

plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")

for i in range(len(cm)):
    for j in range(len(cm)):
        plt.text(j, i, cm[i, j], ha='center', va='center')

plt.xticks([0,1], ['No Deposit', 'Deposit'])
plt.yticks([0,1], ['No Deposit', 'Deposit'])

plt.colorbar()
plt.show()
```

Output:




```
Accuracy: 78.83 %

Confusion Matrix:
 [[1437  305]
 [ 404 1203]]
```

```
Classification Report:
              precision    recall  f1-score

           0       0.78      0.82      0.80
           1       0.80      0.75      0.77
```