**Aim**: **:-** Perform Regression Analysis using Scipy and Sci-kit learn.

**Objective**:

- a. Perform Logistic Regression to find relationships between variables.
- b. Apply regression model techniques to predict data.

Dataset Description:
Big Data
14+ columns

**age**: The age of the individual.
**workclass**: The type of employment (e.g., private, self-employed, government).
**fnlwgt**: Final weight, representing the number of people the individual represents.
**education**: The highest level of education achieved.
**education-num**: The number of years of education completed.
**marital-status**: The marital status of the individual (e.g., married, single).
**occupation**: The type of job or occupation.
**relationship**: The individual's relationship status within a household (e.g., husband, wife).
**race**: The race of the individual.
**sex**: The gender of the individual.
**capital-gain**: Income from investment sources other than salary/wages.
**capital-loss**: Losses from investment sources other than salary/wages.
**hours-per-week**: The number of hours worked per week.
**native-country**: The country of origin.
**income**: The income level (<=50K or >50K).

Step 1: Load the Dataset

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names
```

+ Code + Text

```
[ ] import pandas as pd

    # Define column names
    columns = [
        'age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
        'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
        'hours-per-week', 'native-country', 'income'
    ]

    # Load the dataset
    df = pd.read_csv('adult.data', names=columns, na_values=' ?', skipinitialspace=True)
    print(df.shape)
    df.head()
```

Step 2: Preprocess the Data

```
[ ] df = df.dropna()
    print(df.shape)
```

```
(32561, 15)
```

```
[ ] df['income'] = df['income'].map({'<=50K': 0, '>50K': 1})
    df['income'].value_counts()
```

| income | count |
|---|---|
| 0 | 24720 |
| 1 | 7841 |

dtype: int64

```
# Select features
features = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week',
            'workclass', 'education', 'marital-status', 'occupation', 'sex']
X = df[features]
y = df['income']
```

```
[ ] X = pd.get_dummies(X, columns=['workclass', 'education', 'marital-status', 'occupation', 'sex'], drop_first=True)
    print(X.shape)  # Now has more columns due to dummy variables
    X.head()
```

| _Priv-e-serv | occupation_Prof-specialty | occupation_Protective-serv | occupation_Sales | occupation_Tech-support | occupation_Transport-moving | sex_Male |
|---|---|---|---|---|---|---|
| False | False | False | False | False | False | True |

**Missing Values:** Real-world data often has gaps. Dropping rows is simple but reduces data (alternatives: imputation).

**Target Encoding**: Logistic Regression needs a numerical target. We mapped <=50K to 0 and >50K to 1 for binary classification.

**One-Hot Encoding**: Categorical variables (e.g., occupation) can't be used directly in math-based models. get_dummies converts them to binary columns (e.g., occupation_Exec-managerial: 1 if true, 0 if not). drop_first=True avoids multicollinearity (dummy variable trap).

X is the feature matrix (inputs), y is the target vector (output).

**Step 3:** Splitting the dataset.

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

Train-Test Split: We train on one subset (X_train, y_train) and evaluate on another (X_test, y_test) to test generalization.

Random State: Fixes the random seed for reproducibility (same split every time).

**Step 4**: Scale the Data.

```
# Scale only numerical columns
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])

# Train the model on scaled data
```

initial error:

```
from sklearn.linear_model import LogisticRegression

# Initialize and fit the model
model = LogisticRegression(max_iter=5000)  # Increase max_iter if it doesn't converge
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to c
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression
LogisticRegression(max_iter=5000)
```

StandardScaler: Transforms features to have mean=0, standard deviation=1 using (x-mean)/std. This puts all numerical features on the same scale.

Logistic Regression uses gradient descent to optimize coefficients. Unscaled features (e.g., capital-gain 0–99999 vs. age 17–90) make convergence slow or impossible.

Fit vs. Transform: fit_transform on training data learns the scaling parameters (mean, std); transform on test data applies them without relearning (avoids data leakage).

**Step 5**: Train the Logistic Regression Model

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

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

# Define numerical columns to scale
num_columns = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

# Initialize scaler
scaler = StandardScaler()

# Scale only numerical columns
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])

# Train the model on scaled data
model = LogisticRegression(max_iter=1000)  # Should converge now
model.fit(X_train_scaled, y_train)
```

**Logistic Regression**: A linear model for binary classification. It predicts the probability of a class (e.g., P(>50K)) using the logistic function:

P(y=1) = 1 / (1 + exp(-(b0 + b1*x1 + b2*x2 + ...)))

- b0: Intercept, bi: Coefficients for each feature xi

**Step 6**: Make Predictions and Evaluate

```
# Make predictions
y_pred = model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.86
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4942
           1       0.75      0.61      0.67      1571

    accuracy                           0.86      6513
   macro avg       0.82      0.77      0.79      6513
weighted avg       0.85      0.86      0.85      6513
```

- **Prediction**: predict outputs class labels (0 or 1) by thresholding probabilities at 0.5 (P>0.5 → 1).
- **Accuracy**: Fraction of correct predictions (simple but can mislead if classes are imbalanced).
- **Classification Report**: Precision (correct positive predictions), recall (true positives caught), F1-score (balance of precision/recall).

**Step 7:** Analyze Relationships.

```python
# feature names and coefficients
feature_names = X.columns
coefficients = model.coef_[0]

# DataFrame for interpretation
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})
coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
print(coef_df.head(10))
print(coef_df.tail(10))
```

```
                            Feature  Coefficient
2                      capital-gain     2.246447
28   marital-status_Married-AF-spouse     2.203413
29  marital-status_Married-civ-spouse     2.168329
37         occupation_Exec-managerial     1.070328
46            occupation_Tech-support     0.957079
5               workclass_Federal-gov     0.948002
44         occupation_Protective-serv     0.837816
1                     education-num     0.787863
43          occupation_Prof-specialty     0.770856
9             workclass_Self-emp-inc     0.601304
                        Feature  Coefficient
35       occupation_Armed-Forces    -0.187277
32      marital-status_Separated    -0.220684
39  occupation_Handlers-cleaners    -0.284757
19           education_Assoc-acdm    -0.385632
31  marital-status_Never-married    -0.500890
41      occupation_Other-service    -0.504615
12          workclass_Without-pay    -0.513856
25            education_Preschool    -0.657225
38    occupation_Farming-fishing    -0.843326
42    occupation_Priv-house-serv    -1.391624
```

- **Coefficients**: Measure feature impact on log-odds. Positive bi increases P(>50K); negative decreases it. Magnitude shows strength.
- **Interpretation**: After scaling, coefficients are comparable across features (e.g., 1 unit change in education-num VS Capital-gain)..

# Linear regression.

## Step 1: Load and Preprocess

```
import pandas as pd


columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
           'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
           'hours-per-week', 'native-country', 'income']
df = pd.read_csv('adult.data', names=columns, na_values=' ?', skipinitialspace=True)

# Drop miss
df = df.dropna()

# Define features, remove hours-per-week
features = ['age', 'education-num', 'capital-gain', 'capital-loss',
            'workclass', 'education', 'marital-status', 'occupation', 'sex']
X = df[features]
y = df['hours-per-week']  # New target

# One-hot encode categorical variables
X = pd.get_dummies(X, columns=['workclass', 'education', 'marital-status', 'occupation', 'sex'], drop_first=True)
```

hours-per-week is now y (what we predict). We removed it from X to avoid using the target as a feature.

## Step 2: Split the Data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## Step 3: Scale the Features

```
from sklearn.preprocessing import StandardScaler

# Numerical columns
num_columns = ['age', 'education-num', 'capital-gain', 'capital-loss']

# Scale
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])
```

Linear Regression also benefits from scaled features (like Logistic Regression) for faster convergence and fair coefficient comparison. Dummy variables stay 0/1.

## Step 4: Train Linear Regression

```
[ ]  from sklearn.linear_model import LinearRegression

     # Initialize and train
     lin_model = LinearRegression()
     lin_model.fit(X_train_scaled, y_train)

     # Predict
     y_pred = lin_model.predict(X_test_scaled)
```

Linear Regression fits a line: y = b0 + b1*x1 + b2*x2 + ....

- b0: Intercept (base hours if all features are 0).
- bi: Coefficients (how much each feature changes hours).
- Predicts continuous values (e.g., 38.7 hours).

### Step 5: Calculate MSE and $R^2$

```
from sklearn.metrics import mean_squared_error, r2_score

# MSE
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")

# R²
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2:.4f}")
```

```
Mean Squared Error: 127.1157
R² Score: 0.1747
```

**MSE**: ~100–150 (hours^2, since `hours-per-week` ranges 1–99).

**RMSE**: ~10–12 hours (square root of MSE, in hours).

**$R^2$**: ~0.20–0.30 (moderate fit—hours worked vary a lot beyond these features).

### Step 6: Analyze Relationships

```
[ ] feature_names = X.columns
    coefficients = lin_model.coef_
    coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})
    coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
    print("Top 10 Positive Influences:")
    print(coef_df.head(10))
    print("\nTop 10 Negative Influences:")
    print(coef_df.tail(10))
```

```
Top 10 Positive Influences:
                        Feature  Coefficient
8        workclass_Self-emp-inc    10.009662
37    occupation_Farming-fishing     7.031947
9     workclass_Self-emp-not-inc     5.412741
46  occupation_Transport-moving     5.273542
4          workclass_Federal-gov     4.951338
7              workclass_Private     4.696466
36    occupation_Exec-managerial     4.473448
5            workclass_Local-gov     4.472043
24           education_Preschool     3.899929
43    occupation_Protective-serv     3.865683

Top 10 Negative Influences:
                                 Feature  Coefficient
41             occupation_Priv-house-serv    -1.088628
29  marital-status_Married-spouse-absent    -1.281585
13                        education_12th    -1.497008
40              occupation_Other-service    -1.844769
27      marital-status_Married-AF-spouse    -2.386385
12                        education_11th    -3.050668
6                 workclass_Never-worked    -3.298040
11                 workclass_Without-pay    -4.307680
30          marital-status_Never-married    -4.695433
32                marital-status_Widowed    -4.975283
```

**Conclusion:**

From this experiment, we have learned about:

- How to apply logistic regression to classify income levels based on various demographic features.
- How regression models can predict income based on independent variables like age, education, work hours.
- Importance of Regression techniques when applied on real world data sets help to gain valuable insights.
- How we can perform linear regression to find the number of hours worked given other independent attributes.

However, the moderate $R^2$ (24%) and RMSE (11.65 hours) suggest limitations. Hours worked are influenced by factors beyond our dataset—personal choice, industry norms, or unrecorded variables—leading to a model that captures only a portion of the variability. The custom accuracy of ~68% within ±5 hours, yet the RMSE indicates some predictions deviate more significantly, reflecting the challenge of predicting a highly variable human behavior like work hours.

In conclusion, this Linear Regression experiment not only achieved its technical goals but also deepened our understanding of data science workflows—preprocessing, modeling, predicting, and evaluating—all while adapting to a new target that better suits regression's strengths.