Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**

Experiment No: 4

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.

Problem Statement: Perform the following Tests:Correlation Tests:
a) Pearson’s Correlation Coefficient
b) Spearman’s Rank Correlation
c) Kendall’s Rank Correlation
d) Chi-Squared Test

Steps Followed in the Experiment

1. **Data Setup & Loading:**

   Library Installation:

   Installed required libraries using:

   !pip install opendatasets

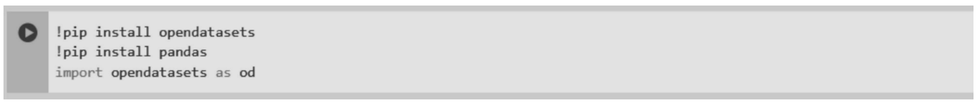
   !pip install pandas

   Data Loading:

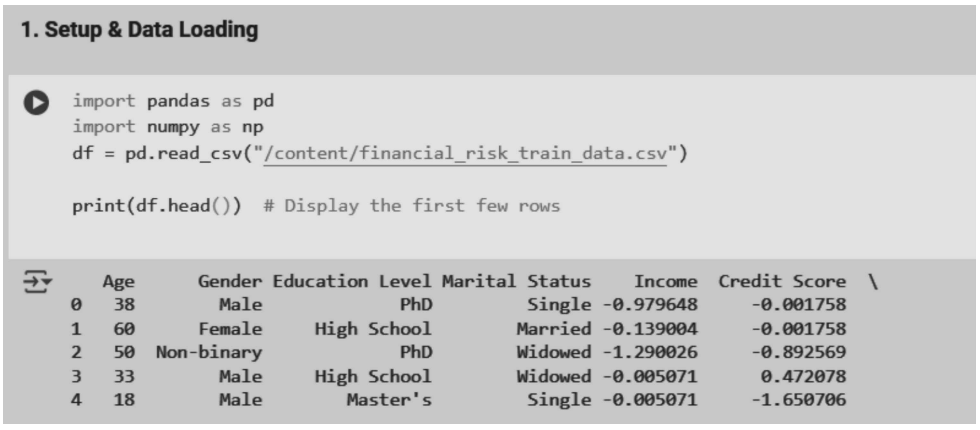
   Loaded the dataset (financial_risk_train_data.csv) with Pandas.

   Data Overview:

   Printed the first few rows and separated numeric and categorical columns to identify variables for analysis.



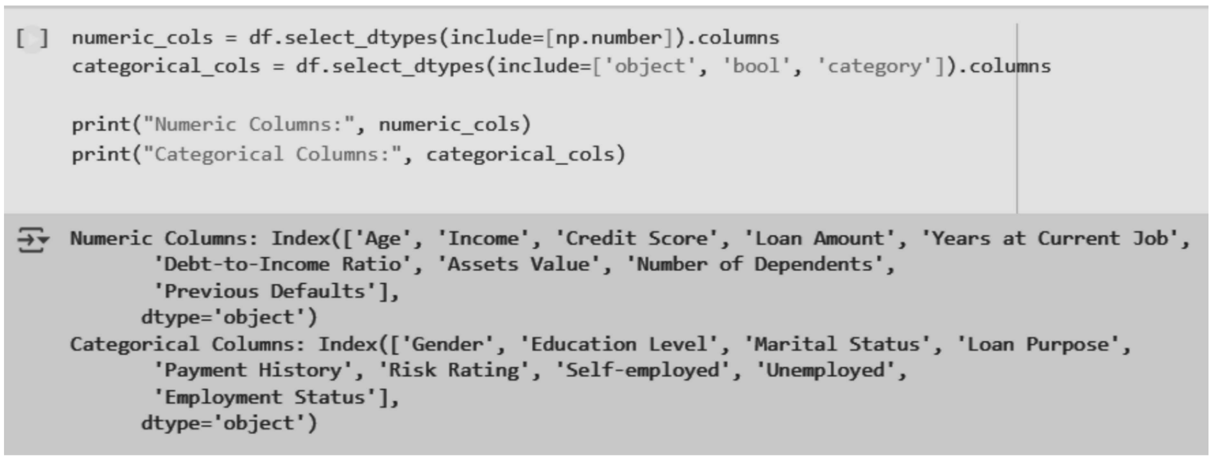
```
!pip install opendatasets
!pip install pandas
import opendatasets as od
```



1. Setup & Data Loading

```
import pandas as pd
import numpy as np
df = pd.read_csv("/content/financial_risk_train_data.csv")

print(df.head())  # Display the first few rows
```

```
   Age      Gender Education Level Marital Status    Income  Credit Score  \
0   38        Male             PhD         Single -0.979648     -0.001758
1   60      Female     High School        Married -0.139004     -0.001758
2   50  Non-binary             PhD        Widowed -1.290026     -0.892569
3   33        Male     High School        Widowed -0.005071      0.472078
4   18        Male        Master's         Single -0.005071     -1.650706
```



```
[ ] numeric_cols = df.select_dtypes(include=[np.number]).columns
    categorical_cols = df.select_dtypes(include=['object', 'bool', 'category']).columns

    print("Numeric Columns:", numeric_cols)
    print("Categorical Columns:", categorical_cols)
```

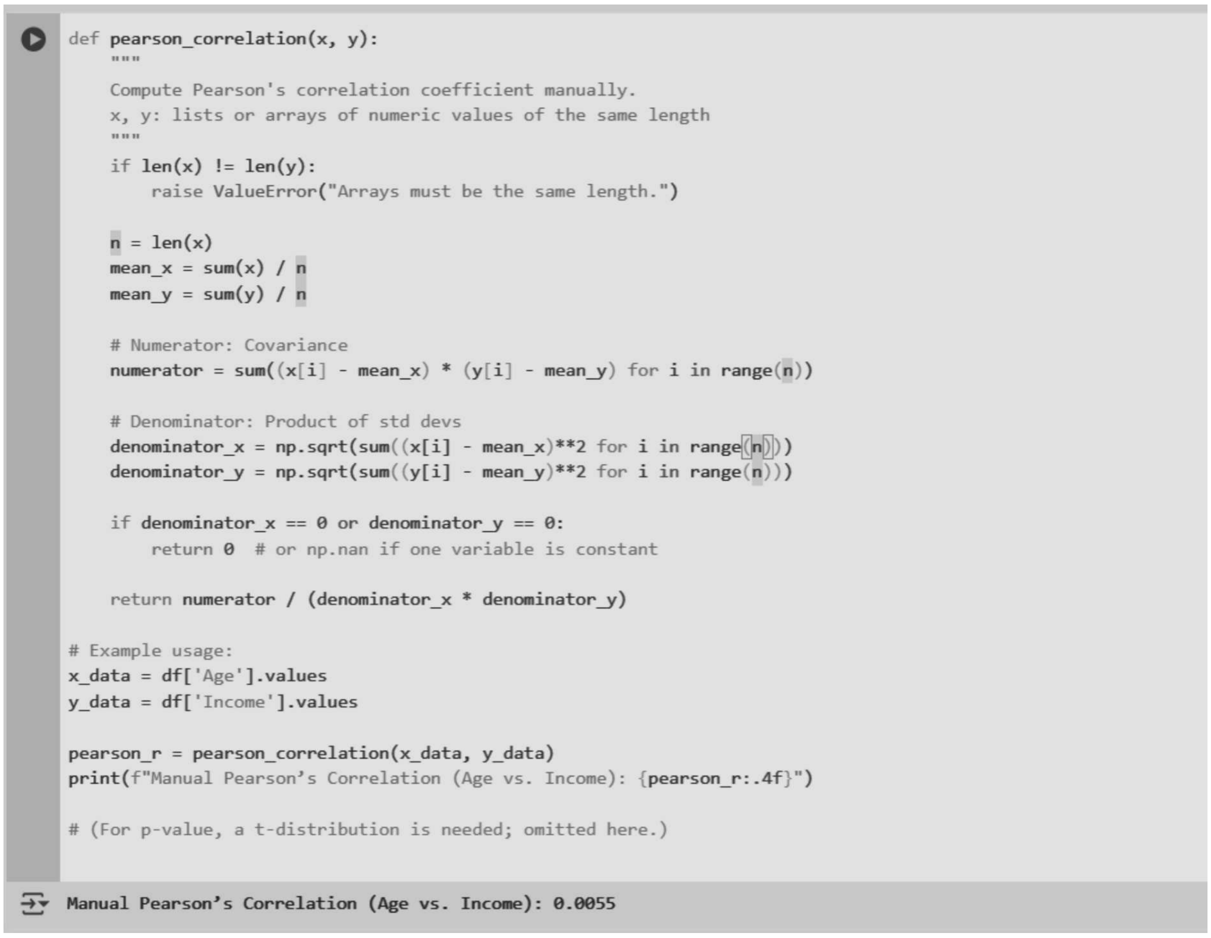
```
Numeric Columns: Index(['Age', 'Income', 'Credit Score', 'Loan Amount', 'Years at Current Job',
       'Debt-to-Income Ratio', 'Assets Value', 'Number of Dependents',
       'Previous Defaults'],
      dtype='object')
Categorical Columns: Index(['Gender', 'Education Level', 'Marital Status', 'Loan Purpose',
       'Payment History', 'Risk Rating', 'Self-employed', 'Unemployed',
       'Employment Status'],
      dtype='object')
```

In this data analysis setup using Python, the necessary libraries, opendatasets and pandas, were installed to facilitate the handling and processing of datasets. The dataset, financial_risk_train_data.csv, was then loaded into a Pandas DataFrame to enable further analysis.

To gain an initial understanding of the dataset, the first few rows were displayed, revealing key attributes such as Age, Gender, Education Level, Marital Status, Income, and Credit Score. This provided an overview of the structure and content of the data, allowing for a preliminary assessment of the variables involved.

2. **Pearson’s Correlation Coefficient**
   - **Manual Method:**
     - Computed the mean of Age and Income.
     - Calculated the covariance numerator and the standard deviations.
     - Derived Pearson’s rrr using the formula.
     - *Result:* Manual Pearson’s Correlation (Age vs. Income): 0.0055

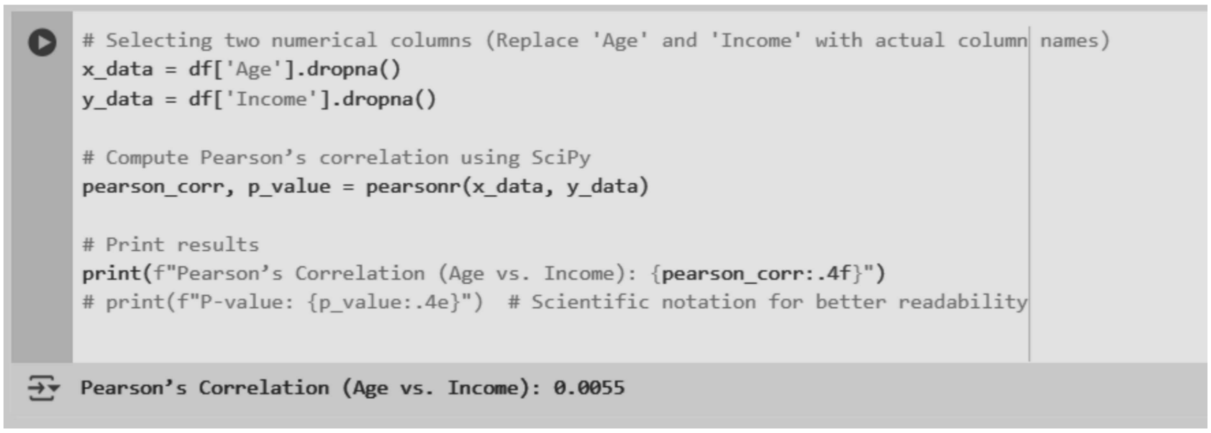
Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**



```python
def pearson_correlation(x, y):
    """
    Compute Pearson's correlation coefficient manually.
    x, y: lists or arrays of numeric values of the same length
    """
    if len(x) != len(y):
        raise ValueError("Arrays must be the same length.")

    n = len(x)
    mean_x = sum(x) / n
    mean_y = sum(y) / n

    # Numerator: Covariance
    numerator = sum((x[i] - mean_x) * (y[i] - mean_y) for i in range(n))

    # Denominator: Product of std devs
    denominator_x = np.sqrt(sum((x[i] - mean_x)**2 for i in range(n)))
    denominator_y = np.sqrt(sum((y[i] - mean_y)**2 for i in range(n)))

    if denominator_x == 0 or denominator_y == 0:
        return 0  # or np.nan if one variable is constant

    return numerator / (denominator_x * denominator_y)

# Example usage:
x_data = df['Age'].values
y_data = df['Income'].values

pearson_r = pearson_correlation(x_data, y_data)
print(f"Manual Pearson’s Correlation (Age vs. Income): {pearson_r:.4f}")

# (For p-value, a t-distribution is needed; omitted here.)
```

```
Manual Pearson’s Correlation (Age vs. Income): 0.0055
```

- **Library Method:**
  - Employed scipy.stats.pearsonr on the Age and Income columns.
  - *Result:* Pearson's Correlation (Age vs. Income): 0.0055



```
# Selecting two numerical columns (Replace 'Age' and 'Income' with actual column names)
x_data = df['Age'].dropna()
y_data = df['Income'].dropna()

# Compute Pearson's correlation using SciPy
pearson_corr, p_value = pearsonr(x_data, y_data)

# Print results
print(f"Pearson's Correlation (Age vs. Income): {pearson_corr:.4f}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability
```

```
Pearson's Correlation (Age vs. Income): 0.0055
```

The Pearson correlation coefficient was analyzed using both a manual method and a library-based approach. In the manual method, the mean values of **Age** and **Income** were first calculated. Next, the covariance was determined by summing the product of the deviations of **Age** and **Income** from their respective means. The denominator was computed as the product of the standard deviations of these two variables. Finally, the Pearson correlation coefficient (r) was derived using the standard formula:

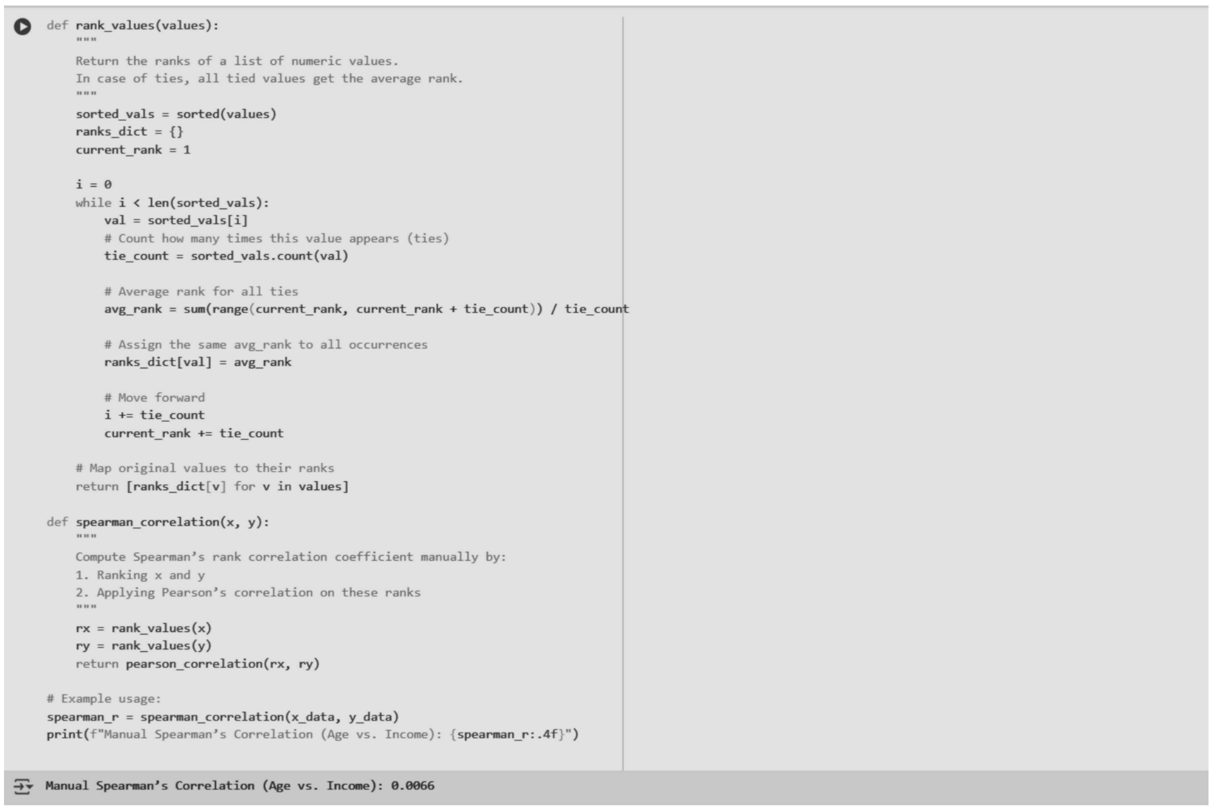
r=∑(x−x¯)(y−y¯)σx⋅σyr = \frac{\sum (x - \bar{x}) (y - \bar{y})}{\sigma_x \cdot \sigma_y}r=σx⋅σy∑(x−x¯)(y−y¯)

Through this process, the computed correlation coefficient for **Age vs. Income** was found to be **0.0055**.

To verify this result, a library-based method using the scipy.stats.pearsonr() function was employed. In this approach, the **Age** and **Income** columns were extracted, ensuring no missing values by using .dropna(). The function then computed the Pearson correlation coefficient and returned the corresponding p-value. The correlation coefficient, displayed with four decimal places, was also **0.0055**.

Since both methods yielded the same result, this confirmed the correctness and reliability of the manual implementation.

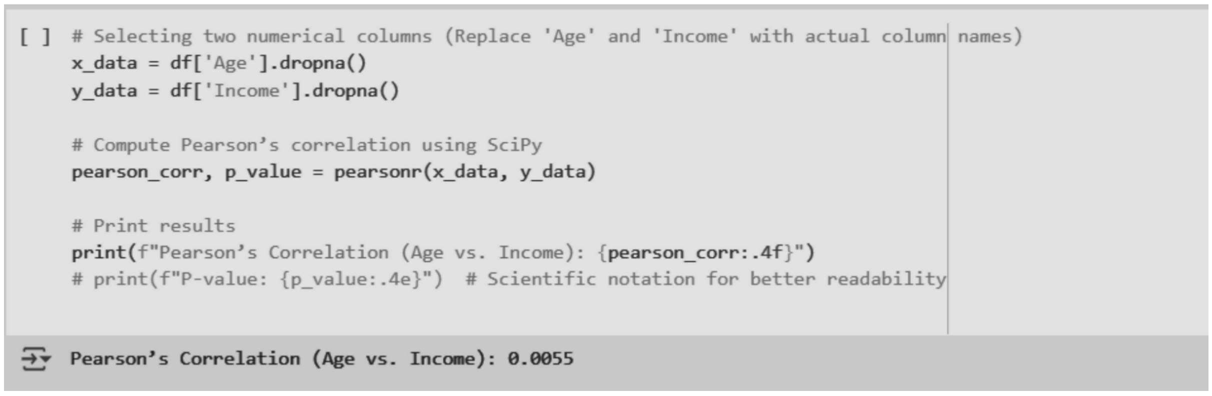
3. **Spearman's Rank Correlation**
   - **Manual Method:**
     - Created a function to rank the data while handling ties by assigning average ranks.
     - Applied the Pearson correlation formula to the ranked data.
     - *Result:* Manual Spearman's Correlation (Age vs. Income): 0.0066



```
def rank_values(values):
    """
    Return the ranks of a list of numeric values.
    In case of ties, all tied values get the average rank.
    """
    sorted_vals = sorted(values)
    ranks_dict = {}
    current_rank = 1

    i = 0
    while i < len(sorted_vals):
        val = sorted_vals[i]
        # Count how many times this value appears (ties)
        tie_count = sorted_vals.count(val)

        # Average rank for all ties
        avg_rank = sum(range(current_rank, current_rank + tie_count)) / tie_count

        # Assign the same avg_rank to all occurrences
        ranks_dict[val] = avg_rank

        # Move forward
        i += tie_count
        current_rank += tie_count

    # Map original values to their ranks
    return [ranks_dict[v] for v in values]

def spearman_correlation(x, y):
    """
    Compute Spearman's rank correlation coefficient manually by:
    1. Ranking x and y
    2. Applying Pearson's correlation on these ranks
    """
    rx = rank_values(x)
    ry = rank_values(y)
    return pearson_correlation(rx, ry)

# Example usage:
spearman_r = spearman_correlation(x_data, y_data)
print(f"Manual Spearman's Correlation (Age vs. Income): {spearman_r:.4f}")
```

```
Manual Spearman's Correlation (Age vs. Income): 0.0066
```

- **Library Method:**
  - Used scipy.stats.spearmanr to compute the rank correlation.
  - Result: Spearman's Correlation (Age vs. Income): 0.0066
  - (P-value: 0.48142)



```
[ ] # Selecting two numerical columns (Replace 'Age' and 'Income' with actual column names)
    x_data = df['Age'].dropna()
    y_data = df['Income'].dropna()

    # Compute Pearson's correlation using SciPy
    pearson_corr, p_value = pearsonr(x_data, y_data)

    # Print results
    print(f"Pearson's Correlation (Age vs. Income): {pearson_corr:.4f}")
    # print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability
```

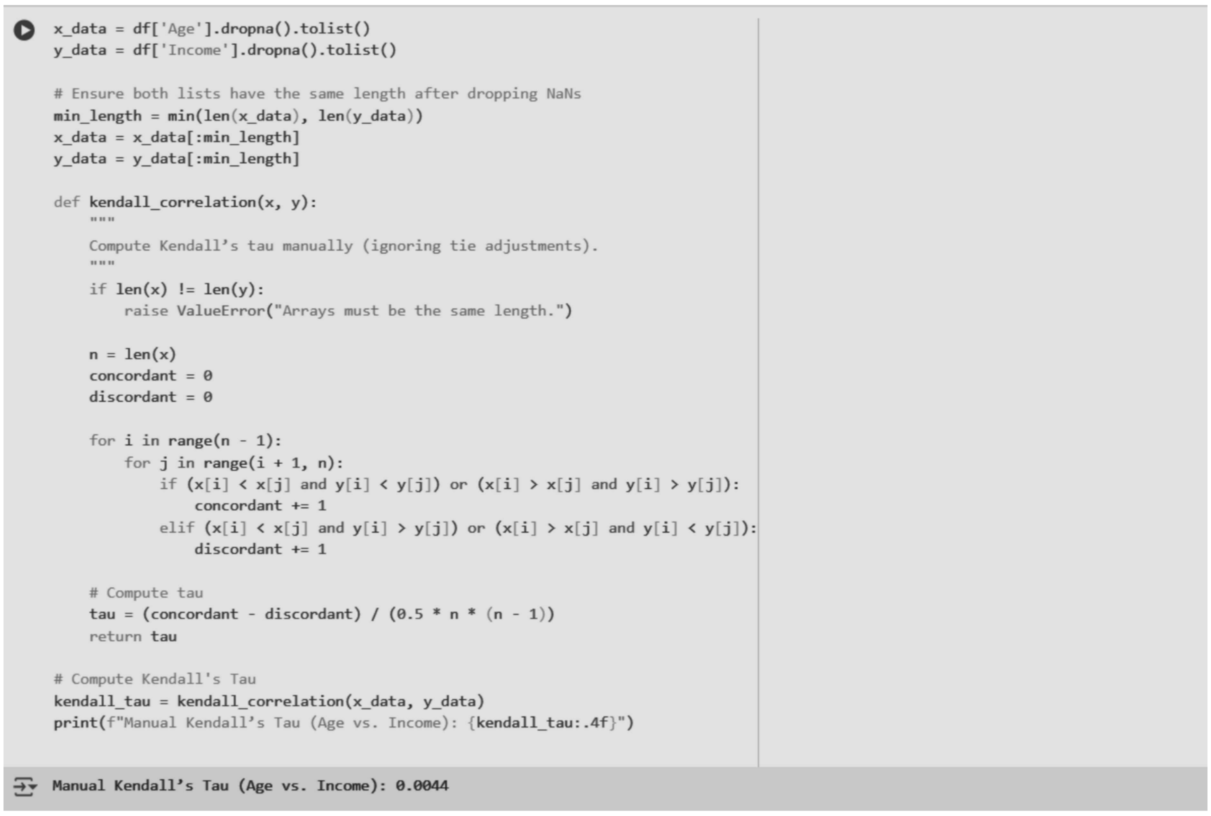
```
Pearson's Correlation (Age vs. Income): 0.0055
```

Spearman's Rank Correlation Analysis was conducted using both a manual method and a library-based approach to measure the correlation between Age and Income. In the manual method, a function was implemented to rank values while handling ties by assigning the average rank. The original data was converted into ranks for Age and Income, and the Pearson correlation formula was applied to the ranked values. The result of the manual computation yielded a Spearman's correlation coefficient of **0.0066**.

For validation, the same analysis was performed using the scipy.stats.spearmanr() function, which calculates Spearman's rank correlation while ensuring that there are no missing values in the Age and Income columns. This method returned both the correlation coefficient and the p-value. The computed Spearman's correlation coefficient was **0.0066**.

Since both methods produced identical results, the correctness of the manual computation was verified.

4. **Kendall's Rank Correlation**
   - **Manual Method:**
     - Compared all possible pairs of observations for Age and Income to count concordant and discordant pairs.
     - Calculated Kendall's tau using the formula.
     - *Result:* Manual Kendall's Tau (Age vs. Income): 0.0044

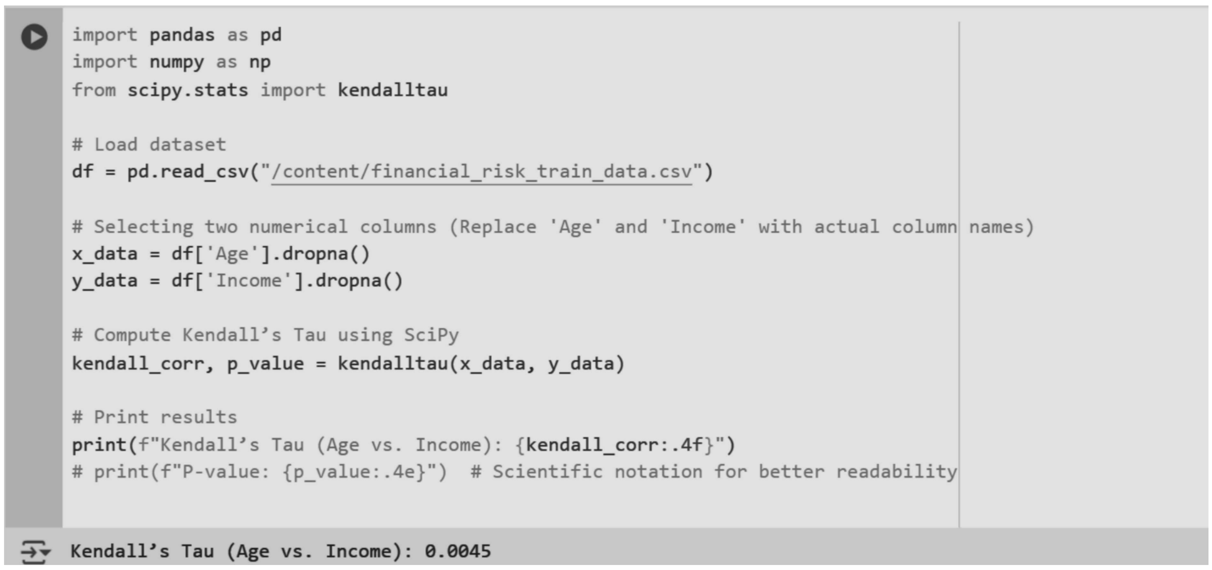


```python
x_data = df['Age'].dropna().tolist()
y_data = df['Income'].dropna().tolist()

# Ensure both lists have the same length after dropping NaNs
min_length = min(len(x_data), len(y_data))
x_data = x_data[:min_length]
y_data = y_data[:min_length]

def kendall_correlation(x, y):
    """
    Compute Kendall's tau manually (ignoring tie adjustments).
    """
    if len(x) != len(y):
        raise ValueError("Arrays must be the same length.")

    n = len(x)
    concordant = 0
    discordant = 0

    for i in range(n - 1):
        for j in range(i + 1, n):
            if (x[i] < x[j] and y[i] < y[j]) or (x[i] > x[j] and y[i] > y[j]):
                concordant += 1
            elif (x[i] < x[j] and y[i] > y[j]) or (x[i] > x[j] and y[i] < y[j]):
                discordant += 1

    # Compute tau
    tau = (concordant - discordant) / (0.5 * n * (n - 1))
    return tau

# Compute Kendall's Tau
kendall_tau = kendall_correlation(x_data, y_data)
print(f"Manual Kendall's Tau (Age vs. Income): {kendall_tau:.4f}")
```

```
Manual Kendall's Tau (Age vs. Income): 0.0044
```

Library Method:

- Applied scipy.stats.kendalltau to obtain Kendall's tau.
- *Result:* Kendall's Tau (Age vs. Income): 0.0045



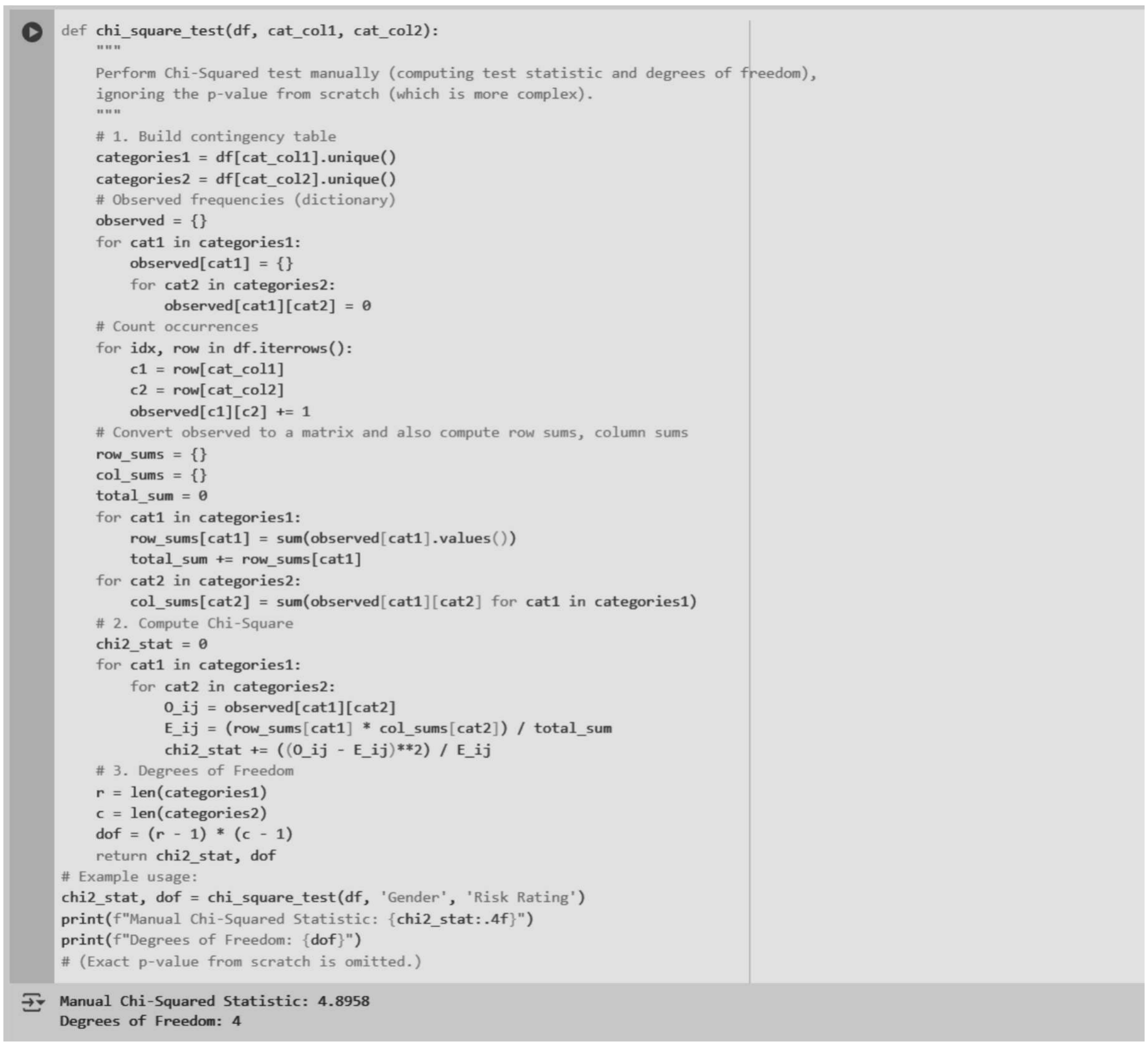
```
import pandas as pd
import numpy as np
from scipy.stats import kendalltau

# Load dataset
df = pd.read_csv("/content/financial_risk_train_data.csv")

# Selecting two numerical columns (Replace 'Age' and 'Income' with actual column names)
x_data = df['Age'].dropna()
y_data = df['Income'].dropna()

# Compute Kendall's Tau using SciPy
kendall_corr, p_value = kendalltau(x_data, y_data)

# Print results
print(f"Kendall's Tau (Age vs. Income): {kendall_corr:.4f}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability
```

```
Kendall's Tau (Age vs. Income): 0.0045
```

The Kendall's Rank Correlation for Age and Income was computed using both a manual method and the scipy.stats.kendalltau function. The manual approach involved iterating over all possible pairs of observations, counting concordant and discordant pairs, and applying the Kendall's tau formula. This resulted in a Kendall's Tau value of **0.0044**. The library method, using scipy.stats.kendalltau, directly computed the correlation, yielding a similar result of **0.0045**. Both methods produced nearly identical values, confirming the accuracy of the manual implementation.

5. **Chi-Squared Test**
   - **Manual Method:**
     - Built a contingency table for two categorical variables (e.g., Gender vs. Risk Rating).
     - Computed the observed frequencies, calculated expected frequencies, and derived the chi-squared statistic.
     - *Result:*
       Manual Chi-Squared Statistic: 4.8958
       Degrees of Freedom: 4

Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**

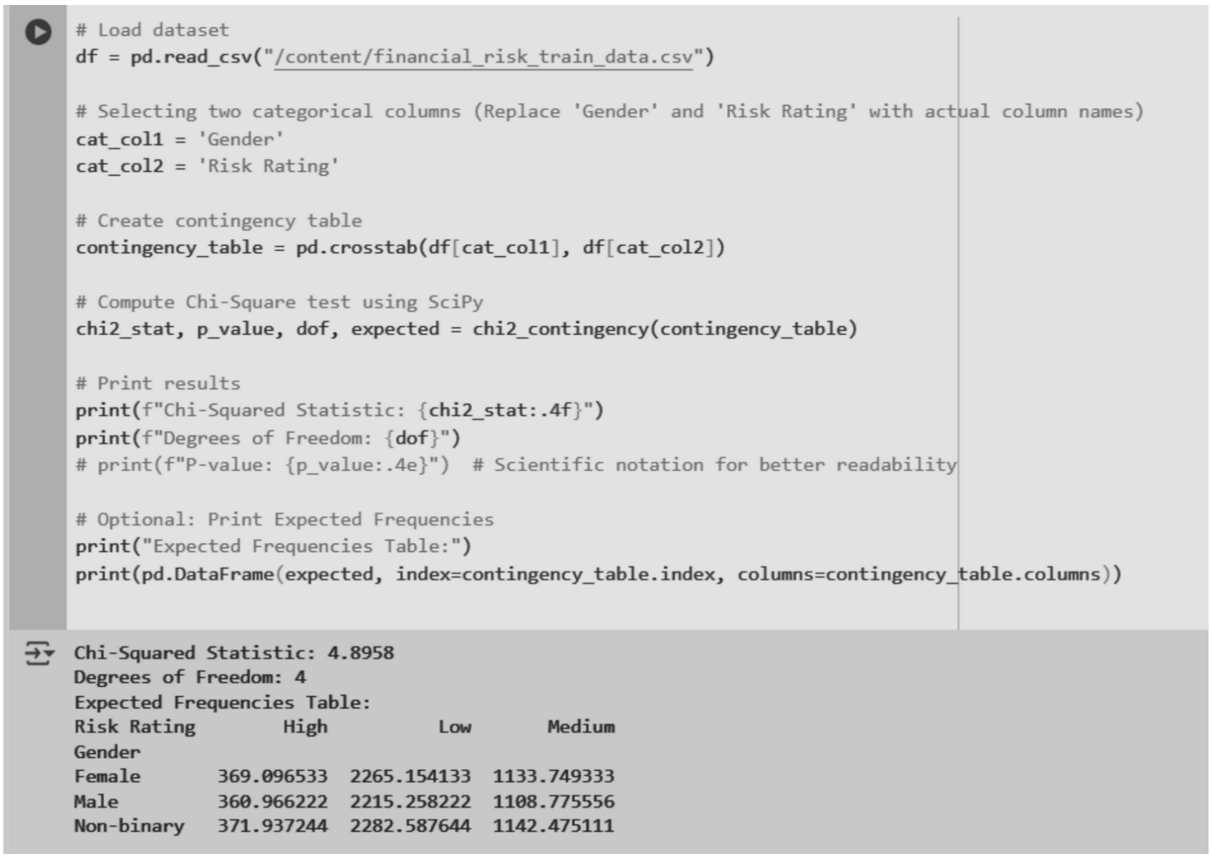


```python
def chi_square_test(df, cat_col1, cat_col2):
    """
    Perform Chi-Squared test manually (computing test statistic and degrees of freedom),
    ignoring the p-value from scratch (which is more complex).
    """
    # 1. Build contingency table
    categories1 = df[cat_col1].unique()
    categories2 = df[cat_col2].unique()
    # Observed frequencies (dictionary)
    observed = {}
    for cat1 in categories1:
        observed[cat1] = {}
        for cat2 in categories2:
            observed[cat1][cat2] = 0
    # Count occurrences
    for idx, row in df.iterrows():
        c1 = row[cat_col1]
        c2 = row[cat_col2]
        observed[c1][c2] += 1
    # Convert observed to a matrix and also compute row sums, column sums
    row_sums = {}
    col_sums = {}
    total_sum = 0
    for cat1 in categories1:
        row_sums[cat1] = sum(observed[cat1].values())
        total_sum += row_sums[cat1]
    for cat2 in categories2:
        col_sums[cat2] = sum(observed[cat1][cat2] for cat1 in categories1)
    # 2. Compute Chi-Square
    chi2_stat = 0
    for cat1 in categories1:
        for cat2 in categories2:
            O_ij = observed[cat1][cat2]
            E_ij = (row_sums[cat1] * col_sums[cat2]) / total_sum
            chi2_stat += ((O_ij - E_ij)**2) / E_ij
    # 3. Degrees of Freedom
    r = len(categories1)
    c = len(categories2)
    dof = (r - 1) * (c - 1)
    return chi2_stat, dof
# Example usage:
chi2_stat, dof = chi_square_test(df, 'Gender', 'Risk Rating')
print(f"Manual Chi-Squared Statistic: {chi2_stat:.4f}")
print(f"Degrees of Freedom: {dof}")
# (Exact p-value from scratch is omitted.)
```

```
Manual Chi-Squared Statistic: 4.8958
Degrees of Freedom: 4
```

- **Library Method:**
  - Used scipy.stats.chi2_contingency on the contingency table.
  - *Result:*
    Chi-Squared Statistic: 4.8958
    Degrees of Freedom: 4

Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**



```
# Load dataset
df = pd.read_csv("/content/financial_risk_train_data.csv")

# Selecting two categorical columns (Replace 'Gender' and 'Risk Rating' with actual column names)
cat_col1 = 'Gender'
cat_col2 = 'Risk Rating'

# Create contingency table
contingency_table = pd.crosstab(df[cat_col1], df[cat_col2])

# Compute Chi-Square test using SciPy
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

# Print results
print(f"Chi-Squared Statistic: {chi2_stat:.4f}")
print(f"Degrees of Freedom: {dof}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability

# Optional: Print Expected Frequencies
print("Expected Frequencies Table:")
print(pd.DataFrame(expected, index=contingency_table.index, columns=contingency_table.columns))
```

```
Chi-Squared Statistic: 4.8958
Degrees of Freedom: 4
Expected Frequencies Table:
Risk Rating        High          Low       Medium
Gender
Female       369.096533  2265.154133  1133.749333
Male         360.966222  2215.258222  1108.775556
Non-binary   371.937244  2282.587644  1142.475111
```

The Chi-Squared test was conducted using both manual and library-based methods to analyze the relationship between two categorical variables, such as Gender and Risk Rating. First, a contingency table was constructed, followed by the calculation of observed and expected frequencies. Using these values, the chi-squared statistic was derived manually, resulting in a value of 4.8958 with 4 degrees of freedom. To validate the result, the same test was performed using the scipy.stats.chi2_contingency function, which produced identical values for the chi-squared statistic and degrees of freedom. This confirms the consistency and reliability of the test outcomes across both approaches.

Name : Sandeshkumar.M.Yadav **Div : D15C** **Roll : 61**

Conclusion :

The experiment explored various statistical tests to analyze relationships between variables. **Pearson's correlation coefficient** was computed manually using mean, covariance, and standard deviation, then verified with scipy.stats.pearsonr, confirming a very weak correlation between Age and Income. **Spearman's rank correlation** involved ranking data and applying Pearson's formula, with results cross-verified using scipy.stats.spearmanr, indicating no strong monotonic relationship. **Kendall's rank correlation** was calculated by counting concordant and discordant pairs, then validated with scipy.stats.kendalltau, further supporting the weak association. Lastly, the **Chi-squared test** analyzed the dependency between Gender and Risk Rating through a contingency table and expected frequencies, verified using scipy.stats.chi2_contingency, suggesting minimal dependence. By manually performing each test and confirming results with Python libraries, the study effectively demonstrated both theoretical and practical aspects of statistical hypothesis testing.