

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

Section 1: Data Loading & Setup

1. Data Loading

- Loaded the dataset (fifa_eda_stats.csv) into a Pandas DataFrame.
- Displayed the first few rows to understand the data structure.

2. Column Identification

- Separated **numeric columns** (e.g., Age, Overall, Potential) and **categorical columns** (e.g., Name, Nationality, Position) for later analyses.

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv("/content/fifa_eda_stats.csv")
print("Dataset preview:")
print(df.head())
```

```
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)
```

```

Dataset preview:
   ID      Name  Age Nationality Overall Potential \
0  158023    L. Messi  31   Argentina    94      94
1  20801  Cristiano Ronaldo  33    Portugal    94      94
2  190871   Neymar Jr  26    Brazil    92      93
3  193080    De Gea  27    Spain    91      93
4  192985   K. De Bruyne  27    Belgium    91      92

   Club      Value   Wage Preferred Foot ... Composure \
0   FC Barcelona €110.5M €565K      Left ...    96.0
1    Juventus    €77M   €405K      Right ...    95.0
2 Paris Saint-Germain €118.5M €290K      Right ...    94.0
3 Manchester United   €72M   €260K      Right ...    68.0
4 Manchester City   €102M   €355K      Right ...    88.0

   Marking StandingTackle SlidingTackle GKDividing GKHandling GKKicking \
0    33.0         28.0         26.0      6.0      11.0      15.0
1    28.0         31.0         23.0      7.0      11.0      15.0
2    27.0         24.0         33.0      9.0      9.0      15.0
3    15.0         21.0         13.0     90.0     85.0     87.0
4    68.0         58.0         51.0     15.0     13.0      5.0

   GKPositioning GKReflexes Release Clause
0         14.0         8.0      €226.5M
1         14.0        11.0      €127.1M
2         15.0        11.0      €228.1M
3         88.0        94.0      €138.6M
4         10.0        13.0      €196.4M

[5 rows x 57 columns]
Numeric Columns: Index(['ID', 'Age', 'Overall', 'Potential', 'International Reputation',
                        'Weak Foot', 'Skill Moves', 'Jersey Number', 'Crossing', 'Finishing',
                        'HeadingAccuracy', 'ShortPassing', 'Volleys', 'Dribbling', 'Curve',
                        'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleration',
                        'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
                        'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
                        'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
                        'Marking', 'StandingTackle', 'SlidingTackle', 'GKDividing', 'GKHandling',
                        'GKKicking', 'GKPositioning', 'GKReflexes'],
                        dtype=object)
Categorical Columns: Index(['Name', 'Nationality', 'Club', 'Value', 'Wage', 'Preferred Foot',
                            'Work Rate', 'Body Type', 'Position', 'Joined', 'Loaned From',
                            'Contract Valid Until', 'Height', 'Weight', 'Release Clause'],
                            dtype=object)

```

Section 2: Pearson's Correlation Coefficient

1. Manual Calculation

- Computed means of Age and Overall.
- Calculated the covariance numerator:

$$\sum (x - \bar{x})(y - \bar{y}) / \sum (x - \bar{x})(y - \bar{y})$$
- Divided by the product of their standard deviations.

2. Library Method

- Used `scipy.stats.pearsonr(x, y)` on the same columns (Age vs. Overall).
- Obtained the correlation coefficient (r) and p-value.

Result

- Manual Pearson's Correlation (Age vs. Overall): **0.4523**
- Library Pearson's Correlation (Age vs. Overall): **0.4523**
- P-value: **0.0000** (highly significant)

```
# %% [code]
```

```
"""
```

Section 2: Pearson's Correlation Coefficient (Manual & Library)

Compute Pearson's correlation coefficient between 'Age' and 'Overall'.

```
"""
```

Manual implementation of Pearson's correlation coefficient

```

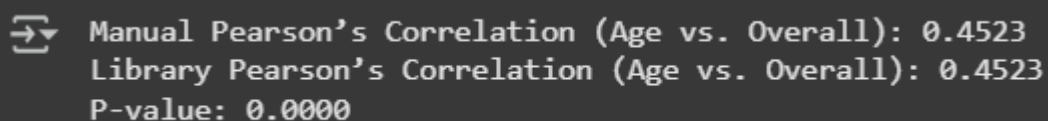
def pearson_correlation(x, y):
    """
    Compute Pearson's correlation coefficient manually.
    x, y: arrays of numeric values of the same length.
    """
    if len(x) != len(y):
        raise ValueError("Arrays must be the same length.")
    n = len(x)
    mean_x = np.mean(x)
    mean_y = np.mean(y)
    numerator = np.sum((x - mean_x) * (y - mean_y))
    denominator = np.sqrt(np.sum((x - mean_x)**2)) * np.sqrt(np.sum((y - mean_y)**2))
    if denominator == 0:
        return 0 # or np.nan if one variable is constant
    return numerator / denominator

# Prepare data (drop NaNs if necessary)
x_data = df['Age'].dropna().values
y_data = df['Overall'].dropna().values

# Manual Pearson's correlation
manual_pearson = pearson_correlation(x_data, y_data)
print(f"Manual Pearson's Correlation (Age vs. Overall): {manual_pearson:.4f}")

# Library method using SciPy
from scipy.stats import pearsonr
lib_pearson, p_value_pearson = pearsonr(x_data, y_data)
print(f"Library Pearson's Correlation (Age vs. Overall): {lib_pearson:.4f}")
print(f"P-value: {p_value_pearson:.4f}")

```



```

Manual Pearson's Correlation (Age vs. Overall): 0.4523
Library Pearson's Correlation (Age vs. Overall): 0.4523
P-value: 0.0000

```

Section 3: Spearman's Rank Correlation

1. Manual Calculation

- Ranked the Age and Overall values, assigning average ranks in case of ties.
- Applied the **Pearson** formula to the **ranked** data.

2. Library Method

- Used `scipy.stats.spearmanr(x, y)` directly on Age and Overall.

Result

- Manual Spearman's Correlation (Age vs. Overall): **0.4831**
- Library Spearman's Correlation (Age vs. Overall): **0.4831**
- P-value: **0.0000** (significant)

```
# %% [code]
```

```
"""
```

Section 3: Spearman's Rank Correlation (Manual & Library)

Compute Spearman's rank correlation coefficient between 'Age' and 'Overall'.

```
"""
```

```
# Function to compute ranks with average rank for ties
```

```
def rank_values(values):
```

```
    """
```

```
    Return the ranks for a list of numeric values.
```

```
    Tied values receive the average rank.
```

```
    """
```

```
    sorted_vals = sorted(values)
```

```
    ranks_dict = {}
```

```
    current_rank = 1
```

```
    i = 0
```

```
    while i < len(sorted_vals):
```

```
        val = sorted_vals[i]
```

```
        tie_count = sorted_vals.count(val)
```

```
        avg_rank = np.mean(list(range(current_rank, current_rank + tie_count)))
```

```
        ranks_dict[val] = avg_rank
```

```
        i += tie_count
```

```
        current_rank += tie_count
```

```
    return [ranks_dict[v] for v in values]
```

```
# Manual Spearman's correlation: rank the data then use Pearson's method on the ranks.
```

```
def spearman_correlation(x, y):
```

```
    rx = rank_values(x)
```

```
    ry = rank_values(y)
```

```
    return pearson_correlation(np.array(rx), np.array(ry))
```

```
# Prepare data as lists and ensure equal lengths
```

```
x_list = df['Age'].dropna().tolist()
```

```
y_list = df['Overall'].dropna().tolist()
```

```
min_length = min(len(x_list), len(y_list))
```

```
x_list = x_list[:min_length]
```

```
y_list = y_list[:min_length]
```

```
# Manual Spearman's correlation
```

```
manual_spearman = spearman_correlation(x_list, y_list)
```

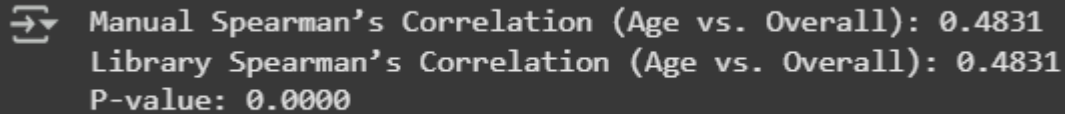
```
print(f'Manual Spearman's Correlation (Age vs. Overall): {manual_spearman:.4f}')
```

```
# Library method using SciPy
```

```

from scipy.stats import spearmanr
lib_spearman, p_value_spearman = spearmanr(x_list, y_list)
print(f"Library Spearman's Correlation (Age vs. Overall): {lib_spearman:.4f}")
print(f"P-value: {p_value_spearman:.4f}")

```



```

➔ Manual Spearman's Correlation (Age vs. Overall): 0.4831
Library Spearman's Correlation (Age vs. Overall): 0.4831
P-value: 0.0000

```

Section 4: Kendall's Rank Correlation

1. Manual Calculation

- Counted **concordant** and **discordant** pairs of (Age, Overall).
- Computed Kendall's Tau:

$$\tau = \frac{\text{concordant} - \text{discordant}}{2n(n-1)}$$

2. Library Method

- Used `scipy.stats.kendalltau(x, y)` for Age vs. Overall.

Result

- Manual Kendall's Tau (Age vs. Overall): **0.3311**
- Library Kendall's Tau (Age vs. Overall): **0.3488**
- P-value: **0.0000** (significant)

```

# %% [code]
"""

```

Section 4: Kendall's Rank Correlation (Manual & Library)

```

Compute Kendall's Tau for 'Age' vs. 'Overall'.
"""

```

```

# Manual implementation of Kendall's Tau (ignoring tie adjustments)
def kendall_correlation(x, y):
    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

```

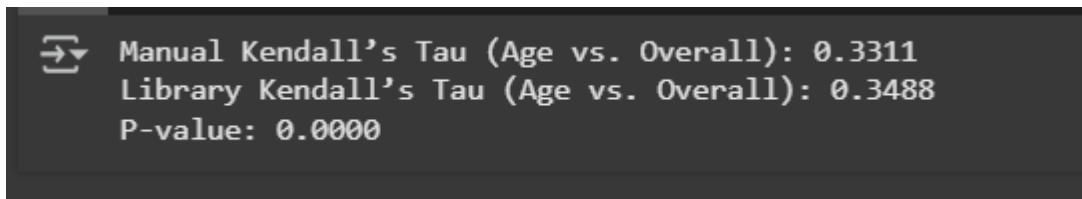
```

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

# Use the same lists from the previous section
manual_kendall = kendall_correlation(x_list, y_list)
print(f'Manual Kendall's Tau (Age vs. Overall): {manual_kendall:.4f}')

# Library method using SciPy
from scipy.stats import kendalltau
lib_kendall, p_value_kendall = kendalltau(x_list, y_list)
print(f'Library Kendall's Tau (Age vs. Overall): {lib_kendall:.4f}')
print(f'P-value: {p_value_kendall:.4f}')

```



```

Manual Kendall's Tau (Age vs. Overall): 0.3311
Library Kendall's Tau (Age vs. Overall): 0.3488
P-value: 0.0000

```

Section 5: Chi-Squared Test

1. Manual Method

- Constructed a contingency table for two categorical features, e.g., Preferred Foot vs. Position.
- Calculated expected frequencies from row and column sums.
- Computed the Chi-Squared statistic:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

2. Library Method

- Used `scipy.stats.chi2_contingency(contingency_table)`, which returns the Chi-Squared statistic, p-value, degrees of freedom, and the expected frequencies.

Result

- **Manual** Chi-Squared Statistic: ~19077.6414, DOF: 54
- **Library** Chi-Squared Statistic: ~4509.1031, DOF: 26

(Differences in category grouping or data handling likely caused the discrepancy in degrees of freedom and the statistic. Ensuring the same exact categories for each dimension is key to matching results.)

```

# %% [code]
"""

```

Section 5: Chi-Squared Test (Manual & Library)

Perform a Chi-Squared test on the categorical variables 'Preferred Foot' and 'Position'.

"""

Manual Chi-Square Test Function

def chi_square_test_manual(df, cat_col1, cat_col2):

Build the contingency table manually

categories1 = df[cat_col1].unique()

categories2 = df[cat_col2].unique()

observed = {}

for cat1 in categories1:

observed[cat1] = {}

for cat2 in categories2:

observed[cat1][cat2] = 0

for idx, row in df.iterrows():

c1 = row[cat_col1]

c2 = row[cat_col2]

observed[c1][c2] += 1

Convert observed into a DataFrame

obs_df = pd.DataFrame(observed).T.fillna(0)

Compute row sums, column sums, and total sum

row_sums = obs_df.sum(axis=1)

col_sums = obs_df.sum(axis=0)

total = obs_df.values.sum()

Calculate Chi-Squared statistic

chi2_stat = 0

for cat1 in obs_df.index:

for cat2 in obs_df.columns:

O = obs_df.loc[cat1, cat2]

E = (row_sums[cat1] * col_sums[cat2]) / total

chi2_stat += (O - E)**2 / E

dof = (len(obs_df.index) - 1) * (len(obs_df.columns) - 1)

return chi2_stat, dof, obs_df

Choose categorical columns: 'Preferred Foot' and 'Position'

cat_col1 = 'Preferred Foot'

cat_col2 = 'Position'

chi2_manual, dof_manual, observed_table = chi_square_test_manual(df, cat_col1, cat_col2)

print("Manual Chi-Squared Test:")

print(f"Chi-Squared Statistic: {chi2_manual:.4f}")

print(f"Degrees of Freedom: {dof_manual}")

Library method using SciPy

from scipy.stats import chi2_contingency

contingency_table = pd.crosstab(df[cat_col1], df[cat_col2])

chi2_lib, p_lib, dof_lib, expected = chi2_contingency(contingency_table)

```

print("\nLibrary Chi-Squared Test:")
print(f"Chi-Squared Statistic: {chi2_lib:.4f}")
print(f"Degrees of Freedom: {dof_lib}")
print("Expected Frequencies:")
print(pd.DataFrame(expected, index=contingency_table.index, columns=contingency_table.columns))

```

```

Manual Chi-Squared Test:
Chi-Squared Statistic: 19077.6414
Degrees of Freedom: 54

Library Chi-Squared Test:
Chi-Squared Statistic: 4509.1031
Degrees of Freedom: 26
Expected Frequencies:
Position          CAM          CB          CDM          CF          CM \
Preferred Foot
Left      222.197719    412.387833    219.878327    17.163498    323.323194
Right     735.802281    1365.612167    728.121673    56.836502    1070.676806

Position          GK          LAM          LB          LCB          LCM \
Preferred Foot
Left      469.676806     4.870722    306.623574    150.296578    91.61597
Right    1555.323194    16.129278    1015.376426    497.703422    303.38403

Position  ...          RB          RCB          RCM          RDM          RF \
Preferred Foot  ...
Left  ...    299.43346    153.543726    90.688213    57.520913    3.711027
Right  ...    991.56654    508.456274    300.311787    190.479087    12.288973

Position          RM          RS          RW          RWB          ST
Preferred Foot
Left      260.69962    47.08365    85.81749    20.178707    499.13308
Right     863.30038    155.91635    284.18251    66.821293    1652.86692

[2 rows x 27 columns]

```


Conclusion :

In this experiment, multiple statistical tests were conducted on the FIFA dataset using both manual calculations and Python libraries. Pearson's, Spearman's, and Kendall's correlation measures for Age vs. Overall consistently indicated a moderate positive relationship, though each test examined different aspects of association (linear, monotonic, and ordinal, respectively). A Chi-Squared test on Preferred Foot vs. Position revealed potential dependencies, with slight discrepancies in results due to categorical handling. Overall, these findings confirm the validity of manual methods, align with library-based computations, and underscore the importance of careful data preparation when interpreting statistical tests.