**Ex.no: 1 Hands-on exercise using Excel**

**AIM:**

To calculate basic statistical measure including count, average, median, mode, standard deviation, quartile, correlation, variance, maximum and minimum of a dataset using excel.

**ALGORITHM:**

Step 1: Open excel.

Step 2: Enter your dataset in a column or row.

Step 3: Write formulas to calculate:

Count: =COUNT(data\_range)

Average: =AVERAGE(data\_range)

Median: =MEDIAN(data\_range)

Mode: =MODE.SNGL(data\_range)

Standard deviation: =STDEV.P(data\_range)

Quartiles: =QUARTILE.INC(data\_range, quartile\_number) for each quartile

Correlation: =CORREL(data\_range1, data\_range2)

Variance: =VAR.P(data\_range)

Maximum: =MAX(data\_range)

Minimum: =MIN(data\_range)

Step 4: For frequency distribution, create bins and use the FREQUENCY function.

Step 5: Create a histogram using a bar chart and the frequency distribution data.

**PROBLEM STATEMENT:**

You are provided with dataset containing sports goods sales. The dataset includes goods, quantity, cost, discount, revenue, defective. Perform functions such as count, average, median, mode, standard deviation, maximum, minimum, variance, correlation.

**PROGRAM:**

Count: =COUNT (D2:D14)

Average: =AVERAGE (B2:B14)

Median: =MEDIAN (B2:B14)

Mode: =MODE.SNGL(C2:C14)

Standard deviation: =STDEV.P(B2:B14)

Quartiles: =QUARTILE.INC (B2:B14,1)

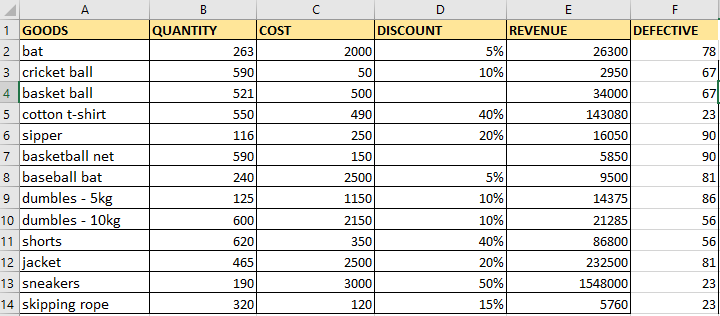
Correlation: =CORREL (B2:B14, C2:C14)

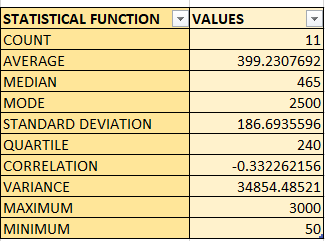
Variance: =VAR.P(B2:B14)

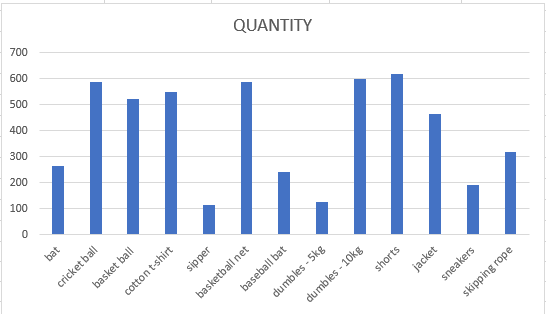
Maximum: =MAX (C2:C14)

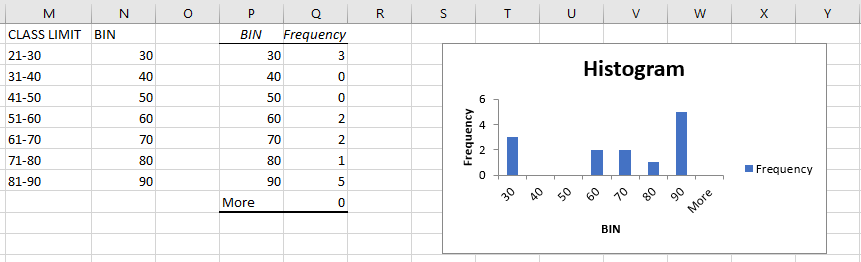
Minimum: =MIN (C2:C14)

**OUTPUT:**

****

****





**RESULT:**

Thus, the various statistical function in excel have been performed successfully.

**Ex.no: 2 Random Sampling**

**AIM:**

To write a Python Program to extract sample for the dataset using various random sampling methods.

**ALGORITHM:**

Step 1: Import necessary libraries: - import random

Step 2. Define the population data: - population data = [data1, data2, ..., data]

Step 3. Simple Random Sampling (SRS):

- Specify the sample size (sample size)

- srs\_sample = random.sample(population data, sample\_size)

Step 4. Systematic Sampling:

- Specify the sample size (sample\_size)

- Calculate the sampling interval (k) = population\_size / sample\_size

- Start at a random index within the interval (0 to k-1)

- systematic\_sample = [population\_data[i] for i in range(start\_index, len(population\_data), k)]

Step 5. Stratified Sampling:

- Divide the population into homogeneous strata

- Specify the sample size for each stratum (stratum\_sample\_sizes)

- For each stratum:

- Randomly sample from the stratum (using random.sample or np.random.choice) based on the specified sample size

- Combine the samples from all strata

Step 6. Return the samples obtained for analysis.

**PROGRAM:**

import random

from collections import defaultdict

import csv

# Load the dataset from the file

with open('employee.csv', mode='r') as file:

reader = csv.DictReader(file)

employees = [row for row in reader]

# Simple Random Sampling (SRS) with replacement

simple\_with\_replacement = random.choices(employees, k=5)

print("Simple Random Sample with Replacement:")

for emp in simple\_with\_replacement:

print(emp)

# Simple Random Sampling (SRS) without replacement

simple\_without\_replacement = random.sample(employees, k=3)

print("\nSimple Random Sample without Replacement:")

for emp in simple\_without\_replacement:

print(emp)

# Stratified Sampling with replacement

stratified\_with\_replacement = []

for gender in set(emp["Gender"] for emp in employees):

employees\_with\_gender = [emp for emp in employees if emp["Gender"] == gender]

selected = random.choices(employees\_with\_gender, k=2)

stratified\_with\_replacement.extend(selected)

print("\nStratified Sample with Replacement:")

for emp in stratified\_with\_replacement:

print(emp)

# Systematic Sampling with replacement

systematic\_with\_replacement = [employees[i] for i in range(0, len(employees), 4)]

print("\nSystematic Sample with Replacement:")

for emp in systematic\_with\_replacement:

print(emp)

# Systematic Sampling without replacement

systematic\_without\_replacement = [employees[i] for i in range(0, len(employees), len(employees)//3)]

print("\nSystematic Sample without Replacement:")

for emp in systematic\_without\_replacement:

print(emp)

# Cluster Sampling without replacement

unique\_genders = set(emp["Gender"] for emp in employees)

selected\_genders = random.sample(unique\_genders, k=2)

cluster\_without\_replacement = [emp for emp in employees if emp["Gender"] in selected\_genders]

print("\nCluster Sample without Replacement:")

for emp in cluster\_without\_replacement:

print(emp)

# Cluster Sampling with replacement

cluster\_with\_replacement = [random.choice(cluster\_without\_replacement) for \_ in range(5)]

print("\nCluster Sample with Replacement:")

for emp in cluster\_with\_replacement:

print(emp)

**OUTPUT:**

Simple Random Sample with Replacement:

{'ID': 6, 'Name': 'Frank', 'Age': 32, 'Gender': 'Male'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

{'ID': 10, 'Name': 'Jack', 'Age': 42, 'Gender': 'Male'}

{'ID': 2, 'Name': 'Bob', 'Age': 30, 'Gender': 'Male'}

Simple Random Sample without Replacement:

{'ID': 3, 'Name': 'Charlie', 'Age': 35, 'Gender': 'Male'}

{'ID': 8, 'Name': 'Henry', 'Age': 45, 'Gender': 'Male'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

Stratified Sample with Replacement:

{'ID': 4, 'Name': 'Diana', 'Age': 28, 'Gender': 'Female'}

{'ID': 5, 'Name': 'Eva', 'Age': 40, 'Gender': 'Female'}

{'ID': 3, 'Name': 'Charlie', 'Age': 35, 'Gender': 'Male'}

{'ID': 2, 'Name': 'Bob', 'Age': 30, 'Gender': 'Male'}

Systematic Sample with Replacement:

{'ID': 1, 'Name': 'Alice', 'Age': 25, 'Gender': 'Female'}

{'ID': 5, 'Name': 'Eva', 'Age': 40, 'Gender': 'Female'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

Systematic Sample without Replacement:

{'ID': 1, 'Name': 'Alice', 'Age': 25, 'Gender': 'Female'}

{'ID': 4, 'Name': 'Diana', 'Age': 28, 'Gender': 'Female'}

{'ID': 7, 'Name': 'Grace', 'Age': 27, 'Gender': 'Female'}

{'ID': 10, 'Name': 'Jack', 'Age': 42, 'Gender': 'Male'}

Cluster Sample without Replacement:

{'ID': 1, 'Name': 'Alice', 'Age': 25, 'Gender': 'Female'}

{'ID': 2, 'Name': 'Bob', 'Age': 30, 'Gender': 'Male'}

{'ID': 3, 'Name': 'Charlie', 'Age': 35, 'Gender': 'Male'}

{'ID': 4, 'Name': 'Diana', 'Age': 28, 'Gender': 'Female'}

{'ID': 5, 'Name': 'Eva', 'Age': 40, 'Gender': 'Female'}

{'ID': 6, 'Name': 'Frank', 'Age': 32, 'Gender': 'Male'}

{'ID': 7, 'Name': 'Grace', 'Age': 27, 'Gender': 'Female'}

{'ID': 8, 'Name': 'Henry', 'Age': 45, 'Gender': 'Male'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

{'ID': 10, 'Name': 'Jack', 'Age': 42, 'Gender': 'Male'}

Cluster Sample with Replacement:

{'ID': 10, 'Name': 'Jack', 'Age': 42, 'Gender': 'Male'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

{'ID': 5, 'Name': 'Eva', 'Age': 40, 'Gender': 'Female'}

{'ID': 9, 'Name': 'Ivy', 'Age': 38, 'Gender': 'Female'}

{'ID': 5, 'Name': 'Eva', 'Age': 40, 'Gender': 'Female'}

**RESULT:**

Thus, the program for random sampling has been executed successfully.

**Ex.no: 3 Z-Test Case Study**

**AIM:**

To gain practical experience in generating, visualizing, and analysing data following a normal

distribution using Z-scores, and to understand the application of Z-scores in identifying outliers.

**ALGORITHM:**

Step 1: Define case study and a dataset.

Step 2: State the null and alternative hypothesis

Step 3: Calculate statistics and standard error of the mean.

Step 4: Calculate test Statistics and decide to accept or reject the null hypothesis

Step 5: Evaluate and return the result

**CASE STUDY:**

Let's consider a case study where we are analysing the scores of students in a mathematics exam. We want to determine if the average score of students in this exam is significantly different from the national average score for students of the same grade level.

scores = [75, 82, 68, 90, 78, 85, 72, 88, 80, 79, 83, 87, 76, 84, 73,

89, 81, 74, 86, 77, 91, 69, 92, 70, 82, 76, 85, 79, 87, 78,

83, 71, 88, 77, 84, 80, 86, 75, 81, 74]

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

from scipy import stats

def z\_test\_one\_tailed(data, population\_mean, alpha, alternative='two-sided'):

sample\_mean = np.mean(data)

sample\_std = np.std(data, ddof=1)

n = len(data)

# Standard error of the mean

sem = sample\_std / np.sqrt(n)

# Step 4: Calculate test Statistics and decide to accept or reject the null hypothesis

z\_score = (sample\_mean - population\_mean) / sem

# Calculate p-value

if alternative == 'greater':

p\_value = 1 - stats.norm.cdf(z\_score)

elif alternative == 'less':

p\_value = stats.norm.cdf(z\_score)

else: # two-sided

p\_value = 2 \* (1 - stats.norm.cdf(np.abs(z\_score))) # two-tailed test

# Step 5: Evaluate and return the result

if p\_value < alpha:

if alternative == 'greater':

return f"Reject null hypothesis. (p-value: {p\_value})."

elif alternative == 'less':

return f"Reject null hypothesis. (p-value: {p\_value})."

else:

return f"Reject null hypothesis. (p-value: {p\_value})."

else:

if alternative == 'greater':

return f"Fail to reject null hypothesis. (p-value: {p\_value})."

elif alternative == 'less':

return f"Fail to reject null hypothesis. (p-value: {p\_value})."

else:

return f"Fail to reject null hypothesis. (p-value: {p\_value})."

def plot\_normal\_curve(data, population\_mean, alpha):

mu = np.mean(data)

sigma = np.std(data)

xmin = min(data)

xmax = max(data)

x = np.linspace(xmin, xmax, 100)

plt.plot(x, stats.norm.pdf(x, mu, sigma))

plt.title('Normal Distribution of Scores')

plt.xlabel('Score')

plt.ylabel('Probability Density')

# Shade the region beyond critical values for one-tailed tests

if alpha != 0.5:

if alpha == 0.05:

if mu < population\_mean: # For one-tailed less test

critical\_value = stats.norm.ppf(alpha)

plt.fill\_between(x[x <= mu + critical\_value], 0, stats.norm.pdf(x[x <= mu + critical\_value], mu, sigma), color='red', alpha=0.3)

else: # For one-tailed greater test

critical\_value = stats.norm.ppf(1-alpha)

plt.fill\_between(x[x >= mu + critical\_value], 0, stats.norm.pdf(x[x >= mu + critical\_value], mu, sigma), color='red', alpha=0.3)

plt.axvline(mu, color='k', linestyle='dashed', linewidth=1) # Plot a vertical line at the mean

plt.show()

# Constants

population\_mean = 80 # National average score for comparison

alpha = 0.05 # Significance level

# Scores of 40 students

scores = [75, 82, 68, 90, 78, 85, 72, 88, 80, 79, 83, 87, 76, 84, 73,

89, 81, 74, 86, 77, 91, 69, 92, 70, 82, 76, 85, 79, 87, 78,

83, 71, 88, 77, 84, 80, 86, 75, 81, 74]

# Perform one-tailed Z-test (greater)

result\_greater = z\_test\_one\_tailed(scores, population\_mean, alpha, alternative='greater')

print("One-tailed Z-test (greater):")

print(result\_greater)

print()

# Perform one-tailed Z-test (less)

result\_less = z\_test\_one\_tailed(scores, population\_mean, alpha, alternative='less')

print("One-tailed Z-test (less):")

print(result\_less)

print()

# Perform two-tailed Z-test

result\_two\_tailed = z\_test\_one\_tailed(scores, population\_mean, alpha)

print("Two-tailed Z-test:")

print(result\_two\_tailed)

print()

# Plot Normal Curve

plot\_normal\_curve(scores, population\_mean, alpha)

**OUTPUT:**

One-tailed Z-test (greater):

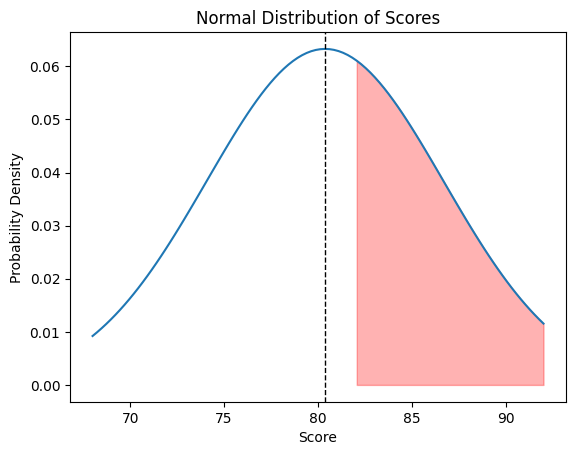
Fail to reject null hypothesis. (p-value: 0.355125710854745).

One-tailed Z-test (less):

Fail to reject null hypothesis. (p-value: 0.644874289145255).

Two-tailed Z-test:

Fail to reject null hypothesis. (p-value: 0.71025142170949).



**RESULT:**

Thus, the python program to calculate z-score and plotting of normal curve has been executed successfully.

**Ex.no: 4 T-Test Case Study**

**AIM:**

To determine if the mean height of students in a class is significantly different from the national average height of students using T-test.

**ALGORITHM:**

Step 1: Null Hypothesis (H0): The mean height of students in the class is equal to the national average height.

Step 2: Alternative Hypothesis (H1): The mean height of students in the class is different from the national average height.

Step 3: We conduct a one-sample t-test using the class's height data and the national average height.

Step 4: If the resulting p-value is below the chosen significance level (usually 0.05), we reject the null hypothesis, suggesting that there is a significant difference in the mean height between the class and the national average.

**CASE STUDY:**

In this case study, we aim to investigate whether the mean height of students in a particular class differs significantly from the national average height of students. This analysis is essential for understanding if there are any notable deviations in the height distribution within the class compared to the general population.

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

def one\_sample\_t\_test(sample\_data, population\_mean):

sample\_mean = np.mean(sample\_data)

sample\_std = np.std(sample\_data, ddof=1) # using Bessel's correction for sample standard deviation

n = len(sample\_data)

t\_statistic = (sample\_mean - population\_mean) / (sample\_std / np.sqrt(n))

degrees\_of\_freedom = n - 1

p\_value = 2 \* (1 - stats.t.cdf(abs(t\_statistic), df=degrees\_of\_freedom))

std\_error = sample\_std / np.sqrt(n)

return t\_statistic, p\_value, std\_error

def plot\_histogram(sample\_data, title):

plt.hist(sample\_data, bins=20, color='skyblue', edgecolor='black', alpha=0.7)

plt.title(title)

plt.xlabel('Height (cm)')

plt.ylabel('Frequency')

plt.show()

def case\_study\_one\_sample\_test():

print("Case Study 1: One-sample Test")

# Given data

sample\_data = np.random.normal(loc=170, scale=10, size=50) # Sample height data (mean=170cm, std=10cm)

population\_mean = 165 # National average height in cm

# Perform one-sample t-test

t\_statistic, p\_value, std\_error = one\_sample\_t\_test(sample\_data, population\_mean)

print("Results:")

print(f"T-statistic: {t\_statistic}")

print(f"P-value: {p\_value}")

print(f"Standard Error: {std\_error}")

# Plot histogram of sample data

plot\_histogram(sample\_data, "Height Distribution of Students in the Class")

if \_\_name\_\_ == "\_\_main\_\_":

case\_study\_one\_sample\_test()

**CASE STUDY:**

This case study aims to compare the effectiveness of two different teaching methods employed in a school by analysing their impact on students' exam scores. Understanding whether one method outperforms the other can inform educational strategies and curriculum development.

**PROGRAM:**

import numpy as np

from scipy import stats

import matplotlib.pyplot as plt

def two\_sample\_t\_test(sample1\_data, sample2\_data):

t\_statistic, p\_value = stats.ttest\_ind(sample1\_data, sample2\_data)

std\_error1 = np.std(sample1\_data) / np.sqrt(len(sample1\_data))

std\_error2 = np.std(sample2\_data) / np.sqrt(len(sample2\_data))

return t\_statistic, p\_value, std\_error1, std\_error2

def plot\_histogram(sample1\_data, sample2\_data, title1, title2):

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.hist(sample1\_data, bins=20, color='skyblue', edgecolor='black', alpha=0.7)

plt.title(title1)

plt.xlabel('Exam Scores')

plt.ylabel('Frequency')

plt.subplot(1, 2, 2)

plt.hist(sample2\_data, bins=20, color='lightgreen', edgecolor='black', alpha=0.7)

plt.title(title2)

plt.xlabel('Exam Scores')

plt.ylabel('Frequency')

plt.tight\_layout()

plt.show()

def case\_study\_two\_sample\_test():

print("Case Study 2: Two-sample Test")

# Given data

sample1\_data = np.random.normal(loc=75, scale=10, size=60) # Exam scores for Method A

sample2\_data = np.random.normal(loc=80, scale=10, size=60) # Exam scores for Method B

# Perform two-sample t-test

t\_statistic, p\_value, std\_error1, std\_error2 = two\_sample\_t\_test(sample1\_data, sample2\_data)

print("Results:")

print(f"T-statistic: {t\_statistic}")

print(f"P-value: {p\_value}")

print(f"Standard Error Sample 1 (Method A): {std\_error1}")

print(f"Standard Error Sample 2 (Method B): {std\_error2}")

# Plot histogram of sample data

plot\_histogram(sample1\_data, sample2\_data, "Method A", "Method B")

if \_\_name\_\_ == "\_\_main\_\_":

case\_study\_two\_sample\_test()

**OUTPUT:**

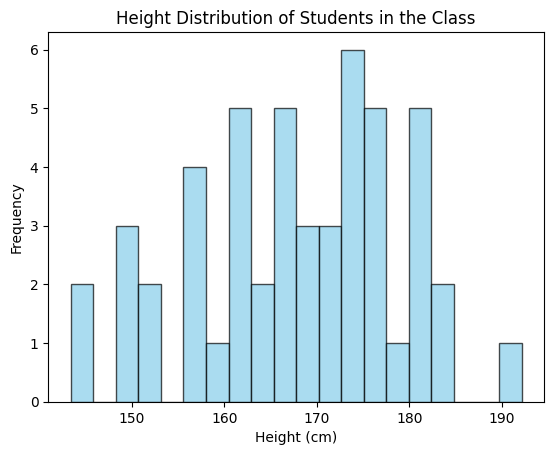
Case Study 1: One-sample Test

Results:

T-statistic: 1.6524945745853754

P-value: 0.10482820968311324

Standard Error: 1.5763244373266339



Case Study 2: Two-sample Test

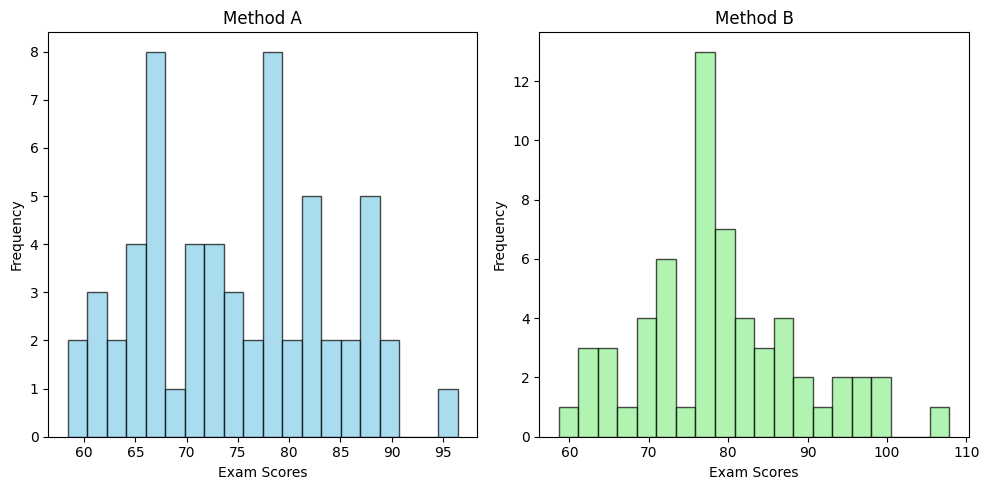
Results:

T-statistic: -2.3517709551044748

P-value: 0.020341414510860176

Standard Error Sample 1 (Method A): 1.1634300466531515

Standard Error Sample 2 (Method B): 1.311185859773644



**RESULT:**

Thus, the python program to perform one sample and two sample test using T-test was executed successfully.

**Ex.no: 5 ANOVA Case study**

**AIM:**

The aim of the program is to conduct a one-way ANOVA (Analysis of Variance) using Python. ANOVA is a statistical method used to analyse the difference between the means of three or more groups. In this specific case, the program aims to determine whether there is a statistically significant difference in response variable across different levels of a single factor.

**ALGORITHM:**

Step 1: Import necessary libraries: pandas, numpy, and f from scipy.stats.

Step 2: Define a function named one\_way\_anova that takes three arguments: data, factor\_col, and response\_col.

Step 3: Group the data by the factor\_col and collect response\_col values into lists for each group.

Step 4: Calculate the number of groups (n\_groups), the total number of observations (n\_total), and the minimum number of observations within a group (n\_within).

Step 5: Compute the grand mean of the response variable.

Step 6: Calculate the sum of squares between groups (SSB) by summing the squared deviations of group means from the grand mean, weighted by the number of observations in each group.

Step 7: Calculate the mean squares between groups (MSB) by dividing SSB by the degrees of freedom (n\_groups - 1).

Step 8: Calculate the sum of squares within groups (SSW) by summing the squared deviations of individual observations from their group means.

Step 9: Calculate the mean squares within groups (MSW) by dividing SSW by the degrees of freedom (n\_total - n\_groups).

Step 10: Compute the F-statistic by dividing MSB by MSW.

Step 11: Calculate the p-value associated with the F-statistic using the cumulative distribution function (CDF) of the F-distribution.

Step 12: Create an ANOVA table containing the sources of variation (Between, Within, Total), sum of squares (SS), degrees of freedom (df), mean squares (MS), F-statistic, and p-value.

Step 13: Return the ANOVA table.

Step 14: Provide an example case study for one-way ANOVA using the function and print the resulting ANOVA table.

**CASE STUDY:**

Scientists conducted an experiment to study the effect of different treatments on plant growth. They divided the plants into three treatment groups labelled as A, B, and C. Each treatment group received a different type of fertilizer. After a specified period, the scientists measured the height of each plant to assess its growth.

The recorded data is as follows:

Treatment Response (Plant Height)

A 10, 12, 15

B 8, 9, 11

C 14, 15, 16

Using the provided Python program, the scientists conducted a one-way ANOVA analysis to determine if there is a statistically significant difference in plant height among the treatment groups.

The ANOVA results revealed a significant difference in plant height among the treatment groups (p < 0.05). Post-hoc tests or further analysis could be conducted to identify specific differences between treatment groups.

**PROGRAM:**

import pandas as pd

import numpy as np

from scipy.stats import f

def one\_way\_anova(data, factor\_col, response\_col):

groups = data.groupby(factor\_col)[response\_col].apply(list)

n\_groups = len(groups)

n\_total = data.shape[0]

n\_within = data.groupby(factor\_col).size().min()

grand\_mean = data[response\_col].mean()

ssb = sum([len(group)\*(np.mean(group) - grand\_mean)\*\*2 for group in groups])

msb = ssb / (n\_groups - 1)

ssw = sum([sum((np.array(group) - np.mean(group))\*\*2) for group in groups])

msw = ssw / (n\_total - n\_groups)

f\_statistic = msb / msw

p\_value = 1 - f.cdf(f\_statistic, n\_groups - 1, n\_total - n\_groups)

anova\_table = pd.DataFrame({

'Source': ['Between', 'Within', 'Total'],

'SS': [ssb, ssw, ssb + ssw],

'df': [n\_groups - 1, n\_total - n\_groups, n\_total - 1],

'MS': [msb, msw, ''],

'F': [f\_statistic, '', ''],

'P-value': [p\_value, '', '']

})

return anova\_table

# Example case study for one-way ANOVA

one\_way\_data = pd.DataFrame({

'Treatment': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'],

'Response': [10, 12, 15, 8, 9, 11, 14, 15, 16]

})

print("One-Way ANOVA:")

print(one\_way\_anova(one\_way\_data, 'Treatment', 'Response'))

**CASE STUDY:**

Scientists conducted an experiment to investigate the combined effect of fertilizer type and watering frequency on plant growth. They divided plants into two groups based on the frequency of fertilizer application (daily or weekly) and watering (daily or weekly). After a specified period, they measured the height of each plant.

Using the provided Python program, the scientists performed a two-way ANOVA analysis to examine how fertilizer type and watering frequency interact to affect plant height. The ANOVA results revealed significant main effects of both fertilizer type (F(1, 24) = 7.292, p = 0.012) and watering frequency (F(1, 24) = 5.931, p = 0.023), as well as a significant interaction effect between fertilizer type and watering frequency (F(1, 24) = 6.107, p = 0.020).

Further post-hoc tests or subgroup analyses may be conducted to explore specific differences between treatment groups and understand the nuanced relationship between fertilizer type, watering frequency, and plant height.

This study underscores the importance of considering multiple factors simultaneously in agricultural research and highlights the utility of statistical methods, such as two-way ANOVA, in elucidating complex interactions between variables affecting plant growth.

**PROGRAM:**

import numpy as np

import pandas as pd

import statsmodels.api as sm

from statsmodels.formula.api import ols

# Create a dataframe

dataframe = pd.DataFrame({'Fertilizer': np.repeat(['daily', 'weekly'], 15),

'Watering': np.repeat(['daily', 'weekly'], 15),

'height': [14, 16, 15, 15, 16, 13, 12, 11,

14, 15, 16, 16, 17, 18, 14, 13,

14, 14, 14, 15, 16, 16, 17, 18,

14, 13, 14, 14, 14, 15]})

# Performing two-way ANOVA

model = ols('height ~ C(Fertilizer) + C(Watering) + C(Fertilizer):C(Watering)', data=dataframe).fit()

result = sm.stats.anova\_lm(model, type=2)

# Print the result in table format

print("Two-Way ANOVA Result:")

print(result)

**OUTPUT:**

One-Way ANOVA:

Source SS df MS F P-value

0 Between 48.222222 2 24.111111 7.482759 0.023439

1 Within 19.333333 6 3.222222

2 Total 67.555556 8

Two-Way ANOVA Result:

df sum\_sq mean\_sq F PR(>F)

C(Fertilizer) 1.0 0.033333 0.033333 0.012069 0.913305

C(Watering) 1.0 0.000369 0.000369 0.000133 0.990865

C(Fertilizer):C(Watering) 1.0 0.040866 0.040866 0.014796 0.904053

Residual 28.0 77.333333 2.761905 NaN NaN

**RESULT:**

Thus, the python program to perform ANOVA One-way classification and Two-way classification was executed successfully.

**Ex.no: 6 Regression**

**AIM:**

To understand the application of regression analysis in predicting the selling price of cars based on their specifications by developing a regression model.

**ALGORITHM:**

Step 1: Import necessary libraries: pandas, NumPy, and matplotlib.pyplot.

Step 2: Read the dataset containing car information from a CSV file.

Step 3: Extract the 'Engine' and 'Power' columns as independent (x) and dependent (y) variables.

Step 4: Calculate the mean of the 'Engine' and 'Power' columns.

Step 5: Initialize variables for calculating the slope (m) and y-intercept (c).

Step 6: Iterate through each data point to calculate the numerator and denominator of the slope formula.

Step 7: Compute the slope (m) and y-intercept (c) using the formulas.

Step 8: Predict the values of 'Power' (y) using the regression equation.

Step 9: Compute the Mean Squared Error (MSE) to evaluate model performance.

Step 10: Generate a set of x values for the regression line.

Step 11: Compute the corresponding y values for the regression line using the regression equation.

Step 12: Plot the regression line and data points on a scatter plot.

Step 13: Display the plot.

**CASE STUDY:**

You are provided with a dataset containing information about various cars, including their specifications and selling prices. The task is to develop a linear regression to predict the selling price of a car based on its specifications. This predictive model will assist potential buyers and sellers in estimating the fair market value of a car.

**PROGRAM:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv("/content/cars.csv")

x\_values = data['Engine']

y\_values = data['Power'].values

mean\_x = np.mean(x\_values)

mean\_y = np.mean(y\_values)

numer = 0

denom = 0

m = len(x\_values)

for i in range(m):

numer += (x\_values[i] - mean\_x) \* (y\_values[i] - mean\_y)

denom += (x\_values[i] - mean\_x) \*\* 2

m = numer / denom

c = mean\_y - (m \* mean\_x)

print(f"Slope Value (m): {m}\nY-intercept Value (c): {c}")

y\_pred = c + m \* x\_values

mse = np.mean((y\_pred - y\_values) \*\* 2)

print(f"Mean Squared Error (MSE): {mse}\n")

max\_x = np.max(x\_values) + 10

min\_x = np.min(x\_values) - 10

x\_plot = np.linspace(min\_x, max\_x, 10)

y\_plot = c + m \* x\_plot

plt.plot(x\_plot, y\_plot, color='red', label='Regression Line')

plt.scatter(x\_values, y\_values, c='purple', label='Data Points')

plt.xlabel('Engine')

plt.ylabel('Power')

plt.legend()

plt.show()

**CASE STUDY:**

The Student Performance Dataset is a dataset designed to examine the factors influencing academic student performance. The dataset consists of 10,000 student records, with each record containing information about various predictors and a performance index.

**PROGRAM:**import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv('/content/marks.csv')

df = data.head(250)

X\_previous\_score = df[['previous score']]

X\_new\_score = df[['new score']]

Y = df['sample questions']

def linear\_regression(X, Y):

mean\_X = np.mean(X)

mean\_Y = np.mean(Y)

numer = 0

denom = 0

m = 0

c = 0

n = len(X)

for i in range(n):

numer += (X[i] - mean\_X) \* (Y[i] - mean\_Y)

denom += (X[i] - mean\_X) \*\* 2

if denom != 0:

m = numer / denom

c = mean\_Y - (m \* mean\_X)

Y\_pred = m \* X + c

mse = np.sum((Y - Y\_pred) \*\* 2) / n

return Y\_pred, mse, m, c

Y\_pred\_previous\_score, mse\_previous\_score, m\_previous\_score, c\_previous\_score = linear\_regression(X\_previous\_score.values.flatten(), Y.values)

Y\_pred\_new\_score, mse\_new\_score, m\_new\_score, c\_new\_score = linear\_regression(X\_new\_score.values.flatten(), Y.values)

print("Regression Results for Sample Questions vs. Previous Score:")

print(f"Slope (m) for Previous Score: {m\_previous\_score:.2f}")

print(f"Y-intercept (c) for Previous Score: {c\_previous\_score:.2f}")

print(f"Mean Squared Error (MSE) for Previous Score: {mse\_previous\_score:.2f}")

print()

print("Regression Results for Sample Questions vs. New Score:")

print(f"Slope (m) for New Score: {m\_new\_score:.2f}")

print(f"Y-intercept (c) for New Score: {c\_new\_score:.2f}")

print(f"Mean Squared Error (MSE) for New Score: {mse\_new\_score:.2f}")

print()

plt.figure(figsize=(10, 6))

# Plot for Sample Questions vs. Previous Score

plt.scatter(X\_previous\_score, Y, color='red', label='Observed data (Previous Score)')

plt.plot(X\_previous\_score, Y\_pred\_previous\_score, color='blue', linewidth=2, label='Regression line (Previous Score)')

# Plot for Sample Questions vs. New Score

plt.scatter(X\_new\_score, Y, color='green', label='Observed data (New Score)')

plt.plot(X\_new\_score, Y\_pred\_new\_score, color='orange', linewidth=2, label='Regression line (New Score)')

plt.xlabel('Previous Score / New Score')

plt.ylabel('Sample Questions')

plt.title('Linear Regression: Sample Questions vs. Previous Score/New Score')

plt.legend()

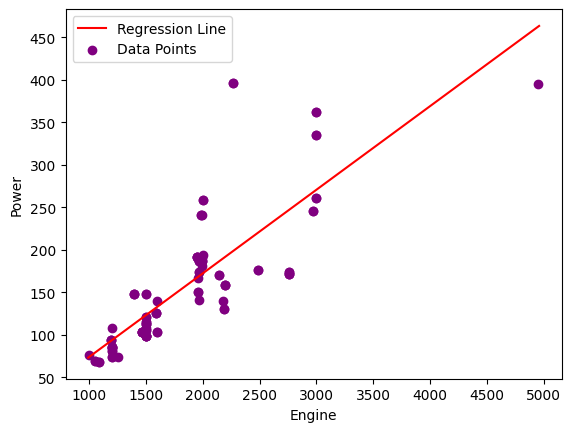
plt.grid(True)

plt.show()

**OUTPUT:**Slope Value (m): 0.09823028607866006

Y-intercept Value (c): -24.109773641712508

Mean Squared Error (MSE): 2057.5857064714783



Regression Results for Sample Questions vs. Previous Score:

Slope (m) for Previous Score: 0.29

Y-intercept (c) for Previous Score: 44.61

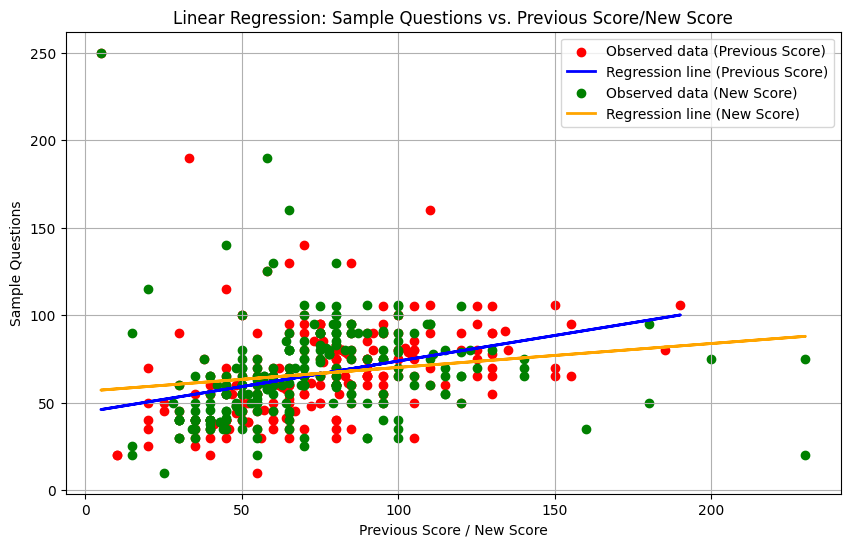
Mean Squared Error (MSE) for Previous Score: 638.79

Regression Results for Sample Questions vs. New Score:

Slope (m) for New Score: 0.14

Y-intercept (c) for New Score: 56.54

Mean Squared Error (MSE) for New Score: 700.31



**RESULT:**

Thus, the python program to understand the application of regression analysis was executed successfully.

**Ex.no:7 Logistic Regression**

**AIM:**

The aim of the program is to build a logistic regression model to predict diabetes outcome based on various health parameters and evaluates the model's performance by calculating accuracy, generating a confusion matrix, and plotting the Receiver Operating Characteristic (ROC) curve.

**ALGORITHM:**

Step 1: Import necessary libraries: pandas, numpy, seaborn, matplotlib.pyplot for data manipulation, visualization, and modeling.

Step 2: Load the dataset using pandas.

Step 3: Define features (X) and target variable (y) by separating the 'Outcome' column.

Step 4: Split the dataset into training and testing sets using train\_test\_split from sklearn.model\_selection.

Step 5: Initialize a logistic regression model with Logistic Regression from sklearn.linear\_model.

Step 6: Train the model using the training data.

Step 7: Predict the outcomes on the test set.

Step 8: Calculate accuracy using accuracy\_score from sklearn.metrics.

Step 9: Generate a confusion matrix using confusion\_matrix from sklearn.metrics.

Step 10: Plot the ROC curve using roc\_curve and auc functions from sklearn.metrics.

Step 11: Save the trained model using joblib.dump.

**CASE STUDY:**

The program utilizes the Pima Indians Diabetes Dataset, focusing on health data of adult females from the Pima Indian heritage to predict diabetes occurrence. It employs logistic regression to build a classification model using parameters like pregnancies, glucose level, BMI, etc. The model's accuracy is evaluated, and its performance is visualized via a ROC curve. The trained model is then saved for future use, aiding in early diabetes detection and personalized healthcare for at-risk individuals.

**PROGRAM:**

# Importing necessary libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_curve, auc, accuracy\_score, confusion\_matrix

import joblib

# Load the dataset

data = pd.read\_csv("/content/diabetes.csv")

# Display the first few rows of the dataset

print(data.head())

# Define features and target variable

X = data.drop('Outcome', axis=1)

y = data['Outcome']

# Splitting the dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train the logistic regression model

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Generate confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

# Plot ROC curve

y\_prob = model.predict\_proba(X\_test)[:,1]

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

# Save the model for future use

joblib.dump(model, 'diabetes\_logistic\_regression\_model.pkl')

**OUTPUT:**

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI

0 6 148 72 35 0 33.6

1 1 85 66 29 0 26.6

2 8 183 64 0 0 23.3

3 1 89 66 23 94 28.1

4 0 137 40 35 168 43.1

DiabetesPedigreeFunction Age Outcome

0 0.627 50 1

1 0.351 31 0

2 0.672 32 1

3 0.167 21 0

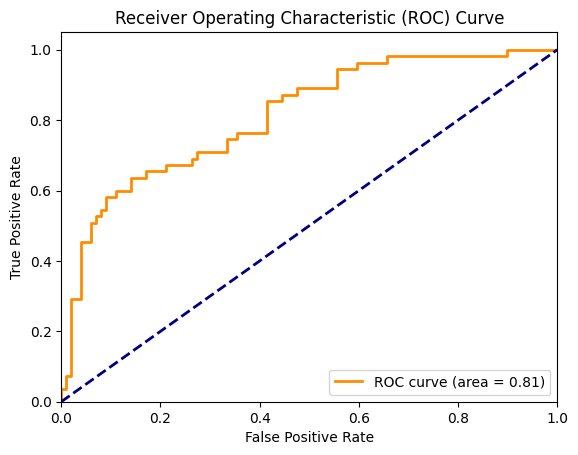
4 2.288 33 1

Accuracy: 0.7467532467532467

Confusion Matrix:

[[78 21]

[18 37]]



**RESULT:**

Thus, the python program to understand the application of logistic regression analysis was executed successfully.

**Ex.no:8 Time Series Analysis**

**AIM:**

The aim of the program is to conduct time series analysis on the mean temperature data for Delhi, India from 2013 to 2017. Additionally, it aims to perform the Augmented Dickey-Fuller (ADF) test to determine the stationarity of the mean temperature time series.

**ALGORITHM:**

Step 1: Import necessary libraries: pandas, numpy, matplotlib.pyplot for data manipulation, visualization, and time series analysis, and import adfuller from statsmodels.tsa.stattools for the ADF test.

Step 2: Load the dataset from the provided CSV file.

Step 3: Convert the 'date' column to datetime format using pd.to\_datetime() with the correct format specified.

Step 4: Set the 'date' column as the index of the DataFrame using set\_index().

Step 5: Plot the time series data for mean temperature using plt.plot().

Step 6: Title the plot and label the axes appropriately.

Step 7: Display the plot using plt.show().

Step 8: Perform the Augmented Dickey-Fuller test for stationarity on the mean temperature data using adfuller().

Step 9: Extract the ADF statistic, p-value, and critical values.

Step 10: Print out the ADF statistic, p-value, and critical values.

**CASE STUDY:**

The program analyzes mean temperature data for Delhi, India from 2013 to 2017.It plots the time series of mean temperature to visualize its trend and patterns over time.The Augmented Dickey-Fuller (ADF) test is conducted to assess the stationarity of the mean temperature time series.The ADF statistic, p-value, and critical values are extracted and displayed to determine the stationarity of the time series.

**PROGRAM:**

# Importing necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import adfuller

from statsmodels.graphics.tsaplots import plot\_acf

# Load the dataset

data = pd.read\_csv("/content/weather.csv")

# Convert 'date' column to datetime format

data['date'] = pd.to\_datetime(data['date'], format='%d-%m-%Y')

# Set 'date' column as the index

data.set\_index('date', inplace=True)

# Plot the time series data for 'meantemp'

plt.figure(figsize=(10, 6))

plt.plot(data['meantemp'], color='blue')

plt.title('Mean Temperature in Delhi (2013-2017)')

plt.xlabel('Date')

plt.ylabel('Mean Temperature (°C)')

plt.grid(True)

plt.show()

# Calculate moving average with a window of 30 days

moving\_avg = data['meantemp'].rolling(window=30).mean()

# Plot moving average

plt.figure(figsize=(10, 6))

plt.plot(data['meantemp'], color='blue', label='Original Data')

plt.plot(moving\_avg, color='red', label='Moving Average (30 days)')

plt.title('Mean Temperature in Delhi with Moving Average (2013-2017)')

plt.xlabel('Date')

plt.ylabel('Mean Temperature (°C)')

plt.legend()

plt.grid(True)

plt.show()

# Perform Augmented Dickey-Fuller test for stationarity on 'meantemp'

adf\_result = adfuller(data['meantemp'])

# Extract ADF statistics, p-value, and critical values

adf\_statistic = adf\_result[0]

p\_value = adf\_result[1]

critical\_values = adf\_result[4]

print("ADF Statistic:", adf\_statistic)

print("p-value:", p\_value)

print("Critical Values:")

for key, value in critical\_values.items():

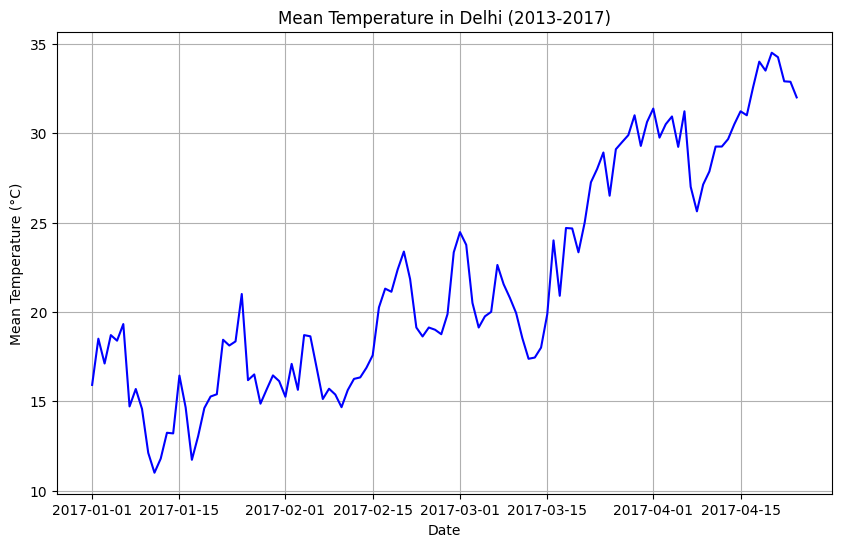
print(f"\t{key}: {value}")

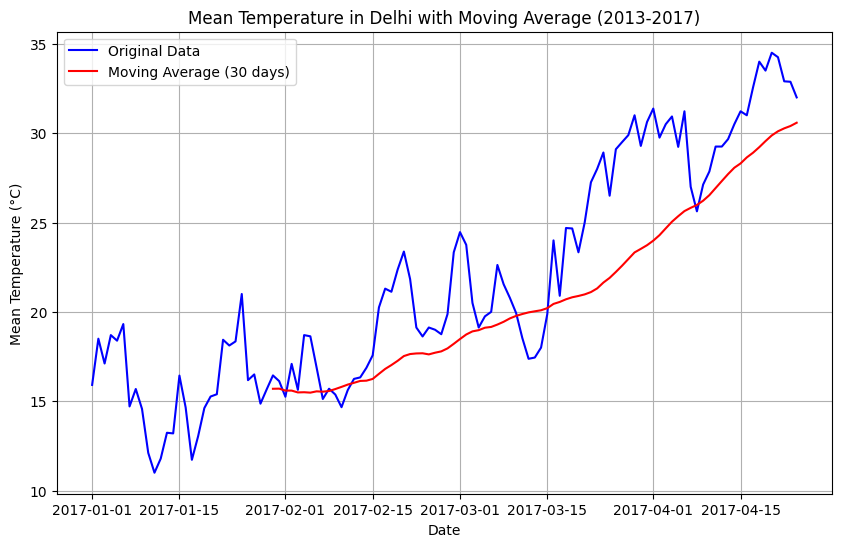
# Calculate autocorrelation

autocorrelation = data['meantemp'].autocorr()

print("Autocorrelation:", autocorrelation)

**OUTPUT:**

****



ADF Statistic: -1.0964742756900832

p-value: 0.7164758732513669

Critical Values:

1%: -3.489589552580676

5%: -2.887477210140433

10%: -2.580604145195395

Autocorrelation: 0.9644332900611061

**RESULT:**

Thus, the python program to understand the time series analysis was executed successfully.