INDEX

Practical 1:

Write a Python program that takes a student's exam score as input and prints the corresponding grade (A, B, C, D, or F) based on the following criteria:

A: 90-100

B: 80-89

C: 70-79

D: 60-69

F: Below 60

Code:

```
score = int(input("enter exam score"))
if (score <= 100 & score >=90):
    print("Grade A")
elif (score <= 89 & score >=80):
    print("Grade B")
elif (score <= 79 & score >=70):
    print("Grade C")
elif (score <= 69 & score >=60):
    print("Grade D")
else:
    print("Grade F")
```

Output:

```
enter exam score 80
Grade B
```

Practical 2:

Write a python program that allows the user to enter exactly 20 floating point values. The program then prints the sum, average (arithmetic mean), maximum and minimum of the values entered using loop.

Code:

```
numbers = []
for i in range(1,21):
    num = float(input("Enter 20 floating point values"))
    numbers.append(num)
print("Sum of 20 floating point values:", (sum(numbers)))
print("Average of numbers:", sum(numbers)/len(numbers))
print("Maximum of numbers:", max(numbers))
print("Minimum of numbers:", min(numbers))
```

Output:

```
Enter 20 floating point values5.2
Enter 20 floating point values8.9
Enter 20 floating point values12.3
Enter 20 floating point values6.5
Enter 20 floating point values1.2
Enter 20 floating point values3.21
Enter 20 floating point values6.99
Enter 20 floating point values12.45
Enter 20 floating point values36.98
Enter 20 floating point values25.25
Enter 20 floating point values0.2
Enter 20 floating point values0.02
Enter 20 floating point values65
Enter 20 floating point values2
Enter 20 floating point values3
Enter 20 floating point values8
Enter 20 floating point values98
Enter 20 floating point values63
Enter 20 floating point values44
Enter 20 floating point values54
Sum of 20 floating point values: 456.20000000000005
Average of numbers: 22.810000000000002
Maximum of numbers: 98.0
Minimum of numbers: 0.02
```

Practical 3:

Given an array representing the daily temperatures over a week, calculate the average temperature for the week.

Code:

```
# Given an array representing the daily temperatures over a week, calculate the average temperature for the week.
# Temperature (in Degree Celsius). Monday, 42. Tuesday, 40. Wednesday, 41. Thursday, 39. Friday, 35. Saturday, 38 . Sunday,37

temp_dic = { "Monday":42, "Tuesday":40, "Wednesday":41, "Thursday":39, "Friday":35, "Saturday":38, "Sunday":37 }
print(temp_dic)
temp = temp_dic.values()
temp
import statistics as stat
avg_temp = round(stat.mean(temp), 2)
print("Average of weekly temperture is {}".format(avg_temp))
```

Output:

```
{'Monday': 42, 'Tuesday': 40, 'Wednesday': 41, 'Thursday': 39, 'Friday': 35, 'Saturday': 38, 'Sunday': 37}
Average of weekly temperture is 38.86
```

Practical 4:

Write a guessing game program in which the computer chooses at random an integer in the range 1...100. The user's goal is to guess the number in the least number of tries. For each incorrect guess the user provides, the computer provides feedback whether the user's number is too high or too low.

Code:

```
import numpy as np

secret_no = np.random.randint(1,100)
count = 1
while True:
  if count > 5:
    print("Chal bey nikal")
    break
  guess = int(input())
  if guess == secret_no:
    print("correct guess"," guess 5count", count)
    break
  elif guess < secret_no:
    print("guess is too low. Try again!")
  else:
    print("guess is too high. Try again!")
  count += 1
```

Output:

```
85
guess is too high. Try again!
12
guess is too high. Try again!
7
guess is too high. Try again!
5
guess is too high. Try again!
1
guess is too low. Try again!
Chal bey nikal
```

Practical 5:

Create a Boolean array called above threshold that indicates True for days when the temperature was above the threshold (25C). Use the above threshold Boolean array to filter out the temperatures recorded on those days into a new array called high temperature. Print the high temperature array to display the temperatures recorded on days when the temperature was above the threshold.

Code:

```
temp = np.array([25, 32, 28,20,18,30])
mask = temp > 25
print(mask)
bool_arr = temp[mask]
print("temperature above thresold (25):",  bool_arr)
high_temp = bool_arr
high_temp
```

Output:

```
[False  True  True False False  True]
temperature above thresold (25): [32 28 30]
```

Practical 6:

Compute the seasonal indices from the following time-series data on production.

Code:

```
import numpy as np
data=np.array([[226.7,  194.7,  185.2,  221.1],
[208.1, 176.2,  175.1,  223.2],
[237.1, 201.7,  202.8,  267.7],
[243.3, 201.1,  203.3,  259],
[248.3, 197.4,  205.8,  261.5],
[228.4, 191.1, 190.5, 259.3],
[212.3, 174.9,  177.9,  243.1],
[217.1, 182.4,  202.9,  257.3],
[222.7, 189.6,  213.3,  265.6],
[235.5, 218.1,  236.9,  292.2],
[222.3, 211.6,  236.1,  291.5],
[212.4, 206.0,  225.4,  294.8]])
monthly_avg=np.mean(data,axis=0)
indices=data/monthly_avg
seasonal_indices=np.mean(indices,axis=1)
seasonal_indices/=np.mean(seasonal_indices)
month=["JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC"]
for month, index in zip(month,seasonal_indices):
  print(f"Seasonal Index for {month}: {index:2f}")
```

Output:

```
Seasonal Index for JAN: 0.937462
Seasonal Index for FEB: 0.882902
Seasonal Index for MAR: 1.023993
Seasonal Index for APR: 1.022367
Seasonal Index for MAY: 1.028606
Seasonal Index for JUN: 0.977751
Seasonal Index for JUL: 0.908337
Seasonal Index for AUG: 0.967368
Seasonal Index for SEP: 1.003417
Seasonal Index for OCT: 1.108309
Seasonal Index for NOV: 1.083756
Seasonal Index for DEC: 1.055731
```

Practical 7:

Create a structured array to represent customer orders with fields like 'order_id', 'customer_name', 'product_name', 'quantity', and 'total_price'. Compute metrics such as total revenue, average order value, or identify the most popular products.

Code:

```
import numpy as np
from numpy import record

# Define data types for each field
data_types = [('order_id', 'i4'), ('customer_name', 'U255'), ('product_name', 'U255'),
              ('quantity', 'i4'), ('total_price', 'f8')]

# Sample data (replace with your actual data)
orders_data = [
  (1001, 'Alice', 'Phone Case', 2, 19.99),
  (1002, 'Bob', 'Headphones', 1, 49.99),
  (1003, 'Alice', 'Screen Protector', 3, 9.99),
  (1004, 'Charlie', 'Phone Case', 1, 19.99),
]

# Create the structured array
customer_orders = np.array(orders_data, dtype=data_types)
customer_orders

# Total revenue
total_revenue = customer_orders['total_price'].sum()

# Average order value
average_order_value = total_revenue / len(customer_orders)

# Count product occurrences using np.unique and np.count_nonzero
unique_products, product_counts = np.unique(customer_orders['product_name'], return_counts=True)

# Most popular product (assuming 'quantity' indicates popularity)
most_popular_product_index = np.argmax(product_counts)
most_popular_product = unique_products[most_popular_product_index]

print(f"Total Revenue: ${total_revenue:.2f}")
print(f"Average Order Value: ${average_order_value:.2f}")
print(f"Most Popular Product: {most_popular_product}")
```

Output:

```
Total Revenue: $99.96
Average Order Value: $24.99
Most Popular Product: Phone Case
```

Practical 8:

Compute the probability of occurrence of 3 successes using loop where no. of trials is 10 and probability of success is 0.5.

Code:

```
from scipy.stats import binom
n=10
p = 0.5
x = 0
while (x<3):
  x += 1
  prob = binom.pmf(x,n,p)
print("probability of getting exactly {} success in {} trials : {}". format(x,n,prob)
# print(f"Probability of getting exactly {x} success in {n} trials: {prob:.4f}")
```

Output:

```
probability of getting exactly 3 success in 10 trials : 0.1171875
```

Practical 9:

Compute the probability of occurrence of event less than equal to 2 using loop where lambda = 5.

Code:

```
from scipy.stats import poisson
lambda_ = 5  #avg rate of events per interval
x = 0
while (x<=2):
  x += 1    #no. of events
  prob = poisson.pmf(x, lambda_)
print("probability of observing event less than equal to {} events: {:.4f}".format(x, prob))
```

Output:

```
probability of observing event less than equal to 3 events: 0.1404
```

Practical 10:

Consider the daily simple returns of the given data. The data are in the file d-mmm-0111.txt. Compute the sample mean, standard deviation, skewness, excess kurtosis, minimum, and maximum of each simple return series.

Code:

```
import pandas as pd
from scipy.stats import skew, kurtosis

data = pd.read_csv("/content/drive/MyDrive/Data Science/Practice Data/d-mmm-0111.txt", sep= "\s+", index_col= 0)
print(data.head())
print("*******")
print("Sample mean of Returns=", data.rtn.mean().round(5))
print("Standard Deviation of Returns=", data.rtn.std().round(5))
print("Skewness in Returns=", data.rtn.skew().round(5))
print("Kurtosis in Returns=", data.rtn.kurtosis().round(5))
print("Maximum Returns=", data.rtn.max().round(5))
print("Minimum Returns=", data.rtn.min().round(5))
```

Output:

```
               rtn
date
20010102 -0.010892
20010103 -0.011536
20010104  0.010080
20010105 -0.037290
20010108  0.006001
*******
Sample mean of Returns= 0.00028
Standard Deviation of Returns= 0.01549
Skewness in Returns= 0.02798
Kurtosis in Returns= 4.64738
Maximum Returns= 0.09878
Minimum Returns= -0.08957
```

Practical 11:

Suppose we have a bank counter where customers arrive to conduct transactions. Let's assume that the average rate of customer arrivals at the counter is 4 customers per hour (i.e., $\lambda=4$ customers per hour), following an exponential distribution. We can use the exponential distribution to answer questions such as:

1. What is the probability that the time between two consecutive customer arrivals is less than 15 minutes?
2. What is the expected time until the next customer arrives?
3. What is the probability that the time between two consecutive customer arrivals is greater than 30 minutes?

Code:

```
from scipy.stats import expon
lambdaRate = 4
mean_time_btw_arrival = 1/lambdaRate
prob_less_than_15_min = expon.cdf((15/60),scale = mean_time_btw_arrival) #scale = mean
print(f"1. P(time b/w two consecutive arrivals is less than 15 min.)= {prob_less_than_15_min:.4f}")

expected_time_until_next_arrival = 1/lambdaRate
print(f"2. Expected time until the next customer arrives) = {expected_time_until_next_arrival:.4f}")

prob_greater_than_30_min = expon.sf((30/60), scale = mean_time_btw_arrival)
print(f"3. P(time b/w two consecutive customer arrivals is greater than 30 min.)= {prob_greater_than_30_min:.4f}")
```

Output:

```
1. P(time b/w two consecutive arrivals is less than 15 min.)= 0.6321
2. Expected time until the next customer arrives) = 0.2500
3. P(time b/w two consecutive customer arrivals is greater than 30 min.)= 0.1353
```

Practical 12:

A manufacturing company produces light bulbs, and the lifetime of these bulbs follows a normal distribution with a mean of 1000 hours and a standard deviation of 50 hours. The company claims that at least 95% of its bulbs will last for more than 950 hours. Test this claim using a hypothesis test with a significance level of 0.05.

Code:

```
from scipy.stats import norm

# Define claim and parameters
mean_lifetime = 1000
std_dev = 50
min_acceptable_lifetime = 950
desired_proportion = 0.95
significance_level = 0.05

# Hypothesis definition
null_hypothesis = "lifetime of Bulb > 950 hours"
alternative_hypothesis =  "lifetime of Bulb !> 950 hours"
print("H0: ", null_hypothesis)
print("H1: ", alternative_hypothesis)

# Calculate critical value (one-tailed test)
critical_z = norm.ppf(1 - significance_level)
print("critical value at 5%: ", critical_z)

# Calculate z-score for the minimum acceptable lifetime
z_score = (min_acceptable_lifetime - mean_lifetime) / std_dev
print("calculated Z value:", z_score)

#p value
p_value = 1-norm.cdf(z_score)
print("p value: ", p_value)

# Perform the hypothesis test
if z_score < critical_z:
    print("Z cal < critical_z")
    print("Fail to reject null hypothesis.")
else:
    print("Z cal > critical_z")
    print("Reject null hypothesis.")
```

Output:

```
H0:  lifetime of Bulb > 950 hours
H1:  lifetime of Bulb !> 950 hours
critical value at 5%:  1.6448536269514722
calculated Z value: -1.0
p value:  0.8413447460685429
Z cal < critical_z
Fail to reject null hypothesis.
```

Practical 13:

The lifetimes of a certain type of electronic component follow a log-normal distribution with a mean lifetime of 500 hours and a standard deviation of 100 hours on the natural logarithmic scale. Calculate the following probabilities:

I. The probability that a randomly selected component lasts less than 400 hours.
II. The probability that a randomly selected component lasts between 450 and 550 hours.

Code:

```
from scipy.stats import lognorm

# Define parameters for the log-normal distribution
mean_lifetime_ln = 500
std_dev_ln = 100

# Probabilities to calculate
threshold_1 = 400
threshold_2 = 450
threshold_3 = 550

# Probability of lasting less than 400 hours
p_less_than_400 = lognorm.cdf(threshold_1,s =1, scale = std_dev_ln)

# Probability of lasting between 450 and 550 hours
# CDF (cumulative distribution function) considers probability up to the value
# calculate probability between two values, subtract the CDF at the lower threshold from the CDF at the upper threshold
p_between_450_550 = lognorm.cdf(threshold_3, s =1, scale = std_dev_ln) - lognorm.cdf(threshold_2,s = 1, scale = std_dev

# Print the probabilities
print(f"Probability of lasting less than 400 hours: {p_less_than_400:.4f}")
print(f"Probability of lasting between 450 and 550 hours: {p_between_450_550:.4f}")
```

Output:

```
Probability of lasting less than 400 hours: 0.9172
Probability of lasting between 450 and 550 hours: 0.0222
```

Practical 14:

A group of 5 patients treated with medicine. A is of weight 42,39,38,60 & 41 kgs. Second group of 7 patients from the same hospital treated with medicine B is of weight 38, 42, 56, 64, 68, 69, & 62 kgs. Find whether there is any difference between medicines? Solve using python.

Code:

```
from scipy import stats
import numpy as np

# Hypothesis definition
null_hypothesis = "The is no significant difference in Mean scores of medicine A and Medicine B for both group"
alternative_hypothesis =  "There is significant difference in mean score of medician A and medicine B for both the groups"
print("H0: ", null_hypothesis)
print("H1: ", alternative_hypothesis)

methodA = np.array([80,85,75,90,78])
methodB = np.array([82,88,72,92,80])
t_stat, p_value = stats.ttest_ind(methodA, methodB)
print("t_statistic:", t_stat)
print("p_value:", p_value)
if p_value < t_stat:
  print("Enough evidence to reject Null Hyothesis")
else:
  print("Not having enough evidence. Therefore, Fail to reject Null Hypothesis")
```

Output:

```
H0:  The is no significant difference in Mean scores of medicine A and Medicine B for both group
H1:  There is significant difference in mean score of medician A and medicine B for both the groups
t_statistic: -0.27602622373694236
p_value: 0.7895243197818331
Not having enough evidence. Therefore, Fail to reject Null Hypothesis
```

Practical 15:

A tutoring program is implemented for a group of students to improve their exam scores. The students' exam scores are recorded before and after participating in the tutoring program. Is there a significant improvement in exam scores after the tutoring program? Solve using python.

Code:

```
from scipy import stats
import numpy as np
null_hypothesis = "The is no significant difference in Mean scores of students before and after tutoring"
alternative_hypothesis =  "There is significant difference in mean score of students before and affer tutoring"
print("H0: ", null_hypothesis)
print("H1: ", alternative_hypothesis)

# paired t test
before = np.array([80,85,75,90,78])
after =np.array([82,88,72,92,80])
t_stat, p_value = stats.ttest_rel(before,after)
print("t_statistic:", t_stat)
print("p_value:", p_value)

if p_value < t_stat:
  print("Enough evidence to reject Null Hyothesis")
else:
  print("Not having enough evidence. Therefore, Fail to reject Null Hypothesis")
```

Output:

```
H0:  The is no significant difference in Mean scores of students before and after tutoring
H1:  There is significant difference in mean score of students before and affer tutoring
t_statistic: -1.1239029738980326
p_value: 0.323940830991843
Not having enough evidence. Therefore, Fail to reject Null Hypothesis
```

Practical 16:

A researcher wants to determine if there is a significant association between smoking habits and the incidence of lung cancer in a population. The researcher collects data from a sample of individuals, recording whether each individual is a smoker (S) or a non-smoker (NS), and whether each individual has been diagnosed with lung cancer (LC) or not (NLC).

Code:

```
null_hypothesis = "There is no significant association between smoking habits and the incidence of lung cancer"
alternative_hypothesis = "There is a significant association between smoking habits and the incidence of lung cancer."
print("H0: ", null_hypothesis)
print("H1: ", alternative_hypothesis)
from scipy.stats import chi2_contingency
info = [[135,93],[105,117]]
print("Smoking habits and incidence of lung cancer in a population:\n", info)
stat, P, dof, expected = chi2_contingency(info)
alpha = 0.05
print("Chi square test value:",dof)
print("alpha:", 0.05)
print("P value:", P)
if P <= alpha:
  print("Reject null hypothesis. There is statistically significant association between smoking habits and incidence of lung cancer ")
else:
  print("Accept null hypothesis.We couldn't determine the stronger evidence against null hypothesis to reject the significant assocition ")
```

Output:

```
H0:  There is no significant association between smoking habits and the incidence of lung cancer
H1:  There is a significant association between smoking habits and the incidence of lung cancer.
Smoking habits and incidence of lung cancer in a population:
 [[135, 93], [105, 117]]
Chi square test value: 1
alpha: 0.05
P value: 0.014765196285659136
Reject null hypothesis. There is statistically significant association between smoking habits and incidence of lung cancer
```

Practical 17:

A researcher wants to compare the performance of three different teaching methods (Method A, Method B, and Method C) in improving students' test scores. The researcher randomly assigns 30 students to one of the three teaching methods (10 students per method). After completing the teaching intervention, the students' test scores are recorded. The researcher wants to know if there is a significant difference in test scores among the three teaching methods.

Code:

```
# define hypothesis
print("H0: There is no significant difference among 3 Methods.")
print("H1: There is a significant difference among 3 Methods.")

from scipy import stats
method_A = [96,66,85.4,75,69.9,87,53,67,91,78]
method_B = [60.9,98,78.1,55,66.7,87.4,93.2,81,79,77]
method_C = [51,98,99,76,84,85,86,68,77,74.9]

# one way ANOVA or F test for checking if there is any signficant diffference betweeen two or more method
f_statistic, p_value = stats.f_oneway(method_A, method_B,method_C)
print("f_statistic", f_statistic)
print("p_value", p_value)
if p_value <= 0.05:
  print("Reject H0 ,there is significant difference among 3 methods ")
else:
  print("Fail to reject H0, there is no significant difference among 3 methods")
```

Output:

```
H0: There is no significant difference among 3 Methods.
H1: There is a significant difference among 3 Methods.
f_statistic 0.1345021501426802
p_value 0.8747330395199127
Fail to reject H0, there is no significant difference among 3 methods
```

Practical 18:

Read the excel file 'advertising.csv' in python and use different indexing and selection functions.

Dataset:

index	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
5	8.7	48.9	75.0	7.2
6	57.5	32.8	23.5	11.8
7	120.2	19.6	11.6	13.2
8	8.6	2.1	1.0	4.8
9	199.8	2.6	21.2	15.6

Code:

```
import pandas as pd

data = pd.read_csv("/content/drive/MyDrive/Data Science/Practice Data/advertising.csv"
print(data.head())

# Selecting using iloc (index-based selection)
print("\nSelecting using iloc:")
print(data.iloc[0:3, 1:3])

# Selecting using loc (label-based selection)
print("\nSelecting using loc:")
print(data.loc[0:2, ['TV', 'Radio']])

# Using at to access a single value
print("\nUsing at to access a single value:")
print(data.at[0, 'TV'])

# Using iat to access a single value
print("\nUsing iat to access a single value:")
print(data.iat[0, 0])
```

Output:

i. Using iloc() function

```
Selecting using iloc:
   Radio  Newspaper
0   37.8       69.2
1   39.3       45.1
2   45.9       69.3
```

ii. Using loc() function

```
Selecting using loc:
      TV  Radio
0  230.1   37.8
1   44.5   39.3
2   17.2   45.9
```

iii. Using at() function

```
Using at to access a single value:
230.1
```

iv. Using iat() function

```
Using iat to access a single value:
230.1
```

Practical 19:

Create a random data on housing, handle the missing values using different functions.

1. **Creating random Dataset :**

```
import pandas as pd
import numpy as np
# Creating random housing data
np.random.seed(0)
data = {
    'Price': np.random.randint(100000, 1000000, size=10).astype(float),
    'Bedrooms': np.random.randint(1, 6, size=10).astype(float),  # Define as float
    'Bathrooms': np.random.randint(1, 4, size=10),
    'Area (sq ft)': np.random.randint(1000, 3000, size=10).astype(float),
    'Year Built': np.random.randint(1970, 2020, size=10)}
# Introducing missing values
data['Price'][2] = np.nan
data['Bedrooms'][5] = np.nan
data['Area (sq ft)'][8] = np.nan

df = pd.DataFrame(data)
print("DataFrame with missing values:")
print(df)
```

Dataset with missing values:

	Price	Bedrooms	Bathrooms	Area (sq ft)	Year Built
0	405711.0	5.0	2	1659.0	2001
1	535829.0	1.0	1	2171.0	1980
2	NaN	1.0	1	1910.0	1993
3	252315.0	5.0	2	1423.0	2005
4	982371.0	3.0	3	2312.0	1981
5	459783.0	NaN	1	2985.0	1998
6	404137.0	1.0	3	2289.0	2004
7	222579.0	2.0	1	1697.0	1970
8	710581.0	2.0	2	NaN	1970
9	548242.0	1.0	2	1544.0	2006

2. Handling missing values using different functions:

Code:

```
# Handling missing values using different functions

# Drop rows with missing values
print("\n1. Drop rows with missing values:")
print(df.dropna())

# Fill missing values with a specific value
print("\n2. Fill missing values with a specific value (e.g., 0):")
print(df.fillna(0))
print("\n3. Fill missing values with the mean of the column:")
print(df.fillna(df.mean()))

# Interpolate missing values
print("\n5. Interpolate missing values:")
print(df.interpolate())
print("\n6. Forward fill missing values:")
print(df.ffill())
print("\n7. Backward fill missing values:")
print(df.bfill())
```

Output:

i. Using dropna()

```
1. Drop rows with missing values:
      Price  Bedrooms  Bathrooms  Area (sq ft)  Year Built
0  405711.0       5.0          2        1659.0        2001
1  535829.0       1.0          1        2171.0        1980
3  252315.0       5.0          2        1423.0        2005
4  982371.0       3.0          3        2312.0        1981
6  404137.0       1.0          3        2289.0        2004
7  222579.0       2.0          1        1697.0        1970
9  548242.0       1.0          2        1544.0        2006
```

ii. Using fillna()

```
2. Fill missing values with a specific value (e.g., 0):
      Price  Bedrooms  Bathrooms  Area (sq ft)  Year Built
0  405711.0       5.0          2        1659.0        2001
1  535829.0       1.0          1        2171.0        1980
2       0.0       1.0          1        1910.0        1993
3  252315.0       5.0          2        1423.0        2005
4  982371.0       3.0          3        2312.0        1981
5  459783.0       0.0          1        2985.0        1998
6  404137.0       1.0          3        2289.0        2004
7  222579.0       2.0          1        1697.0        1970
8  710581.0       2.0          2           0.0        1970
9  548242.0       1.0          2        1544.0        2006

3. Fill missing values with the mean of the column:
           Price  Bedrooms  Bathrooms  Area (sq ft)  Year Built
0  405711.000000  5.000000          2   1659.000000        2001
1  535829.000000  1.000000          1   2171.000000        1980
2  502394.222222  1.000000          1   1910.000000        1993
3  252315.000000  5.000000          2   1423.000000        2005
4  982371.000000  3.000000          3   2312.000000        1981
5  459783.000000  2.333333          1   2985.000000        1998
6  404137.000000  1.000000          3   2289.000000        2004
7  222579.000000  2.000000          1   1697.000000        1970
8  710581.000000  2.000000          2   1998.888889        1970
9  548242.000000  1.000000          2   1544.000000        2006
```

iii. Using interpolate(), ffill(), bfill()

```
5. Interpolate missing values:
      Price  Bedrooms  Bathrooms  Area (sq ft)  Year Built
0  405711.0       5.0          2        1659.0        2001
1  535829.0       1.0          1        2171.0        1980
2  394072.0       1.0          1        1910.0        1993
3  252315.0       5.0          2        1423.0        2005
4  982371.0       3.0          3        2312.0        1981
5  459783.0       2.0          1        2985.0        1998
6  404137.0       1.0          3        2289.0        2004
7  222579.0       2.0          1        1697.0        1970
8  710581.0       2.0          2        1620.5        1970
9  548242.0       1.0          2        1544.0        2006

6. Forward fill missing values:
      Price  Bedrooms  Bathrooms  Area (sq ft)  Year Built
0  405711.0       5.0          2        1659.0        2001
1  535829.0       1.0          1        2171.0        1980
2  535829.0       1.0          1        1910.0        1993
3  252315.0       5.0          2        1423.0        2005
4  982371.0       3.0          3        2312.0        1981
5  459783.0       3.0          1        2985.0        1998
6  404137.0       1.0          3        2289.0        2004
7  222579.0       2.0          1        1697.0        1970
8  710581.0       2.0          2        1697.0        1970
9  548242.0       1.0          2        1544.0        2006

7. Backward fill missing values:
      Price  Bedrooms  Bathrooms  Area (sq ft)  Year Built
0  405711.0       5.0          2        1659.0        2001
1  535829.0       1.0          1        2171.0        1980
2  252315.0       1.0          1        1910.0        1993
3  252315.0       5.0          2        1423.0        2005
4  982371.0       3.0          3        2312.0        1981
5  459783.0       1.0          1        2985.0        1998
6  404137.0       1.0          3        2289.0        2004
7  222579.0       2.0          1        1697.0        1970
8  710581.0       2.0          2        1544.0        1970
9  548242.0       1.0          2        1544.0        2006
```

Practical 20:

In the given 'weather.csv' file, compute statistical information using different aggregation and grouping functions.

Code:

```
import pandas as pd
df = pd.read_csv("/content/drive/MyDrive/Data Science/Practice Data/weather.csv")
print(df)
print(df.describe())

# aggregate
city_stats = df.groupby('city').agg({
    'temperature': ['mean', 'std', 'min', 'max'],
    'humidity': ['mean', 'std', 'min', 'max']
})

# Print the grouped statistics
print(city_stats)

# groupby()
city_sums = df.groupby('city')[['temperature', 'humidity']].sum()
print(city_sums)

city_temp_min = df.groupby('city')[['temperature']].min()
city_temp_max = df.groupby('city')[['temperature']].max()
print(city_temp_min,city_temp_max)
```

Dataset:

index	date	city	temperature	humidity
0	5/1/2017	new york	65	56
1	5/2/2017	new york	66	58
2	5/3/2017	new york	68	60
3	5/1/2017	mumbai	75	80
4	5/2/2017	mumbai	78	83
5	5/3/2017	mumbai	82	85
6	5/1/2017	beijing	80	26
7	5/2/2017	beijing	77	30
8	5/3/2017	beijing	79	35

Output:

i. Aggregate

```
       temperature   humidity
count     9.000000   9.000000
mean     74.444444  57.000000
std       6.424778  22.841848
min      65.000000  26.000000
25%      68.000000  35.000000
50%      77.000000  58.000000
75%      79.000000  80.000000
max      82.000000  85.000000
```

```
         temperature                       humidity
                mean       std min max       mean       std min max
city
beijing    78.666667  1.527525  77  80  30.333333  4.509250  26  35
mumbai     78.333333  3.511885  75  82  82.666667  2.516611  80  85
new york   66.333333  1.527525  65  68  58.000000  2.000000  56  60
```

ii. Grouping

```
          temperature  humidity
city
beijing           236        91
mumbai            235       248
new york          199       174
```

```
 City's Minimum temperature
           temperature
city
beijing            77
mumbai             75
new york           65
City's Maximum teperature
           temperature
city
beijing            80
mumbai             82
new york           68
```

Practical 21:

Read the excel file 'sample_pivot' in python and using the 'query ()' function filter out:

I. data having 'Units' less than 4.
II. data having 'Region'= West
III. data having 'Region'=West and 'Unit' less than 4
IV. data not having 'Region'= West.

Code:

```
import pandas as pd

# Read the Excel file
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
```

Dataset:

index	Date	Region	Type	Units	Sales
0	2020-07-11 00:00:00	East	Children's Clothing	18.0	306
1	2020-09-23 00:00:00	North	Children's Clothing	14.0	448
2	2020-04-02 00:00:00	South	Women's Clothing	17.0	425
3	2020-02-28 00:00:00	East	Children's Clothing	26.0	832
4	2020-03-19 00:00:00	West	Women's Clothing	3.0	33

Output:

I. data having 'Units' less than 4.

```
filter1 = df.query("`Units` < 4")
```

```
Filter 1 (Units < 4):
           Date Region                Type  Units  Sales
4   2020-03-19   West     Women's Clothing    3.0     33
28  2020-01-19   East       Men's Clothing    3.0     63
96  2020-11-13   East  Children's Clothing    3.0     72
118 2020-12-28   East  Children's Clothing    3.0     78
134 2020-09-04  North  Children's Clothing    3.0    184
135 2020-01-07   West     Women's Clothing    3.0    350
141 2020-02-12   East     Women's Clothing    3.0    330
173 2020-02-22  North  Children's Clothing    3.0    416
191 2020-05-31   East     Women's Clothing    3.0    124
249 2020-11-03  South       Men's Clothing    3.0    351
288 2020-03-14  North     Women's Clothing    3.0    552
310 2020-02-25   East       Men's Clothing    3.0    330
345 2020-01-21   East  Children's Clothing    3.0    870
355 2020-06-12   West  Children's Clothing    3.0    567
363 2020-09-08   East     Women's Clothing    3.0    676
368 2020-12-26   East       Men's Clothing    3.0     48
373 2020-08-20  North     Women's Clothing    3.0     84
390 2020-04-20  North     Women's Clothing    3.0    336
558 2020-10-06   West       Men's Clothing    3.0    462
563 2020-08-07   East     Women's Clothing    3.0    450
586 2020-09-25  North     Women's Clothing    3.0     87
609 2020-04-09  North     Women's Clothing    3.0     42
684 2020-04-08   East       Men's Clothing    3.0    806
686 2020-02-18   West       Men's Clothing    3.0    918
878 2020-09-03  North     Women's Clothing    3.0    560
900 2020-01-09  North     Women's Clothing    3.0    682
902 2020-09-26   East  Children's Clothing    3.0    462
```

II. data having 'Region'= West

```
filter2 = df.query("`Region` == 'West'")
```

```
           Date Region                Type  Units  Sales
4    2020-03-19   West     Women's Clothing    3.0     33
15   2020-11-26   West       Men's Clothing   27.0    864
21   2020-06-23   West     Women's Clothing   18.0    288
24   2020-06-18   West       Men's Clothing    5.0     70
30   2020-07-13   West  Children's Clothing   30.0    450
..          ...    ...                  ...    ...    ...
969  2020-07-20   West     Women's Clothing   25.0    442
970  2020-11-28   West     Women's Clothing   12.0    770
972  2020-09-21   West       Men's Clothing   35.0    437
985  2020-02-08   West       Men's Clothing   32.0    928
991  2020-11-17   West       Men's Clothing   27.0    486

[136 rows x 5 columns]
```

III. data having 'Region'=West and 'Unit' less than 4

```
filter3 = df.query("`Region` == 'West' and `Units` < 4")
```

```
Filter 3 (Region='West' & Units < 4):
           Date Region                Type  Units  Sales
4    2020-03-19   West     Women's Clothing    3.0     33
135  2020-01-07   West     Women's Clothing    3.0    350
355  2020-06-12   West  Children's Clothing    3.0    567
558  2020-10-06   West       Men's Clothing    3.0    462
686  2020-02-18   West       Men's Clothing    3.0    918
```

IV. data not having 'Region'= West.

```
filter4 = df.query("`Region` != 'West'")
```

```
Filter 4 (Region != 'West'):
           Date Region                 Type  Units  Sales
0    2020-07-11   East  Children's Clothing   18.0    306
1    2020-09-23  North  Children's Clothing   14.0    448
2    2020-04-02  South     Women's Clothing   17.0    425
3    2020-02-28   East  Children's Clothing   26.0    832
5    2020-02-05  North     Women's Clothing   33.0    627
..          ...    ...                  ...    ...    ...
995  2020-02-11   East  Children's Clothing   35.0    735
996  2020-12-25  North       Men's Clothing    NaN   1155
997  2020-08-31  South       Men's Clothing   13.0    208
998  2020-08-23  South     Women's Clothing   17.0    493
999  2020-08-17  North     Women's Clothing   25.0    300

[864 rows x 5 columns]
```

Practical 22:

Read the excel file 'sample_pivot' in python and perform any 5 Vectorize string operations.

Code:

```
import pandas as pd

# Read the Excel file
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
```

Dataset:

index	Date	Region	Type	Units	Sales
0	2020-07-11 00:00:00	East	Children's Clothing	18.0	306
1	2020-09-23 00:00:00	North	Children's Clothing	14.0	448
2	2020-04-02 00:00:00	South	Women's Clothing	17.0	425
3	2020-02-28 00:00:00	East	Children's Clothing	26.0	832
4	2020-03-19 00:00:00	West	Women's Clothing	3.0	33

Output: 5 Vectorized string operations

1.

```
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
df['Type'] = df['Type'].str.upper()
df.head()
```

	Date	Region	Type	Units	Sales
0	2020-07-11	East	CHILDREN'S CLOTHING	18.0	306
1	2020-09-23	North	CHILDREN'S CLOTHING	14.0	448
2	2020-04-02	South	WOMEN'S CLOTHING	17.0	425
3	2020-02-28	East	CHILDREN'S CLOTHING	26.0	832
4	2020-03-19	West	WOMEN'S CLOTHING	3.0	33

2.

```
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
df['Type'] = df['Type'].str[:3]
df.head()
```

	Date	Region	Type	Units	Sales
0	2020-07-11	East	Chi	18.0	306
1	2020-09-23	North	Chi	14.0	448
2	2020-04-02	South	Wom	17.0	425
3	2020-02-28	East	Chi	26.0	832
4	2020-03-19	West	Wom	3.0	33

3.

```
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
df['Region'] = df['Region'].str.replace('East', 'Pink')
df.head()
```

	Date	Region	Type	Units	Sales
0	2020-07-11	Pink	Children's Clothing	18.0	306
1	2020-09-23	North	Children's Clothing	14.0	448
2	2020-04-02	South	Women's Clothing	17.0	425
3	2020-02-28	Pink	Children's Clothing	26.0	832
4	2020-03-19	West	Women's Clothing	3.0	33

4.

```
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
df['Region'] = df['Region'].str.contains('South')
df.head()
```

	Date	Region	Type	Units	Sales
0	2020-07-11	False	Children's Clothing	18.0	306
1	2020-09-23	False	Children's Clothing	14.0	448
2	2020-04-02	True	Women's Clothing	17.0	425
3	2020-02-28	False	Children's Clothing	26.0	832
4	2020-03-19	False	Women's Clothing	3.0	33

5.

```
df = pd.read_excel('/content/drive/MyDrive/Data Science/Practice Data/sample_pivot.xlsx')
df['Region'] = df['Region'].str.strip()
df.head()
```

	Date	Region	Type	Units	Sales
0	2020-07-11	East	Children's Clothing	18.0	306
1	2020-09-23	North	Children's Clothing	14.0	448
2	2020-04-02	South	Women's Clothing	17.0	425
3	2020-02-28	East	Children's Clothing	26.0	832
4	2020-03-19	West	Women's Clothing	3.0	33

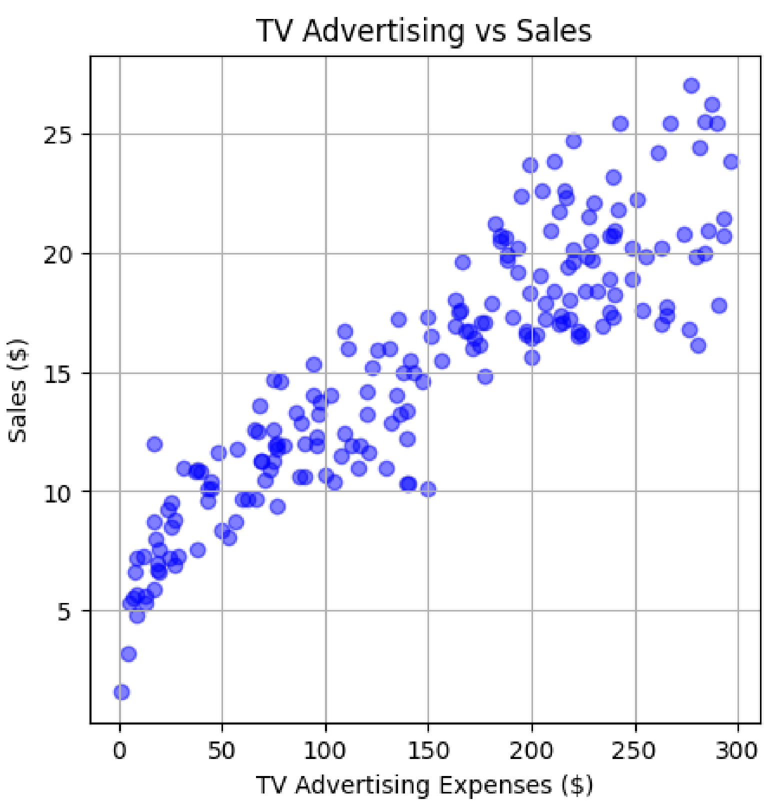
Practical 23:

Construct scatter plot on 'advertising.csv' file to find the correlation between the TV and sales.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("/content/drive/MyDrive/Data Science/Practice Data/advertising.csv")

plt.figure(figsize=(5, 5))
plt.scatter(df['TV'], df['Sales'], color='blue', alpha=0.5)
plt.title('TV Advertising vs Sales')
plt.xlabel('TV Advertising Expenses ($)')
plt.ylabel('Sales ($)')
plt.grid(True)
plt.show()
```

Graph:



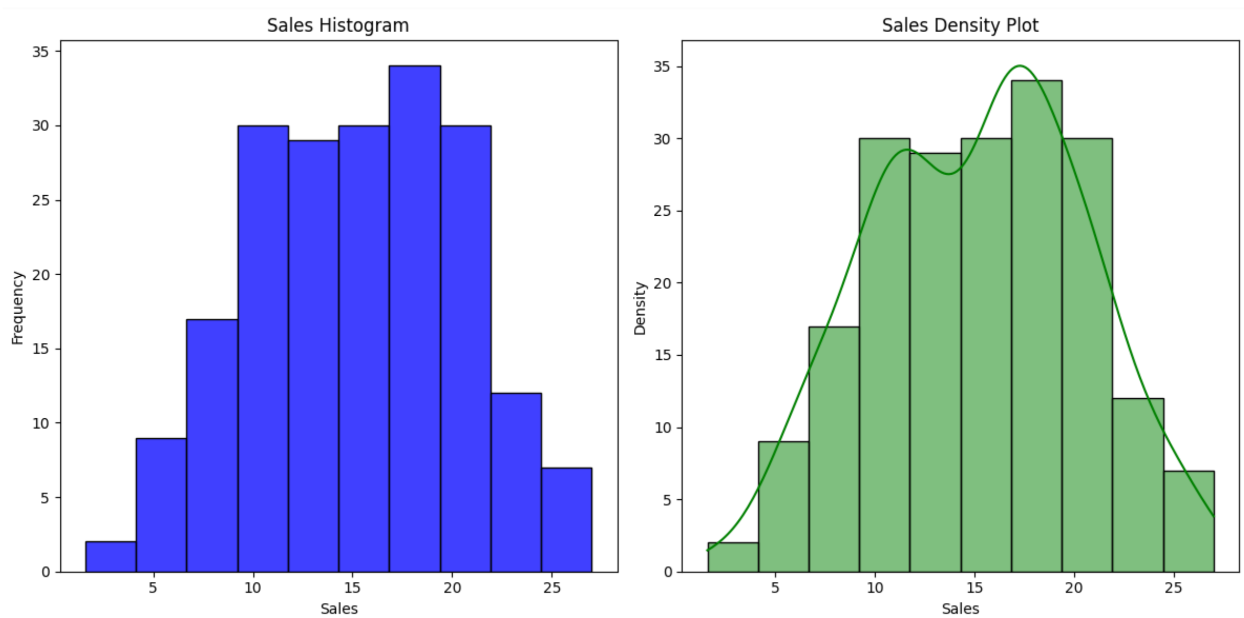
Practical 24:

Construct histogram and density plot on ‘advertising.csv’ file.

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("/content/drive/MyDrive/Data Science/Practice Data/advertising.csv")

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

sns.histplot(df['Sales'], kde=False, ax=axes[0], color='blue')
axes[0].set_title('Sales Histogram')
axes[0].set_xlabel('Sales')
axes[0].set_ylabel('Frequency')

sns.histplot(df['Sales'], kde=True, ax=axes[1], color='green')
axes[1].set_title('Sales Density Plot')
axes[1].set_xlabel('Sales')
axes[1].set_ylabel('Density')

plt.tight_layout()
plt.show()
```

Graph:



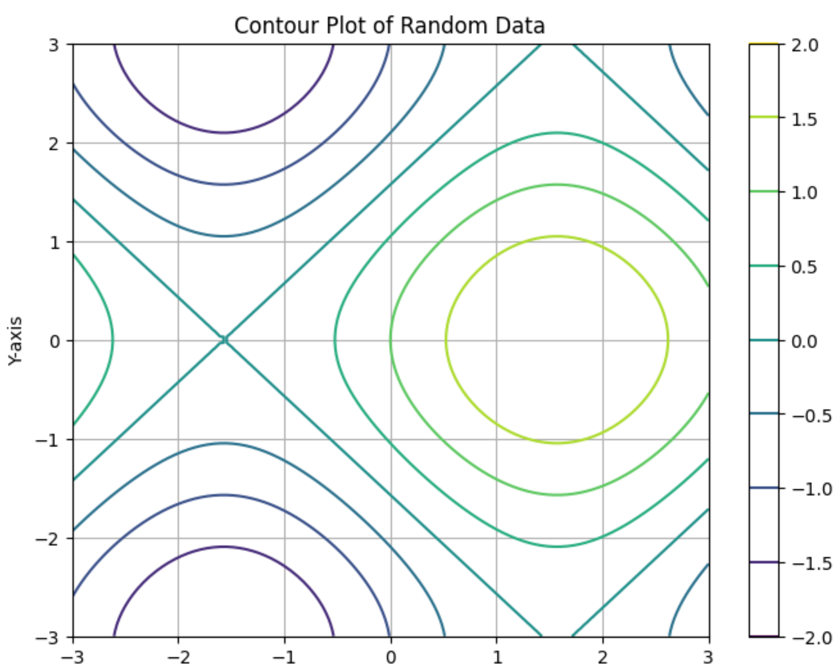
Practical 25:

Construct Contour plot by generating random data using NumPy library

Code:

```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-3.0, 3.0, 100)
y = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(X) + np.cos(Y)


plt.figure(figsize=(8, 6))
contour = plt.contour(X, Y, Z, cmap='viridis')
plt.colorbar(contour)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Contour Plot of Random Data')
plt.grid(True)
plt.show()
```

Contour Plot:



Practical 26:

Perform Simple Linear Regression on 'advertising.csv' to find the impact on total sales due to TV

Dataset:

index	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
5	8.7	48.9	75.0	7.2
6	57.5	32.8	23.5	11.8
7	120.2	19.6	11.6	13.2
8	8.6	2.1	1.0	4.8
9	199.8	2.6	21.2	15.6

Code:

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

df = pd.read_csv("/content/drive/MyDrive/Data Science/Practice Data/advertising.csv")

X = df[['TV']]
y = df['Sales']

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

print('Intercept:', model.intercept_)
print('Coefficient:', model.coef_[0])

predictions = model.predict(X)
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, predictions, color='red', linewidth=2, label='Regression line')
plt.title('TV Advertising vs Sales')
plt.xlabel('TV Advertising Expenses ($)')
plt.ylabel('Sales ($)')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Intercept: 6.974821488229891
Coefficient: 0.055464770469558874
```