# Que. 1

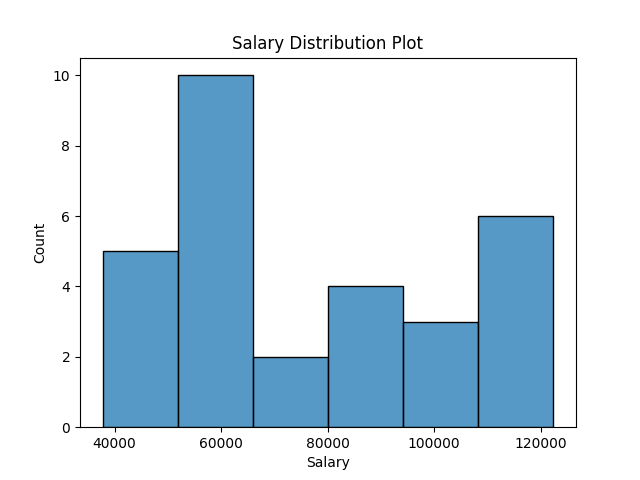
class MaxHeap:  
 def \_\_init\_\_(self):  
 self.heap = []  
  
 def insert(self, val):  
 self.heap.append(val)  
 self.\_heapify\_up(len(self.heap) - 1)  
  
 def delete(self, val):  
 if len(self.heap) == 0:  
 raise IndexError("delete from empty heap")  
 idx = self.heap.index(val)  
 self.\_swap(idx, len(self.heap) - 1)  
 self.heap.remove(val)  
 self.\_heapify\_down(idx)  
  
 def get\_max(self):  
 if len(self.heap) == 0:  
 raise IndexError("get\_max from empty heap")  
 return self.heap[0]  
  
 def \_heapify\_up(self, idx):  
 parent = (idx - 1) // 2  
 if idx > 0 and self.heap[parent] < self.heap[idx]:  
 self.\_swap(parent, idx)  
 self.\_heapify\_up(parent)  
  
 def \_heapify\_down(self, idx):  
 left = 2 \* idx + 1  
 right = 2 \* idx + 2  
 largest = idx  
 if left < len(self.heap) and self.heap[left] > self.heap[largest]:  
 largest = left  
 if right < len(self.heap) and self.heap[right] > self.heap[largest]:  
 largest = right  
 if largest != idx:  
 self.\_swap(idx, largest)  
 self.\_heapify\_down(largest)  
  
 def \_swap(self, i, j):  
 self.heap[i], self.heap[j] = self.heap[j], self.heap[i]  
  
  
*# Example usage*heap = MaxHeap()  
heap.insert(5)  
heap.insert(12)  
heap.insert(3)  
print(heap.heap) *# Output: [12, 5, 3]*print(heap.get\_max()) *# Output: 12*heap.delete(5)  
print(heap.heap) *# Output: [12, 3]*print(heap.get\_max()) *# Output: 12*

# Que. 2

import requests  
from requests.exceptions import RequestException  
  
def download\_urls(urls):  
 max\_retries = 3  
 results = {}  
  
 for url in urls:  
 retry\_count = 0  
 success = False  
  
 while (retry\_count < max\_retries and not success):  
 try:  
 response = requests.get(url)  
 response.raise\_for\_status() *# Raise error for bad response status  
  
 # If successful, store the content in results* results[url] = response.content  
 success = True  
  
 except requests.exceptions.RequestException as e:  
 *# Handle connection errors, HTTP errors, and retries* print(f"Attempt {retry\_count+1} failed for URL: {url}. Error: {e}")  
 retry\_count += 1  
  
 except Exception as e:  
 *# Handle other unexpected exceptions* print(f"Unexpected error occurred for URL: {url}. Error: {e}")  
 retry\_count += 1  
  
 return results  
  
*# usage*urls = [  
 'https://facebook.com',  
 'https://youtube.com',  
 'https://www.python.org',  
 'https://google.com'  
]  
  
downloaded\_contents = download\_urls(urls)  
print(downloaded\_contents.keys()) *# List of URLs successfully downloaded  
  
# output : dict\_keys(['https://facebook.com', 'https://youtube.com', 'https://www.python.org', 'https://google.com'])*

# Que. 3

*# Import necessary libraries*import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from pandas.core.common import random\_state  
from sklearn.linear\_model import LinearRegression  
  
*# Get dataset*df\_sal = pd.read\_csv('D:\python\core1\practice\Salary\_Data.csv')  
df\_sal.head()  
  
*# Describe data*df\_sal.describe()  
  
*# Data distribution*plt.title('Salary Distribution Plot')  
sns.histplot(df\_sal['Salary'])  
plt.show()  
  
*# Relationship between Salary and Experience*plt.scatter(df\_sal['YearsExperience'], df\_sal['Salary'], color = 'lightcoral')  
plt.title('Salary vs Experience')  
plt.xlabel('Years of Experience')  
plt.ylabel('Salary')  
plt.box(False)  
plt.show()  
  
*# Splitting variables*X = df\_sal.iloc[:, :1] *# independent*y = df\_sal.iloc[:, 1:] *# dependent  
  
# Splitting dataset into test/train*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)  
  
*# Regressor model*regressor = LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
*# Prediction result*y\_pred\_test = regressor.predict(X\_test) *# predicted value of y\_test*y\_pred\_train = regressor.predict(X\_train) *# predicted value of y\_train  
  
# Prediction on training set*plt.scatter(X\_train, y\_train, color = 'lightcoral')  
plt.plot(X\_train, y\_pred\_train, color = 'firebrick')  
plt.title('Salary vs Experience (Training Set)')  
plt.xlabel('Years of Experience')  
plt.ylabel('Salary')  
plt.legend(['X\_train/Pred(y\_test)', 'X\_train/y\_train'], title = 'Sal/Exp', loc='best', facecolor='white')  
plt.box(False)  
plt.show()  
  
*# Prediction on test set*plt.scatter(X\_test, y\_test, color = 'lightcoral')  
plt.plot(X\_train, y\_pred\_train, color = 'firebrick')  
plt.title('Salary vs Experience (Test Set)')  
plt.xlabel('Years of Experience')  
plt.ylabel('Salary')  
plt.legend(['X\_train/Pred(y\_test)', 'X\_train/y\_train'], title = 'Sal/Exp', loc='best', facecolor='white')  
plt.box(False)  
plt.show()  
  
*# Regressor coefficients and intercept*print(f'Coefficient: {regressor.coef\_}')  
print(f'Intercept: {regressor.intercept\_}')  
  
*#output:  
#Coefficient: [[9312.57512673]]  
#Intercept: [26780.09915063]*



# Que 5

import pandas as pd  
from sklearn.preprocessing import StandardScaler, LabelEncoder  
  
  
def preprocess\_data(df):  
 *# Handle missing values* df = handle\_missing\_values(df)  
  
 *# Normalize numerical columns* df = normalize\_numerical\_columns(df)  
  
 *# Encode categorical columns* df = encode\_categorical\_columns(df)  
  
 return df  
  
  
def handle\_missing\_values(df):  
 *# Fill missing values with mean for numerical columns* numerical\_cols = df.select\_dtypes(include=['number']).columns  
 df[numerical\_cols] = df[numerical\_cols].fillna(df[numerical\_cols].mean())  
  
 *# Fill missing values with mode for categorical columns* categorical\_cols = df.select\_dtypes(include=['object']).columns  
 df[categorical\_cols] = df[categorical\_cols].fillna(df[categorical\_cols].mode().iloc[0])  
  
 return df  
  
  
def normalize\_numerical\_columns(df):  
 scaler = StandardScaler()  
 numerical\_cols = df.select\_dtypes(include=['number']).columns  
 df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])  
 return df  
  
  
def encode\_categorical\_columns(df):  
 encoder = LabelEncoder()  
 categorical\_cols = df.select\_dtypes(include=['object']).columns  
 df[categorical\_cols] = df[categorical\_cols].apply(encoder.fit\_transform)  
 return df  
  
  
*# Example DataFrame*data = {  
 'A': [1, 2, None, 4],  
 'B': ['X', 'Y', 'Z', 'X'],  
 'C': [0.1, 0.5, 0.3, None],  
 'D': [100, 200, 150, 180]  
}  
df = pd.DataFrame(data)  
  
*# Preprocess the DataFrame*preprocessed\_df = preprocess\_data(df)  
  
print("Preprocessed DataFrame:")  
print(preprocessed\_df)  
  
  
"""  
Output:  
Preprocessed DataFrame:  
 A B C D  
0 -1.234427 0 -1.414214 -1.526564  
1 -0.308607 1 1.414214 1.128330  
2 0.000000 2 0.000000 -0.199117  
3 1.543033 0 0.000000 0.597351  
  
"""

# Que 5

def fibonacci(n):  
 *"""  
 Compute the nth Fibonacci number using recursion.  
 """* if(n <= 1):  
 return n  
 else:  
 return fibonacci(n-1) + fibonacci(n-2)  
  
*# Example usage:*print(fibonacci(10))  
  
*# output: 55*

# Que 6

*# divide number*try:  
 a=int(input("Enter first number :"))  
 b=int(input("Enter second number:"))  
 c=a/b  
 print("The result is :",c)  
except ZeroDivisionError:  
 print("Do Not enter second number zero")  
except ValueError:  
 print("system not responding")  
print("success")  
  
'''  
output:  
Enter first number :40  
Enter second number:0  
Do Not enter second number zero  
success  
'''

# Que 7

import time  
  
def measure\_execution\_time(func):  
 def wrapper(\*args, \*\*kwargs):  
 start\_time = time.time()  
 result = func(\*args, \*\*kwargs)  
 end\_time = time.time()  
 execution\_time = end\_time - start\_time  
 print(f"Function {func.\_\_name\_\_} took {execution\_time:.4f} seconds to execute")  
 return result  
 return wrapper  
  
*# Example usage*@measure\_execution\_time  
def calculate\_multiply(numbers):  
 tot = 1  
 for x in numbers:  
 tot \*= x  
 return tot  
  
*# Call the decorated function*result = calculate\_multiply([1, 2, 3, 4, 5])  
print("Result:", result)  
  
*# Function calculate\_multiply took 0.0000 seconds to execute  
# Result: 120*

# Que 8

def addition(num1, num2):  
 a = num1 + num2  
 print("addition was performed on the two numbers ", num1, ' and ', num2)  
 return a  
  
  
def subtraction(num1, num2):  
 s = num1 - num2  
 print("subtraction was performed on the two numbers ", num1, ' and ', num2)  
 return s  
  
  
def multiplication(num1, num2):  
 t = num1 \* num2  
 print("multiplication was performed on the two numbers ", num1, ' and ', num2)  
 return t  
  
  
def division(num1, num2):  
 d = num1 / num2  
 print("division was performed on the two numbers ", num1, ' and ', num2)  
 return d  
  
  
def calculate(num1, num2, string):  
 result = None  
 if string == '+':  
 result = addition(num1, num2)  
 elif string == '-':  
 result = subtraction(num1, num2)  
 elif string == '\*':  
 result = multiplication(num1, num2)  
 elif string == '/':  
 result = division(num1, num2)  
  
print(addition(num1=4,num2=5))  
print(subtraction(num1=4,num2=5))  
print(multiplication(num1=4,num2=5))  
print(division(num1=4,num2=5))  
  
'''  
output:  
  
addition was performed on the two numbers 4 and 5  
9  
subtraction was performed on the two numbers 4 and 5  
-1  
multiplication was performed on the two numbers 4 and 5  
20  
division was performed on the two numbers 4 and 5  
0.8  
'''

# Que 9

import random  
import string  
  
def generate\_random\_password(length=12):  
 *# Define characters to use in the password* lowercase\_letters = string.ascii\_lowercase  
 uppercase\_letters = string.ascii\_uppercase  
 digits = string.digits  
 special\_characters = string.punctuation   
  
 *# Combine all characters* all\_characters = lowercase\_letters + uppercase\_letters + digits + special\_characters  
  
 *# Generate password* password = ''.join(random.choice(all\_characters) for \_ in range(length))  
 return password  
  
*# Example usage:*password = generate\_random\_password()  
print("Generated Password:", password)

# Que 10

def transpose\_matrix(matrix):  
 *# Check if matrix is empty* if not matrix:  
 return []  
  
 *# Dimensions of the matrix* rows = len(matrix)  
 cols = len(matrix[0])  
  
 *# Create a new matrix to store the transpose* transpose = [[0 for \_ in range(rows)] for \_ in range(cols)]  
  
 *# Compute the transpose* for i in range(rows):  
 for j in range(cols):  
 transpose[j][i] = matrix[i][j]  
  
 return transpose  
  
*# Example matrix*matrix = [  
 [1, 2, 3],  
 [4, 5, 6]  
]  
  
*# Get the transpose of the matrix*transposed\_matrix = transpose\_matrix(matrix)  
  
*# Print the original and transposed matrices*print("Original Matrix:")  
for row in matrix:  
 print(row)  
  
print("\nTransposed Matrix:")  
for row in transposed\_matrix:  
 print(row)  
  
'''  
output:  
Original Matrix:  
[1, 2, 3]  
[4, 5, 6]  
  
Transposed Matrix:  
[1, 4]  
[2, 5]  
[3, 6]  
  
'''