worksheet_02

February 1, 2024

1 Worksheet 02

Name: Haya Naveed UID: U16758779

1.0.1 Topics

- Effective Programming

1.0.2 Effective Programming

a) What is a drawback of the top down approach? #implement functions last The topdown approach is a problem-solving and design strategy that involves breaking down a complex problem into smaller, more manageable sub-problems or modules. The process starts with the overarching problem and progressively refines it into detailed and specific tasks. It can limit creativity and slow down the coding process since you need to write psuedocode for the entire code first (i.e. there's an overhead of high-level design). Top-down approach is based on a fixed and well-defined problem statement. Thus, if the problem requirements, change it can be challenging to adapt the top-down structure; thus, top-down approach can be inflexible and might not work well for teams that anticipate a lot of changes to their code/projects.

b) What is a drawback of the bottom up approach? #implement functions first The bottom up approach begins with the construction of individual components or modules/functions. These components are gradually combined to create larger subsystems and, eventually, the entire system. It requires more planning, because we start with building blocks to create more coherent/bigger functions, and planning bears a time cost. Bottom up approach may not provide a clear overall vision of the system: developers start by building individual components without fully understanding how those components would work together as a whole, which can lead to challenges in ensuring that the componenents work together correctly to deliver the desired functionality.

c) What are 3 things you can do to have a better debugging experience? Do one thing, compartmentalize, write manny functions with smaller bodies rather than fewer functions with large bodes, minimize side effects of your code.

d) (Optional) Follow along with the live coding. You can write your code here:

[5]:
```
class Board:
    def __init__(self): #can have param n later for n queens problem
        self.board = [["-" for _ in range(8)] for _ in range(8)]
```

```
    def __repr__(self):
        res = ""
        for row in range(8): #range(len(self.board))
            for col in range(8): #range(len(self.board[row]))
                res += self.board[row][col]
                res += " "
            res += '\n'
        return res

    def set_queen_at(self,row,col):
        self.board[row][col] = "Q"

    def unset_queen_on_row(self,row):
        self.board[row] = ["-" for _ in range(8)]

    def is_valid_move(self, row, col):
        if not self.is_valid_row(row,col):
            return False
        if not self.is_valid_col(row,col):
            return False
        if not self.is_valid_diag(row,col):
            return False
        return True

    def is_valid_row(self,row,col):
        for j in range(8):
            if j!= col and self.board[row][j] == "Q":
                return False
        reutrn True

    def is_valid_col(self,row,col):
        for i in range(8):
            if i!= row and self.board[i][col] == "Q":
                return False
        reutrn True



    def get_queen_on_row(self, row):
        for i in range(8):
            if self.board[row][i] == "Q":
                return i
        raise ValueError("no queen on row")

    def find_solution(self): #we're done when we've placed a queen on the last␣
↪row
        row = 0
```

```
        col = 0

        while row < 8:
            #we are searching for a solution
            if self.is_valid_move(row,col):
                self.set_queen_at(row,col)
                row+=1 #bc can at most set one queen per row so move on to next␣
↪row after setting queen in the row
                col = 0
            else:
                col+=1
                if col >= 8: #if its out of bounds
                    #we weren't able to place a queen on this row so we need to␣
↪backtrack and adjust the postion of
                    #the queen on the previous row
                    col = self.get_queen_on_row(row-1) #get position, then look␣
↪BEYOND that position
                    col += 1
                    row -= 1


        #we have found a solution
        print("Found a solution: ")
        print(self) #prints the soln


test = Board()
print(test)
test.set_queen_at(1,1)
print(test)
test.unset_queen_on_row(1)
print(test)
```

```
  Cell In[5], line 22
    return True
    ^
IndentationError: expected an indented block
```

1.1 Exercise

This exercise will use the Titanic dataset (https://www.kaggle.com/c/titanic/data). Download the file named `train.csv` and place it in the same folder as this notebook.

The goal of this exercise is to practice using pandas methods. If your:

1. code is taking a long time to run

2. code involves for loops or while loops
3. code spans multiple lines

look through the pandas documentation for alternatives. This cheat sheet may come in handy.

a) Complete the code below to read in a filepath to the `train.csv` and returns the DataFrame.

[55]:
```
import pandas as pd

df = pd.read_csv("train.csv")
df.describe()
```

[55]:

	PassengerId	Survived	Pclass	Age	SibSp	\
count	891.000000	891.000000	891.000000	714.000000	891.000000	
mean	446.000000	0.383838	2.308642	29.699118	0.523008	
std	257.353842	0.486592	0.836071	14.526497	1.102743	
min	1.000000	0.000000	1.000000	0.420000	0.000000	
25%	223.500000	0.000000	2.000000	20.125000	0.000000	
50%	446.000000	0.000000	3.000000	28.000000	0.000000	
75%	668.500000	1.000000	3.000000	38.000000	1.000000	
max	891.000000	1.000000	3.000000	80.000000	8.000000	

	Parch	Fare
count	891.000000	891.000000
mean	0.381594	32.204208
std	0.806057	49.693429
min	0.000000	0.000000
25%	0.000000	7.910400
50%	0.000000	14.454200
75%	0.000000	31.000000
max	6.000000	512.329200

b) Complete the code so it returns the number of rows that have at least one empty column value

[56]:
```
#print(df.shape[0]) #print no of rows
empty_rows_count = df[df.isnull().any(axis=1)].shape[0]
print("there are " +  str(empty_rows_count) + " rows with at least one empty␣
 ↪value")
```

```
there are 708 rows with at least one empty value
```

[57]:
```
df.columns
```

[57]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

c) Complete the code below to remove all columns with more than 200 NaN values

[58]:
```
# for column in df.columns:
#     num_nan = df[column].isna().sum()
#     print(f"Number of NaN values in '{column}': {num_nan}")

#thresh parameter in the dropna method specifies the minimum number of non-null␣
 ↪values required for a column or row to be retained.
df = df.dropna(axis=1, thresh=df.shape[0]-200)
df.columns
```

[58]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked'],
      dtype='object')
```

d) Complete the code below to replaces `male` with 0 and `female` with 1

[59]:
```
df['Sex'] = df['Sex'].replace({'male': 0, 'female': 1})
df.head(10)
```

[59]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	1	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0			
5	6	0	3	Moran, Mr. James	0	NaN	0	0			
6	7	0	1	McCarthy, Mr. Timothy J	0	54.0	0	0			
7	8	0	3	Palsson, Master. Gosta Leonard	0	2.0	3	1			
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	1	27.0	0	2			
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	1	14.0	1	0			

```
4            373450   8.0500        S
5            330877   8.4583        Q
6             17463  51.8625        S
7            349909  21.0750        S
8            347742  11.1333        S
9            237736  30.0708        C
```

e) Complete the code below to add four columns `First Name`, `Middle Name`, `Last Name`, and `Title` corresponding to the value in the `name` column.

For example: `Braund, Mr. Owen Harris` would be:

First Name	Middle Name	Last Name	Title
Owen	Harris	Braund	Mr

Anything not clearly one of the above 4 categories can be ignored.

[60]:
```
#df[['First Name', 'Middle Name', 'Last Name', 'Title']] = ...
#df[['Last Name']] = df['Name'].str.split(',')[0].str.strip()

df[['Last Name', 'Title', 'First Name', 'Middle Name']] = df['Name'].str.
 ↪split(' ', n=3, expand=True)

#some processing to make the column values look cleaner
df['Middle Name'] = df['Middle Name'].str.split(' ', n=1).str[0]
df['Middle Name'] = df['Middle Name'].apply(lambda x: None if x != None and x.
 ↪startswith('(') else x)
df['Last Name'] = df['Last Name'].str.rstrip(',')

df.head(10)
```

[60]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
5            6         0       3
6            7         0       1
7            8         0       3
8            9         1       3
9           10         1       2

                                                 Name  Sex   Age  SibSp  Parch  \
0                             Braund, Mr. Owen Harris    0  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…    1  38.0      1      0
2                              Heikkinen, Miss. Laina    1  26.0      0      0
```

```
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    1  35.0      1      0
4                            Allen, Mr. William Henry    0  35.0      0      0
5                                    Moran, Mr. James    0   NaN      0      0
6                             McCarthy, Mr. Timothy J    0  54.0      0      0
7                      Palsson, Master. Gosta Leonard    0   2.0      3      1
8   Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)    1  27.0      0      2
9                 Nasser, Mrs. Nicholas (Adele Achem)    1  14.0      1      0

              Ticket     Fare Embarked  Last Name    Title First Name  \
0         A/5 21171   7.2500        S     Braund      Mr.       Owen
1          PC 17599  71.2833        C    Cumings     Mrs.       John
2  STON/O2. 3101282   7.9250        S  Heikkinen    Miss.      Laina
3            113803  53.1000        S   Futrelle     Mrs.    Jacques
4            373450   8.0500        S      Allen      Mr.    William
5            330877   8.4583        Q      Moran      Mr.      James
6             17463  51.8625        S   McCarthy      Mr.    Timothy
7            349909  21.0750        S    Palsson  Master.      Gosta
8            347742  11.1333        S    Johnson     Mrs.      Oscar
9            237736  30.0708        C     Nasser     Mrs.   Nicholas

  Middle Name
0     Harris
1    Bradley
2       None
3      Heath
4      Henry
5       None
6          J
7    Leonard
8          W
9       None
```

f) Complete the code below to replace all missing ages with the average age

[61]:
```
df['Age'] = df['Age'].fillna(df['Age'].mean())
df.head(10)
```

[61]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
5            6         0       3
6            7         0       1
7            8         0       3
8            9         1       3
```

```
9           10         1       2
```

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	0	22.000000	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th…	1	38.000000	1	
2	Heikkinen, Miss. Laina	1	26.000000	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.000000	1	
4	Allen, Mr. William Henry	0	35.000000	0	
5	Moran, Mr. James	0	29.699118	0	
6	McCarthy, Mr. Timothy J	0	54.000000	0	
7	Palsson, Master. Gosta Leonard	0	2.000000	3	
8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	1	27.000000	0	
9	Nasser, Mrs. Nicholas (Adele Achem)	1	14.000000	1	

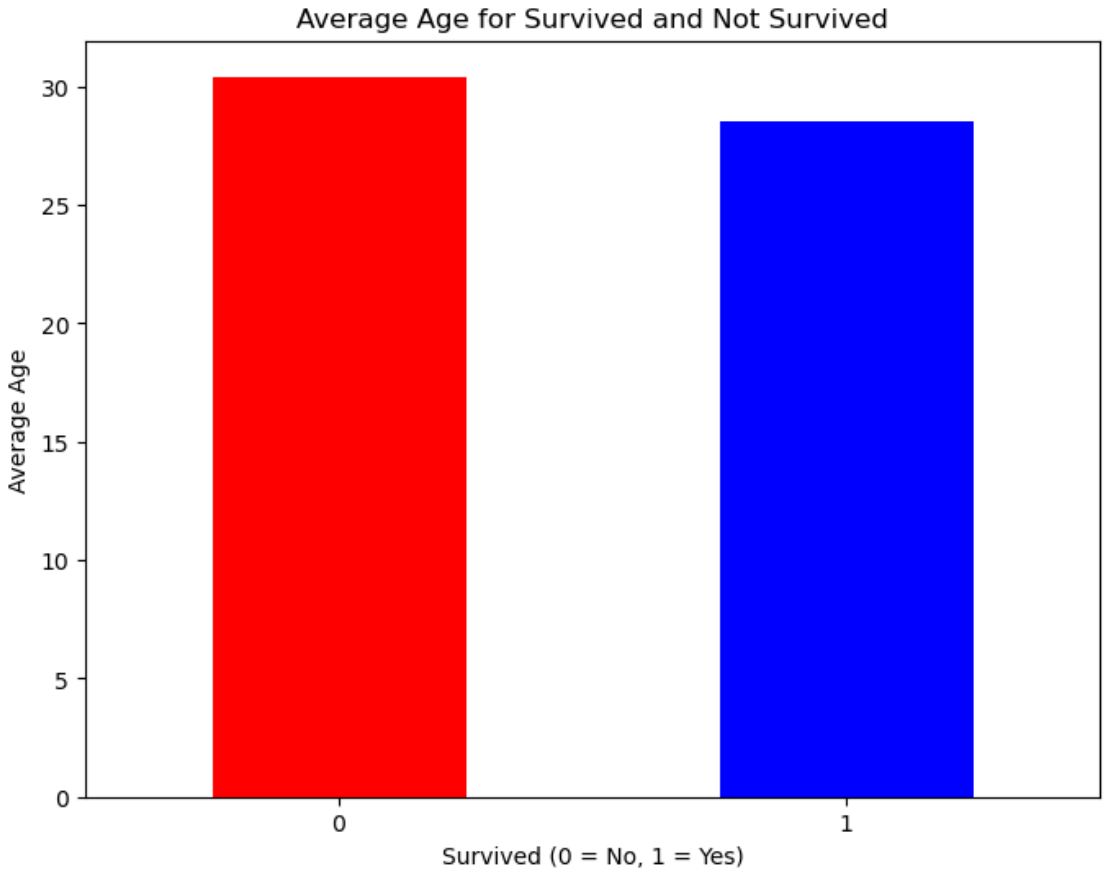
	Parch	Ticket	Fare	Embarked	Last Name	Title	First Name	\
0	0	A/5 21171	7.2500	S	Braund	Mr.	Owen	
1	0	PC 17599	71.2833	C	Cumings	Mrs.	John	
2	0	STON/O2. 3101282	7.9250	S	Heikkinen	Miss.	Laina	
3	0	113803	53.1000	S	Futrelle	Mrs.	Jacques	
4	0	373450	8.0500	S	Allen	Mr.	William	
5	0	330877	8.4583	Q	Moran	Mr.	James	
6	0	17463	51.8625	S	McCarthy	Mr.	Timothy	
7	1	349909	21.0750	S	Palsson	Master.	Gosta	
8	2	347742	11.1333	S	Johnson	Mrs.	Oscar	
9	0	237736	30.0708	C	Nasser	Mrs.	Nicholas	

	Middle Name
0	Harris
1	Bradley
2	None
3	Heath
4	Henry
5	None
6	J
7	Leonard
8	W
9	None

g) Plot a bar chart of the average age of those that survived and did not survive. Briefly comment on what you observe.

[62]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
df.groupby('Survived')['Age'].mean().plot(kind='bar', color=['red', 'blue'])
plt.title('Average Age for Survived and Not Survived')
plt.xlabel('Survived (0 = No, 1 = Yes)')
```

```
plt.ylabel('Average Age')
plt.xticks(rotation=0)
plt.show()


average_age = df.groupby('Survived')['Age'].mean()
print(f"Average Age of those who Survived: {average_age[1]:.2f}")
print(f"Average Age of those who Did Not Survive: {average_age[0]:.2f}")

# The average age of those who survived (28.55 years) is pretty similar to␣
 ↪those who did not survive (30.42 years),
# as can be seen in bar chart below - bars for both groups have pretty similar␣
 ↪height. The average age in the group
# that did not survive is only slightly higher than that of the group that␣
 ↪survived. As the plot show, the red bar is
# slightly higher


# df.groupby('Survived') groups the df by the 'Survived' column. It essentially␣
 ↪creates two groups, one for individuals who did not survive (Survived=0) and␣
 ↪another for those who survived (Survived=1).
# ['Age']: After grouping, we select the 'Age' column from each group.
# .mean(): Finally, we apply the .mean() function to calculate the average age␣
 ↪for each group.
```




```
Average Age of those who Survived: 28.55
Average Age of those who Did Not Survive: 30.42
```

[ ]: