Worksheet 02

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

Topics

- Effective Programming

Effective Programming

a) What is a drawback of the top down approach?

The drawback of the top down approach is that with a program language like Python, it does not work well since we do not have runnable code until the end and therefore cannot run the code more often. Therefore, it is better with type languages where we can run the type checker on the functions that we declare and verify that the inputs and outputs align with one another.

b) What is a drawback of the bottom up approach?

The drawback of the bottom up approach is that it requires a good planning phase but it is a very difficult step since not everyone is good at planning and could be a bit waste of time.

c) What are 3 things you can do to have a better debugging experience?

The three things we can do to have a better debugging experience is to read the error, re-read my code by taking considerate amount of time, and do sanity check where we can.

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

In [ ]:

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.

In [78]:
```
import pandas as pd

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

Out[78]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

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

In [79]:
```
print("there are " +  str(df.isna().any(axis=1).sum()) + " rows with at least one empty value")
```

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

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

In [80]:
```
df = df[df.columns[df.isna().sum() <= 200]]
df.columns
```

Out[80]:
```
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

In [81]:
```
df['Sex'] = df["Sex"].replace({"male": 0, "female": 1})
df.head()
```

Out[81]:

	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	373450	8.0500	S

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.

In [82]:
```
df[['First Name', 'Middle Name', 'Last Name', 'Title']] = pd.DataFrame({
    "First Name": df["Name"].str.split().str[2],
    "Middle Name": df["Name"].str.split().str[3],
    "Last Name": df["Name"].str.split().str[0].str.replace(",", ""),
    "Title": df["Name"].str.split().str[1].str.replace(".", "")
})

# The strings starting "(" is not middle name and therefore should become NaN

df["Middle Name"] = df["Middle Name"].fillna("(")
df.loc[df["Middle Name"].str.startswith("("), "Middle Name"] = float("nan")
df.head()
```

```
<ipython-input-82-550fee67d512>:5: FutureWarning: The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as literal strings when regex=True.
  "Title": df["Name"].str.split().str[1].str.replace(".", "")
```

Out[82]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	First Name	Middle Name	Last Name	Title
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	S	Owen	Harris	Braund	Mr
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	C	John	Bradley	Cumings	Mrs
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	S	Laina	NaN	Heikkinen	Miss
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	S	Jacques	Heath	Futrelle	Mrs
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	S	William	Henry	Allen	Mr

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

In [83]:
```
df['Age'] = df["Age"].fillna(df["Age"].mean())
df.head()
```

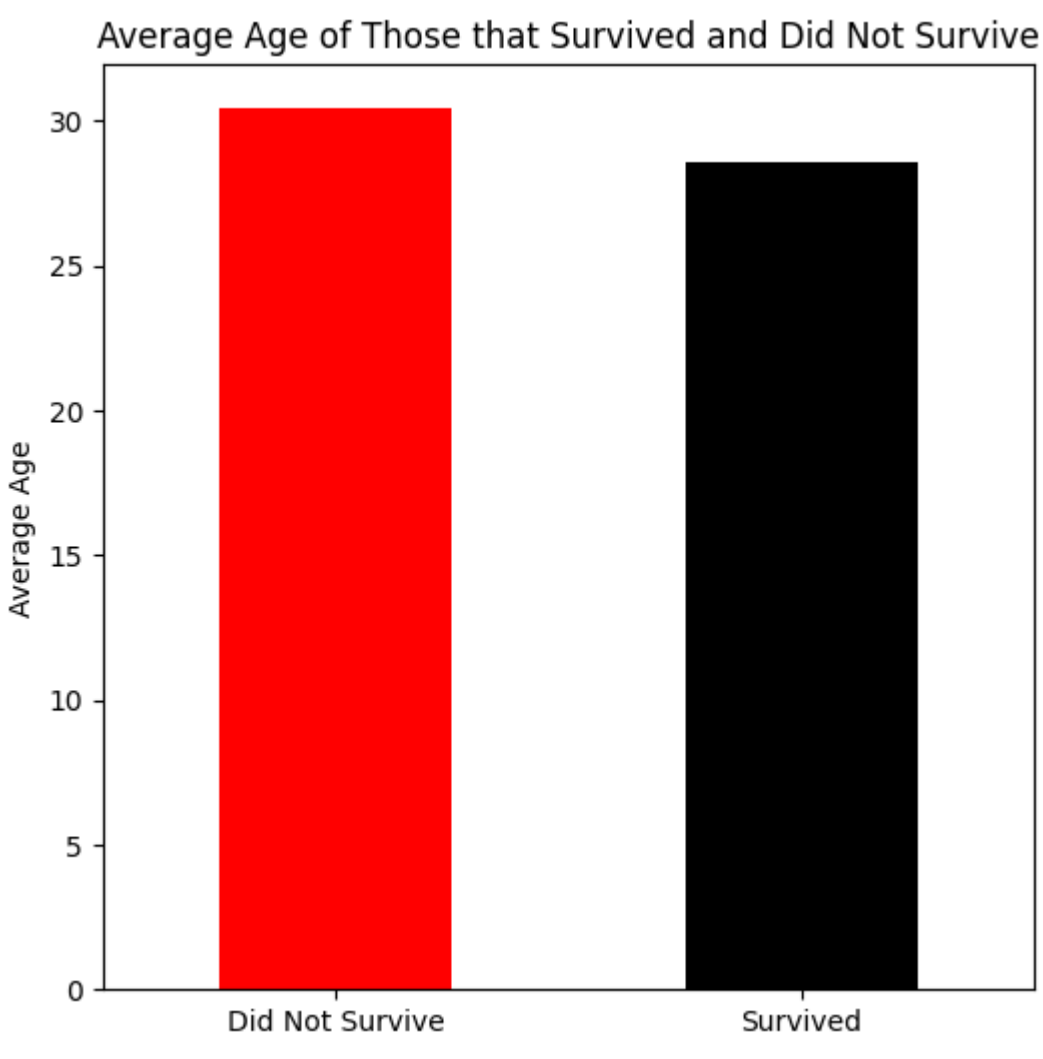
Out[83]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	First Name	Middle Name	Last Name	Title
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	S	Owen	Harris	Braund	Mr
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	C	John	Bradley	Cumings	Mrs
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	S	Laina	NaN	Heikkinen	Miss
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	S	Jacques	Heath	Futrelle	Mrs
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	S	William	Henry	Allen	Mr

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

In [84]:
```
barGraph = df.groupby("Survived")["Age"].mean().plot(kind="bar", title="Average Age Survived VS Not Survive",
                                                     ylabel="Average Age", xlabel = "", rot = 0,
                                                     color = ["red", "black"], figsize = (6,6))
barGraph.set_xticklabels(["Did Not Survive", "Survived"])
```

Out[84]:
```
[Text(0, 0, 'Did Not Survive'), Text(1, 0, 'Survived')]
```



In [85]:
```
print("The average age of the people that did not survive is higher than the average age of the people that survived.")
```

```
The average age of the people that did not survive is higher than the average age of the people that survived.
```

In [ ]:
```

```