Worksheet 02

Name: Wilbert Limson UID: U11894403

Topics

- Effective Programming

Effective Programming

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

You often cannot test run your code until everything is written

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

You may end up writing code/functions that you do not need later on

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

1. Don't Panic! 2. Read the error carefully. 3. Re-read your code - take your time. (Based on Lecture)

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 [1]:

```
import pandas as pd
```

```
df = pd.read_csv("C:/Users/Wilbert Limson/train.csv")
df.describe()
```

Out[1]:

	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 [2]:

```
print("there are " +  str( df.isnull().any(axis=1).sum()) + " rows with at least one e
```

```
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 [3]:

```
df = df.dropna(axis=1, thresh=len(df) - 200)
df.columns
```

Out[3]:

```
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 [4]:

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

Out[4]:

	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 [5]:

```
pattern = r'(?P<Last_Name>[^,]+), (?P<Title>\w+)\. (?P<First_Name>\w+)(?: (?P<Middle_N
df[['First Name', 'Middle Name', 'Last Name', 'Title']] =  df['Name'].str.extract(patt
df['Title'], df['Middle Name'] = df['Middle Name'], df['Title']
df.head()
```

Out[5]:

	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

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

In [6]:

```python
df['Age'] = df['Age'].fillna(df['Age'].mean())
df.head()
```

Out[6]:

	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

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

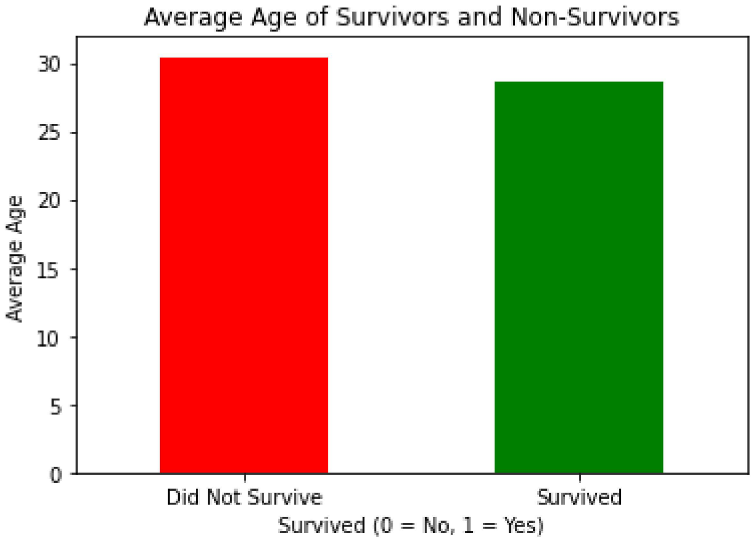
In [7]:

```python
import matplotlib.pyplot as plt

# Assuming 'df' is the DataFrame and it has been loaded with the 'Survived' and 'Age'

# Calculating the average age of those who survived and those who did not
average_ages = df.groupby('Survived')['Age'].mean()

# Plotting the bar chart
average_ages.plot(kind='bar', color=['red', 'green'])
plt.title('Average Age of Survivors and Non-Survivors')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Average Age')
plt.xticks(ticks=[0, 1], labels=['Did Not Survive', 'Survived'], rotation=0)
plt.show()
```



In [8]:

```
print("The red bar represents the average age of individuals who did not survive. The 
print("The green bar represents the average age of the survivors. Similarly, its heigh
print("The difference between the two bars. A higher bar for one group would indicate 
```

```
The red bar represents the average age of individuals who did not survive. The height
of the bar corresponds to the average age calculated for this group.
The green bar represents the average age of the survivors. Similarly, its height corr
esponds to the average age of this group.
The difference between the two bars. A higher bar for one group would indicate a high
er average age compared to the other.
```