1 Worksheet 02

Name: Jian Xie

UID: 75516303

1.0.1 Topics

- Effective Programming

1.0.2 Effective Programming

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

1. Inflexibility: Early decisions can make it difficult to incorporate chang
2. Overlooked Details: High-level planning might overlook some detail the implementation phase.
3. Delayed Testing: Testing is often deferred until significant parts of th

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

1. Lack of Overall Vision
2. Integration Challenges: Integrating individual components into a co challenging
3. Efficiency Issues: Without an overarching plan, there can be redunda

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

1. Stay organized - have a plan and prioritize
2. Have a structure: Do one thing at a time; Compartmentalize functio
3. Have a good programming method. Think about using top down or solve the 90% problem first.

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

In [ ]:
```
1
```

1.1 Exercise

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

The goal of this exercise is to practice using pandas (https://pypi.org/proje

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 shee (https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf) may come in handy.

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

In [20]:

```
1 import pandas as pd
2
3 df = pd.read_csv('train.csv')# your code here
4 df.describe()
```

	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 leas

In [3]:
```
df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 1
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
5	6	0	3	Moran, Mr. James	male	NaN	0	0	3308
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	1746
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	3499
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	3477
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	2377

In [5]:
```
num_rows_with_missing_values = df.isna().any(axis=1).sum()
print("there are " +  str(num_rows_with_missing_values) + "
```

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

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

In [21]:

```
print("origin df shape: " + str(df.shape))
cols_to_drop = df.columns[df.isna().sum() > 200]
df = df.drop(columns=cols_to_drop)
print("cleaned df shape: " + str(df.shape))
df.columns
```

```
origin df shape: (891, 12)
cleaned df shape: (891, 11)
```

```
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 [22]:

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0
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

e) Complete the code below to add four columns `First Name` , `Middle` 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 [23]:

```
def parse_name(name):
    # Split the name into last name and the rest
    last_name, rest = name.split(', ', 1)
    # Split the rest by the first space to separate title fr
    title, first_middle = rest.split('. ', 1) if '. ' in res
    # Further split first and middle names
    names_split = first_middle.split(' ')
    first_name = names_split[0] if len(names_split) > 0 else
    middle_name = ' '.join(names_split[1:]) if len(names_spl

    return pd.Series([first_name, middle_name, last_name, ti

df[['First Name', 'Middle Name', 'Last Name', 'Title']] = df
df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	0	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	0	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	0	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	1	27.0	0	2	347742
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	1	14.0	1	0	237736

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

In [24]:

```
# Calculate the average age, excluding NaN values
average_age = df['Age'].mean()

# Replace NaN values in the Age column with the average age
df['Age'].fillna(average_age, inplace=True)

df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
0	1	0	3	Braund, Mr. Owen Harris	0	22.000000	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.000000	1	0	PC 175
2	3	1	3	Heikkinen, Miss. Laina	1	26.000000	0	0	STON/ 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.000000	1	0	11380
4	5	0	3	Allen, Mr. William Henry	0	35.000000	0	0	37345
5	6	0	3	Moran, Mr. James	0	29.699118	0	0	33087
6	7	0	1	McCarthy, Mr. Timothy J	0	54.000000	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	0	2.000000	3	1	34990
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	1	27.000000	0	2	34774
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	1	14.000000	1	0	23773

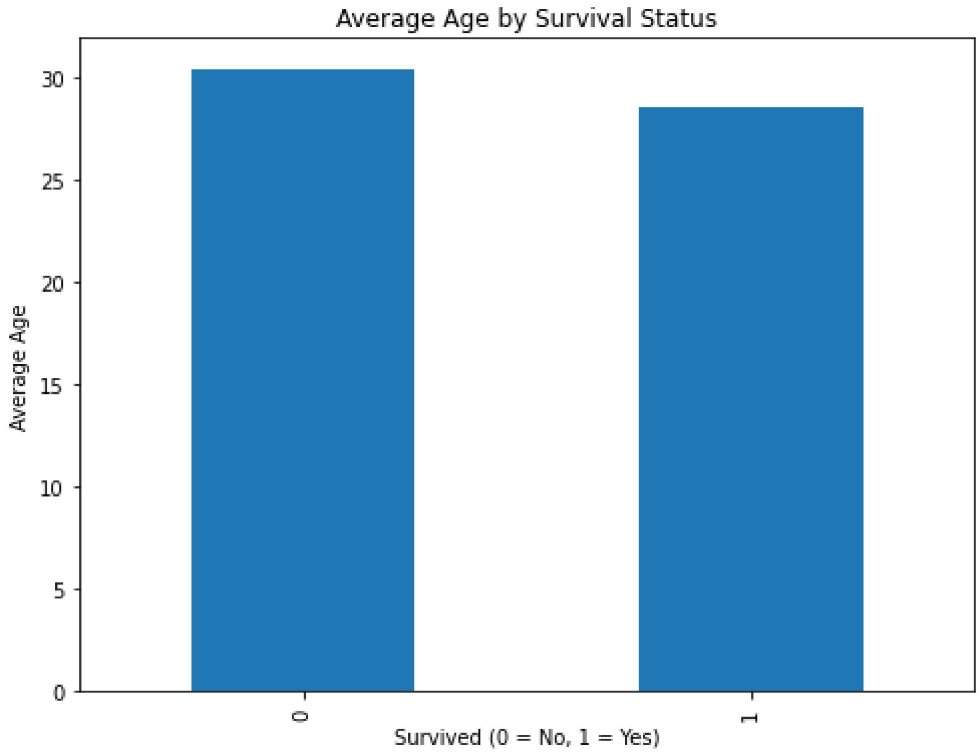
g) Plot a bar chart of the average age of those that survived and did not what you observe.

In [26]:
```
# Calculate the average age for each survival status
average_age_by_survival = df.groupby('Survived')['Age'].mean
print(average_age_by_survival)
```

```
Survived
0    30.415100
1    28.549778
Name: Age, dtype: float64
```

In [27]:
```
average_age_by_survival.plot(kind='bar', title='Average Age
```

```
<AxesSubplot:title={'center':'Average Age by Survival Status'}, xlabel='Survived (0 = No
```



Obeservations

Age Affect on Survival: Since we fill the missing ages with the average a
individuals had a bit higher suvival rate.