Question 1 [20 points]
In no more than 500 words, explain the differences between a Jupyter Notebook (.ipynb extension) and a Python file (.py extension). Focus on the advantages and disadvantages of
one or the other, particularly for data analysts/scientists. You are encouraged to use examples and screenshots of the different scripts to support your arguments.

Jupyter notebook is used to run blocks of code at a time and can be good for testing or exploratory data analysis where you don't want to run the entire code at the same time, whereas a python file is beneficial in software development to create and run an entire program. While this means that jupyter is preferential for testing purposes, a traditional python file actually works better with version control systems such as git so if you want to track the changes you are making/not risk ruining code then a traditional python file may work better.

```
In [12]: alive_counts = titanic['alive'].value_counts()

         print("Did they survive?")
         print(alive_counts)

         Did they survive?
         no     549
         yes    342
         Name: alive, dtype: int64

In [3]:  alive_counts_2 = titanic['survived'].value_counts()

         print("Did they survive?")
         print(alive_counts_2)

         Did they survive?
         0    549
         1    342
         Name: survived, dtype: int64
```

Another benefit of using jupyter notebook is the visualisations you can create in the programme and how easy it is to show this. This means you can show graphs, and text blocks, and different types of code all in the same palace and it is easily sectioned off rather than new files or new code blocks that don't appear as distinct in a traditional python file. This is especially useful as a data analyst as you are likely to be creating multiple graphs and visualisations and you may want to explain what is happening in these graphs outside of the code used to create them to give further context.

```
15      return True
16  # The patch item allows us to create fake values for the input function to co
17  class TestIsogram(unittest.TestCase):
18      @patch("builtins.input", return_value="flower")
19      def test_is_isogram_true(self, mock_input):
20          self.assertTrue(isogram())
21      # This test case checks that the function has correctly identified the is
22
23      @patch("builtins.input", return_value="burrow")
24      def test_is_isogram_false(self, mock_input):
25          self.assertFalse(isogram())
26      # This test checks that the function has identified a word as not an isog
27
28      @patch("builtins.input", return_value="burRow")
29      def test_is_isogram_case_insensitive(self, mock_input):
30          self.assertTrue(isogram())
31      # This test checks if the function is case sensitive ie. showing that the
32
33      @patch("builtins.input", return_value="123flower?")
34      def test_is_isogram_non_alpha(self, mock_input):
35          self.assertTrue(isogram())
36      # This test checks that the function only takes in letter characters and
37
38  if __name__ == "__main__":
39      unittest.main()
```

On the other hand, it may be difficult to organise your code in jupyter notebooks as the code doesn't necessarily follow a logical order. For example, if you were to forget you had not run a block of code and then later on try to run a block that refers to variables in this earlier code, it would not work and it may be hard to work out where in your notebook you had initially made this variable. Whereas, with a python file, as you are running the whole document at a time, you don't need to worry about what order to run the code in.

Overall, both pieces of software have their advantages and disadvantages. To summarise, jupyter is most useful for data analysis in terms of running small sections and completing exploratory data analysis, whereas a python file is useful for intentional design and running larger blocks of code at a time to create a programme.

Question 2 [10 points]
What is the difference between a Pandas DataFrame and a Pandas series. Show an example of how you create each of them.
A pandas series is a single column/row of data which si otherwise known as a labelled array. Then a pandas dataframe is a collection of multiple pandas series arranged in a table, similar to how a excel spreadsheet would be laid out.
This is how to create a pandas series:

```
import pandas as pd

names = pd.Series(['Karen','Sarah','Jane','Melanie'])

print(names)
```

```
/Users/lucyroberts/PycharmProjects/pythonProject/cfg-python/venv/bin/python
0      Karen
1      Sarah
2       Jane
3    Melanie
dtype: object

Process finished with exit code 0
```

This is how to create a pandas dataframe:

```
import pandas as pd

employees = {
   'Name' : ['Karen','Sarah','Jane','Melanie'],
   'Job' : ['Software Developer','Accountant','Data Analyst','Software
Developer'],
   'Age' : [38, 32, 22, 45]
}

df = pd.DataFrame(employees)
print(df)
```

```
/Users/lucyroberts/PycharmProjects/pythonProject/cfg-python/venv/bin/python
      Name                Job  Age
0    Karen  Software Developer   38
1    Sarah          Accountant   32
2     Jane        Data Analyst   22
3  Melanie  Software Developer   45

Process finished with exit code 0
```

Question 3 [10 points]
Starting from the argument that a Pandas DataFrame represents rectangular data, use the
internet and other resources to describe in no more than a few sentences the difference between rectangular and non-rectangular data.
Rectangular data refers to data that is organised in a tabular format such as a spreadsheet and all rows within the table have the same format. Whereas non rectangular data does not fit inside a simple table and each set of data will have different requirements.

Rectangular example:

restaurant_name	head_chef	special_dish	city
Fratellos	Gina Graham	Prawn Linguine	London
Lily's fried chicken	Lily Statham	Fried chicken burger with mozarella dippers	Newcastle
Manchester Tandoori	Guy Kane	Seafood Jalfrezi	Manchester

Non rectangular example:

'Dish name' : 'satay curry', 'Allergen' : 'Peanut', 'Recommended by' : 'Gordan Ramsay',
'Dish name' : 'thai red curry',
'Dish name' : 'veggie curry', 'Recommended by' : 'Nigella lawson',
'Dish name': 'seafood risotto', 'Allergen' : 'Shellfish, 'Chefs note': 'mussells have been replaced by squid'

Question 4 [10 points]
Starting from the data visualisation usage from Session 2, give examples of when figures could be:
a. Of use to the data scientist to identify patterns in the data/highlight the important parts of a data set;
The below bar chart shows how we filtered the data down to see which directors had made the most films in this dataset easily. If this was not filtered down into the top 10 and sorted into

descending order then it would be harder to see the pattern/how different directors compared against each other. We can more easily see who has the most films and the important directors who have made the most films rather than indie directors who have just had one big hit.

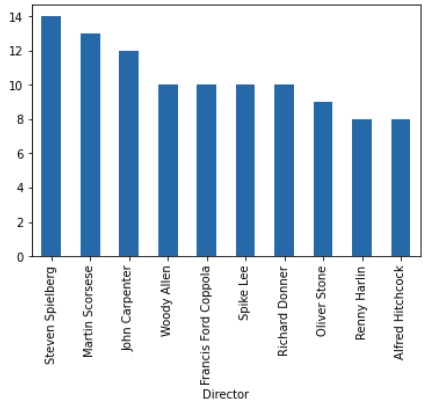
Lets plot a bar graph of top 10 movie directors

In [ ]:

```
# Step 1: Find top 10 movie directors
top_10_directors = movies_df.groupby('Director')['Title'].count().sort_values(ascending=False).head(10)

# Step 2: Plot them in graph
top_10_directors.plot(kind='bar')
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x23d577dd490>



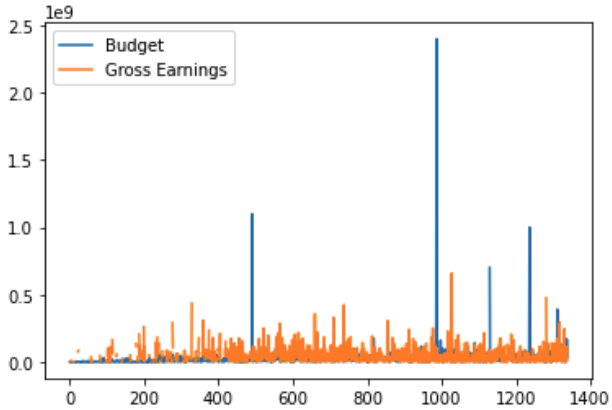
b. Tell a story and create business presentations.

Figures can be used to more clearly illustrate trends and patterns in data rather than just showing numbers. For example, the below graph could be used by a director to explain the point of how higher budget films will often generate higher sales, and the size of these returns are shown.

In [ ]:

```
movies_df[['Budget', 'Gross Earnings']].plot.line()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x23d57770100>



Question 5 [50 points]
Each sub-question is worth 10 points.
Using the titanic dataset which you can read into your notebook using the following code,
import pandas as pd
titanic = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv')
answer the following questions:
a. How many columns and rows does the data have?

In [1]:
```
import pandas as pd

titanic = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/titanic.csv')

num_rows, num_columns = titanic.shape

print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_columns}")
```

```
Number of rows: 891
Number of columns: 15
```

There are 891 rows and 15 columns

b. Get a sense of your data and find the min, max, and count/mean depending on the data type.

In [2]:
```
print(titanic.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

In [4]:
```
min_age = titanic['age'].min()
max_age = titanic['age'].max()
mean_age = titanic['age'].mean()

print(f"Mean Age: {mean_age:.2f}")
print(f"Minimum Age: {min_age}")
print(f"Maximum Age: {max_age}")
```

```
Mean Age: 29.70
Minimum Age: 0.42
Maximum Age: 80.0
```

In [5]:
```
gender_counts = titanic['sex'].value_counts()

print("Gender Counts:")
print(gender_counts)
```

```
Gender Counts:
male      577
female    314
Name: sex, dtype: int64
```

In [25]:
```
class_counts = titanic['pclass'].value_counts().rename('Total Count')

deaths_by_class = titanic[titanic['survived'] == 0].groupby('pclass').size().rename('Dead Count')

result_df = pd.concat([class_counts, deaths_by_class], axis=1, sort=False)

result_df = result_df.sort_index()

print(result_df)
```

```
   Total Count  Dead Count
1          216          80
2          184          97
3          491         372
```

In [12]:
```
alive_counts = titanic['alive'].value_counts()

print("Did they survive?")
print(alive_counts)
```

```
Did they survive?
no     549
yes    342
Name: alive, dtype: int64
```

In [13]:
```
embark_counts = titanic['embark_town'].value_counts()

print("Where did they embark?")
print(embark_counts)
```

```
Where did they embark?
Southampton    644
Cherbourg      168
Queenstown      77
Name: embark_town, dtype: int64
```

In [16]:
```
adult_age = 18

children = titanic[titanic['age'] < adult_age].shape[0]
dead_children = titanic[(titanic['age'] < adult_age) & (titanic['survived'] == 0)].shape[0]

print(f"Number of children on the titanic, under {adult_age} years old: {children}")
print(f"Number of children, under {adult_age} years old who died: {dead_children}")
```

```
Number of children on the titanic, under 18 years old: 113
Number of children, under 18 years old who died: 52
```

In [24]:
```

```

```
   Total Count  Dead Count
1          216          80
2          184          97
3          491         372
```

In [26]:
```
gender_counts = titanic['sex'].value_counts().rename('Total Count')

deaths_by_gender = titanic[titanic['survived'] == 0].groupby('sex').size().rename('Dead Count')

result_df = pd.concat([gender_counts, deaths_by_gender], axis=1, sort=False)

result_df = result_df.sort_index()

print(result_df)
```

```
        Total Count  Dead Count
female          314          81
male            577         468
```

c. Give an overview (code and an explanation) of all missing values in the data.

In [28]:
```
missing_data = titanic.isnull().sum()

print("Missing Data:")
print(missing_data)
```

```
Missing Data:
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

There are 177 records missing the passengers age, 688 missing the deck letter, and 2 missing the mebarkment town. It is likely that the deck number was not remembered by people and as the boat had gone down they could not check what their deck is which is why this number is so high. It is possible that the age may be missing for a lot of people as they were unwilling to share their age. The embarked location data that is missing I am unsure of the origin, it may be that these people were stowaways and so there was no record of where they got on to the boat but there was when they got off the boat as there are examples of these with passengers who survived the trip but not of those who did not survive the trip.

```
dead_data = titanic[titanic['survived'] == 0]
alive_data = titanic[titanic['survived'] == 1]

missing_dead = dead_data.isnull().sum()
missing_alive = alive_data.isnull().sum()

print("Missing Data for Those Who Died:")
print(missing_dead)

print("\nMissing Data for Those Who Are Alive:")
print(missing_alive)
```

```
Missing Data for Those Who Died:
survived         0
pclass           0
sex              0
age            125
sibsp            0
parch            0
fare             0
embarked         0
class            0
who              0
adult_male       0
deck           482
embark_town      0
alive            0
alone            0
dtype: int64

Missing Data for Those Who Are Alive:
survived         0
pclass           0
sex              0
age             52
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           206
embark_town      2
alive            0
alone            0
dtype: int64
```

d. Delete the rows where you do not have information about the age of the person. Then group the passengers in a 10 year age range (for example, you can do something like 0 – 10, 11 – 20, 21 – 30, etc).

```
data_without_null_age = titanic.dropna(subset=['age'])

data_without_null_age = data_without_null_age.copy()

age_bins = range(0, 101, 10)
age_labels = [f"{start}-{end}" for start, end in zip(age_bins[:-1], age_bins[1:])]

data_without_null_age.loc[:, 'age_range'] = pd.cut(data_without_null_age['age'], bins=age_bins, labels=age_labels, right=False)

passengers_by_age_range = data_without_null_age.groupby('age_range').size()

print("Passengers grouped by 10-year age range:")
print(passengers_by_age_range)
```

In [37]:
```
data_without_null_age = titanic.dropna(subset=['age'])

data_without_null_age = data_without_null_age.copy()

age_bins = range(0, 101, 10)
age_labels = [f"{start}-{end}" for start, end in zip(age_bins[:-1], age_bins[1:])]

data_without_null_age.loc[:, 'age_range'] = pd.cut(data_without_null_age['age'], bins=age_bins, labels=age_labels, r

passengers_by_age_range = data_without_null_age.groupby('age_range').size()

print("Passengers grouped by 10-year age range:")
print(passengers_by_age_range)
```

```
Passengers grouped by 10-year age range:
age_range
0-10       62
10-20     102
20-30     220
30-40     167
40-50      89
50-60      48
60-70      19
70-80       6
80-90       1
90-100      0
dtype: int64
```

e. For each age category created in d), find out how many passengers are female/male, and how many travelled in each class.

In [38]:
```
passengers_by_sex = data_without_null_age.groupby('sex').size()

passengers_by_class = data_without_null_age.groupby('pclass').size()

print("Number of passengers by sex:")
print(passengers_by_sex)

print("\nNumber of passengers by class:")
print(passengers_by_class)
```

```
Number of passengers by sex:
sex
female    261
male      453
dtype: int64

Number of passengers by class:
pclass
1    186
2    173
3    355
dtype: int64
```