Project log

Project: Roller coaster

————————————————————

- Import all the relevant data science/analysis libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
plt.style.use('ggplot')
pd.set_option('display.max_columns', 200)
```

- Import my data (data frame as df)

Understanding my data

- I did some basic data understanding

~ Shape command (tells me the number of rows/columns in my dataset)
~ Head command (shows me some rows) (note: I used the display max columns option in line 6 so that I can see all the columns in one go)
~ List out all the columns using the columns command (I like to visually see all the columns I am working with)

- Use dot dtypes to Identify the Dtype. Every column in a series and every pandas series has a type
- A lot of the columns were objects (string type columns), some are float values (float64(decimals)), some are integers (int64(whole numbers))

- Describe function (gives me a info and statistics about numeric data in my dataset

Preparing my data

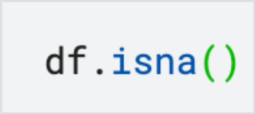
- Dropping columns and rows that irrelevant
- I found that some values had a string and a numeric version so some can be deleted
- I removed some columns by commenting them out
- Now I have a subset
- I reassigned my data frame to be this new subset and closed it with the copy command so that python knows its not just a reference to the

old data frame but an actual brand new data frame

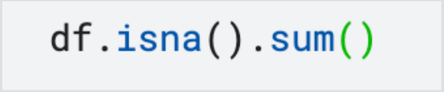
- 'opening_Date_clean' column's Dtype is object, I want it to be date time column because it contains dates. So I changed it to a date time Dtype

- Renaming my columns (Removing any spaces in the titles, using capitals where needed) and reassign the data set so it can include the new titles of columns

- Identifying NULL values (used df.isna() and then .sum() to get a summary which is easier and quicker with regards to viewing)

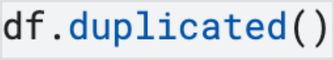


```
df.isna()
```

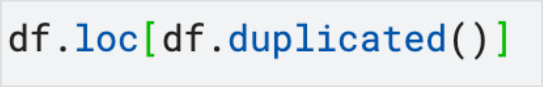


```
df.isna().sum()
```

- Checking for duplications in our rows, are there any 2 rows that are identically the same? (I used duplicated and then I used loc (locate) to locate where the duplications were)

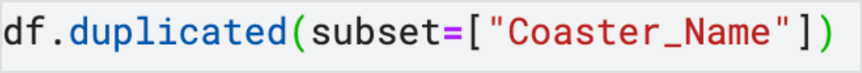


```
df.duplicated()
```

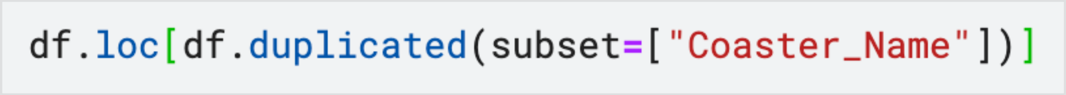


```
df.loc[df.duplicated()]
```

- Checked for duplications in specific columns e.g. in the Coaster_Name column - there was some
- Located the duplications

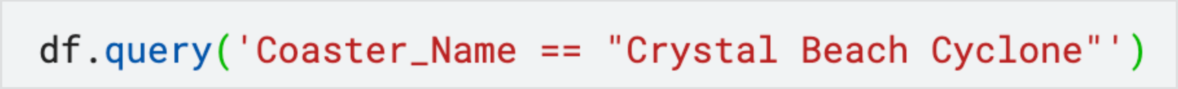


```
df.duplicated(subset=["Coaster_Name"])
```



```
df.loc[df.duplicated(subset=["Coaster_Name"])]
```

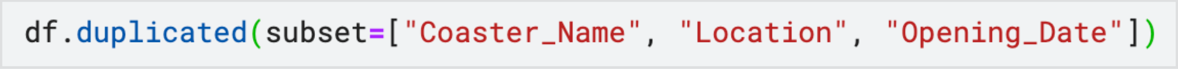
- Find out why there is a duplication, using the query command, I looked into one of the coaster names so I can analyse the values to find out why!



```
df.query('Coaster_Name == "Crystal Beach Cyclone"')
```

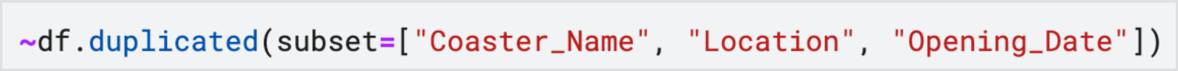
- I found that the reason why there are 2 roller coasters with the same name (crystal beach cyclone) is because the ride was first introduced in the year 1926 and then in 1927. This could be because the ride was opened, closed, and then reopened again.

- I only want the record of when it was first introduced. What I will do is remove any rows where a certain amount of columns are the same (in our case we are finding duplicated rows where the coaster name, location and opening date are the same)



```
df.duplicated(subset=["Coaster_Name", "Location", "Opening_Date"])
```

- Now I want to take the inverse of this, so just the columns that are not duplicates OR just the first columns of the duplicates. I use a tilde for this.



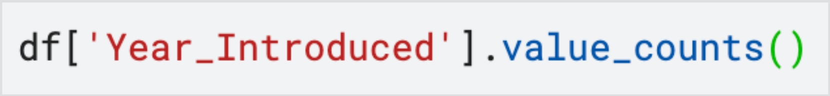
```
~df.duplicated(subset=["Coaster_Name", "Location", "Opening_Date"])
```

- Locate these duplicates, and save this new subset that will now only includes the rows where just the rows that are not duplicates are included and/or rows that don’t have coaster_name, location and opening date as identical.

Understanding the features

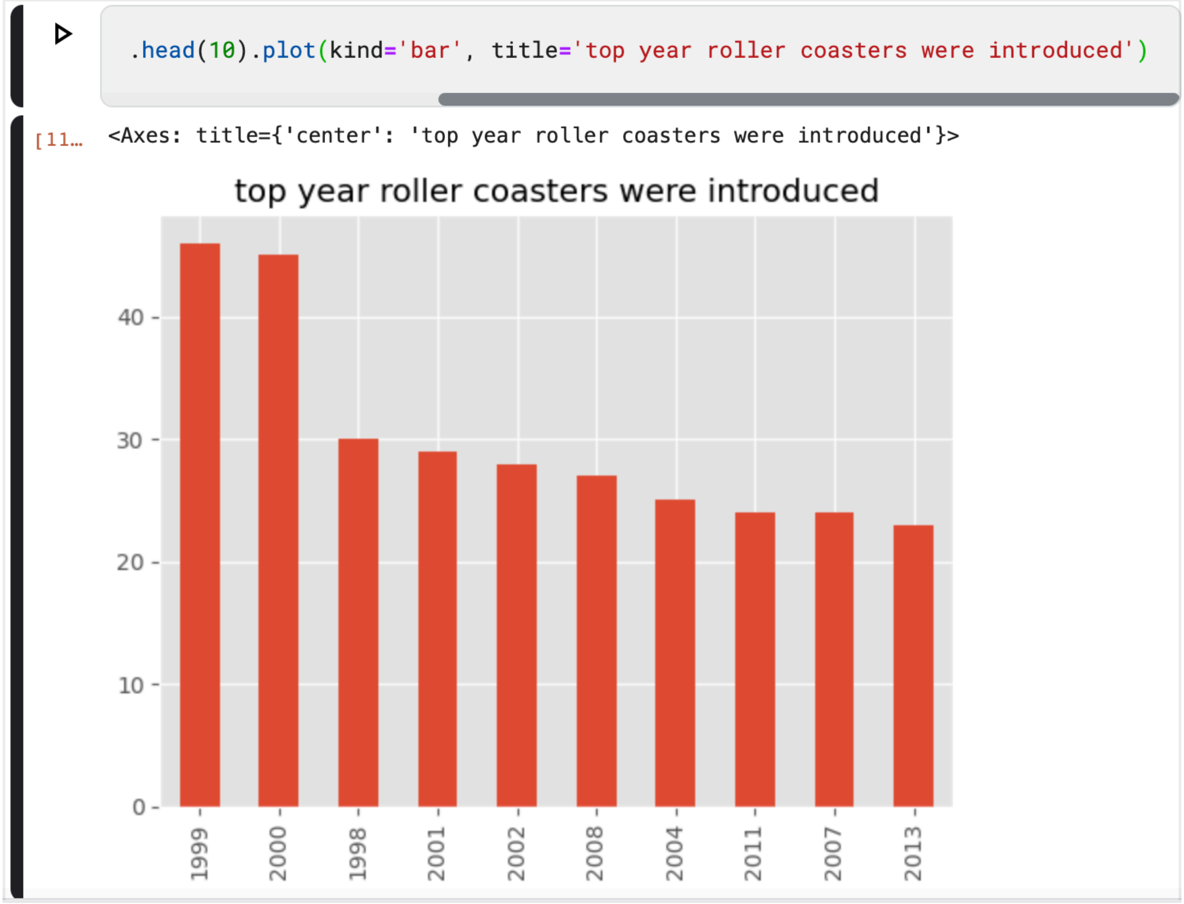
(Univariate analysis)

- **Running value counts on a single column (year_introduced (year rides were introduced/opened)). Gives us an idea of what year had the most roller coasters introduced and what years were the least.**



```
df['Year_Introduced'].value_counts()
```

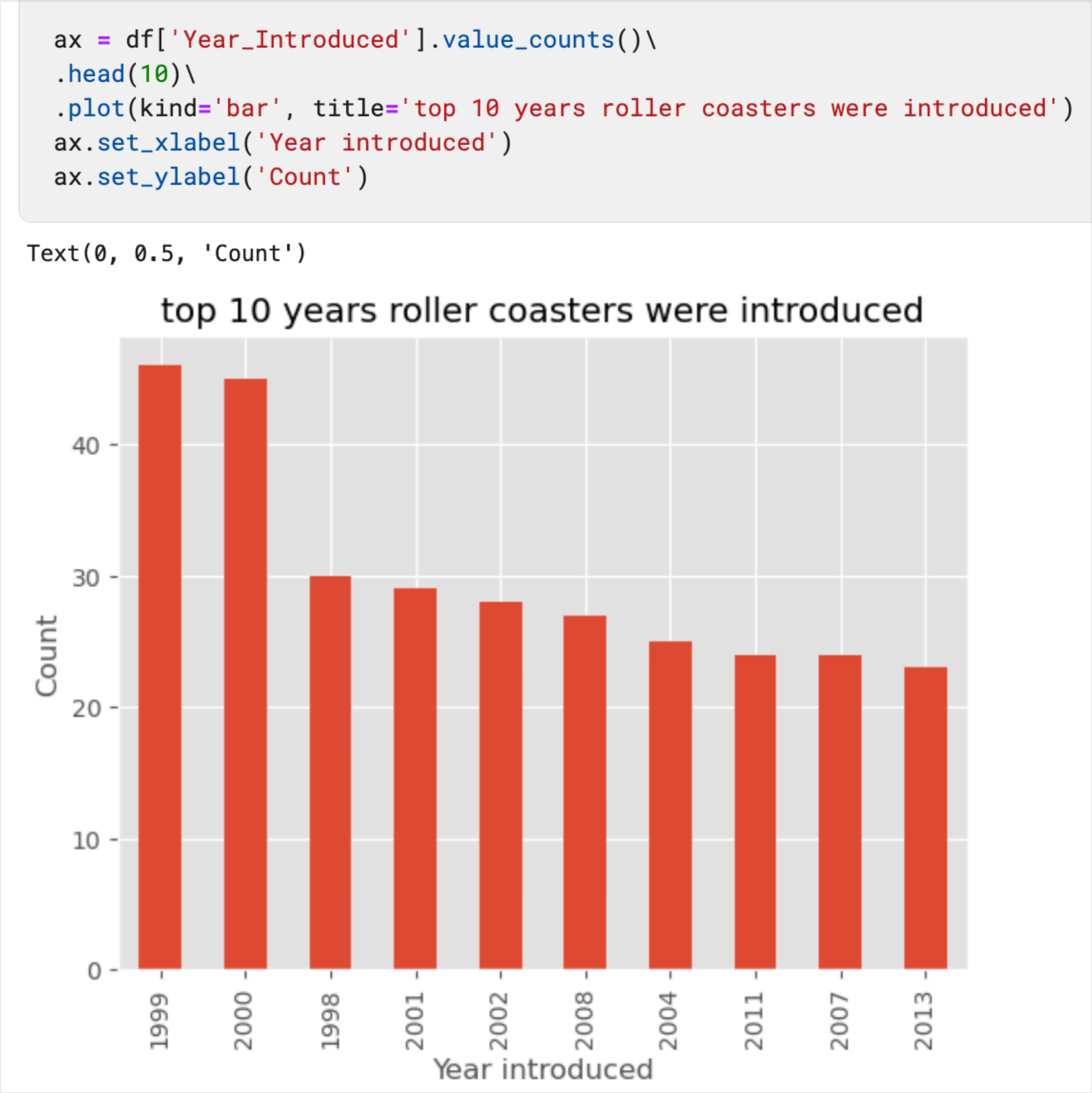
- Now I want to see the top 10 years where most roller coasters were introduced and I made a plot to go with it (bar chart with title)



```
.head(10).plot(kind='bar', title='top year roller coasters were introduced')
```

```
[11… <Axes: title={'center': 'top year roller coasters were introduced'}>
```

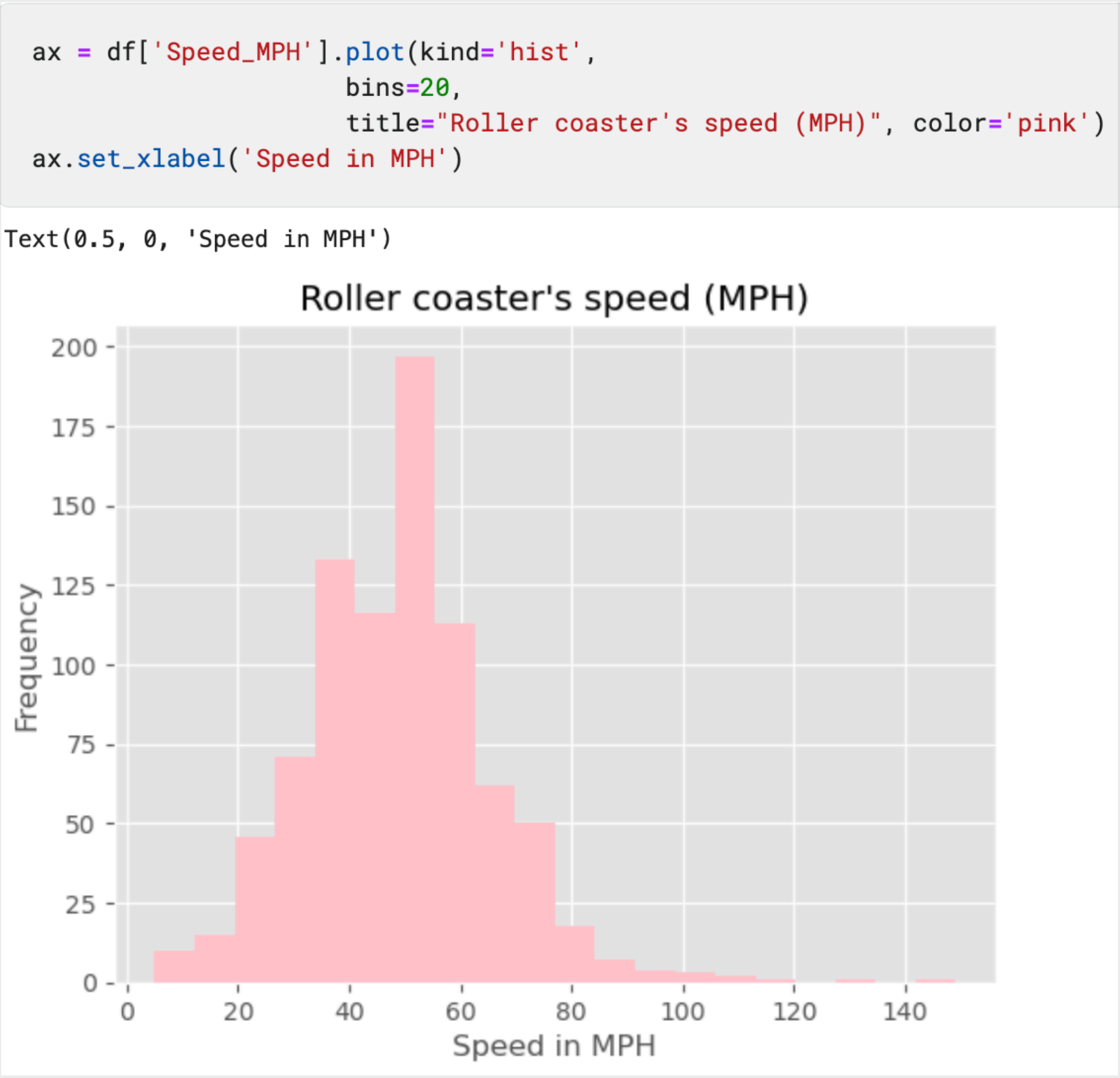
- Save it as a matplotlib axis
- And with the axis I added a few more info to it.



```
ax = df['Year_Introduced'].value_counts()\
.head(10)\
.plot(kind='bar', title='top 10 years roller coasters were introduced')
ax.set_xlabel('Year introduced')
ax.set_ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```

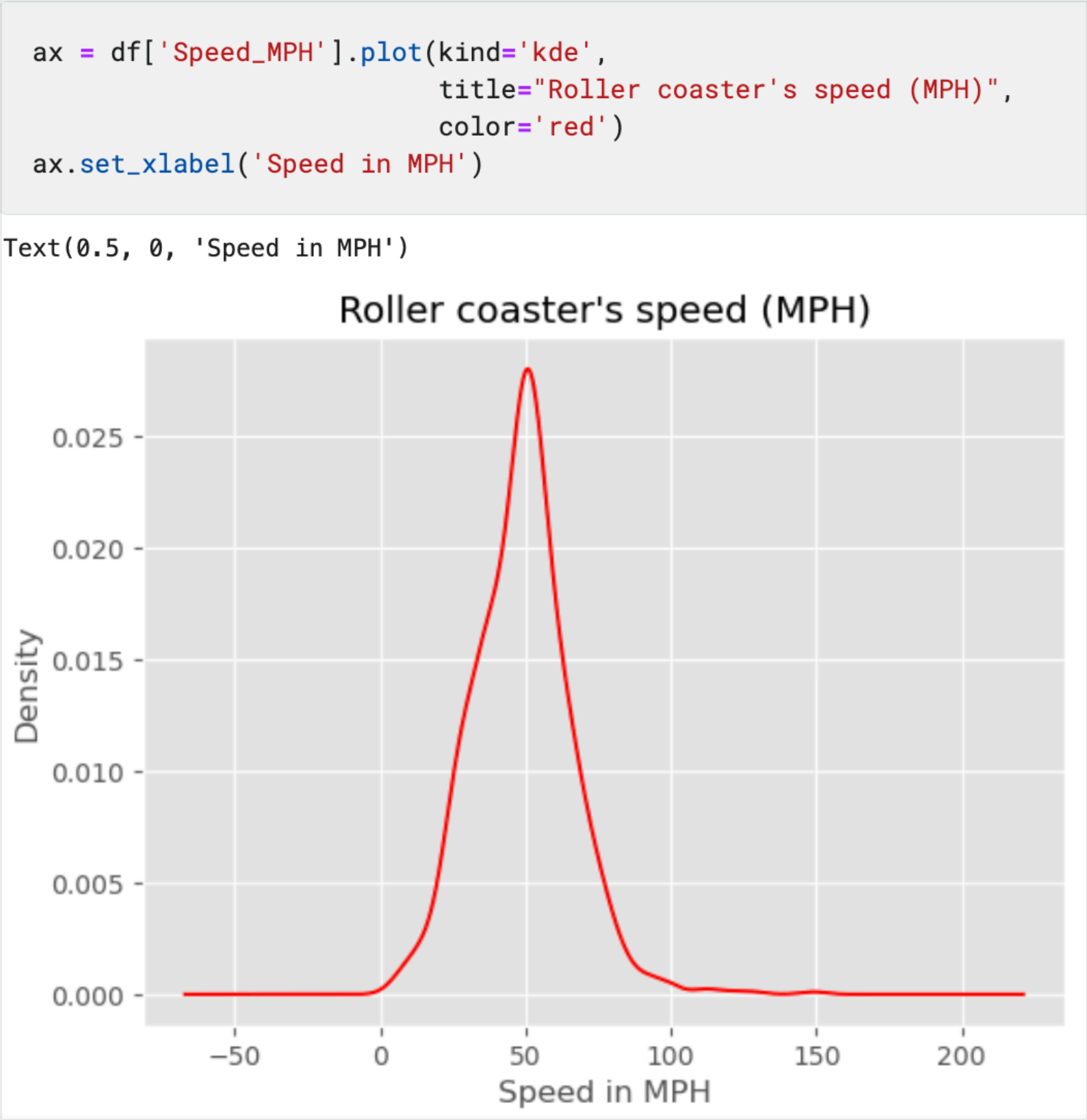
- Made a plot for speed using a histogram to show me the count of MPH. This shows me the distribution of the speed.



```
ax = df['Speed_MPH'].plot(kind='hist',
                          bins=20,
                          title="Roller coaster's speed (MPH)", color='pink')
ax.set_xlabel('Speed in MPH')
```

```
Text(0.5, 0, 'Speed in MPH')
```

- I noticed the most common speed to be around 50 MPH
- I used the same code with a Kernel density plot (KDE)



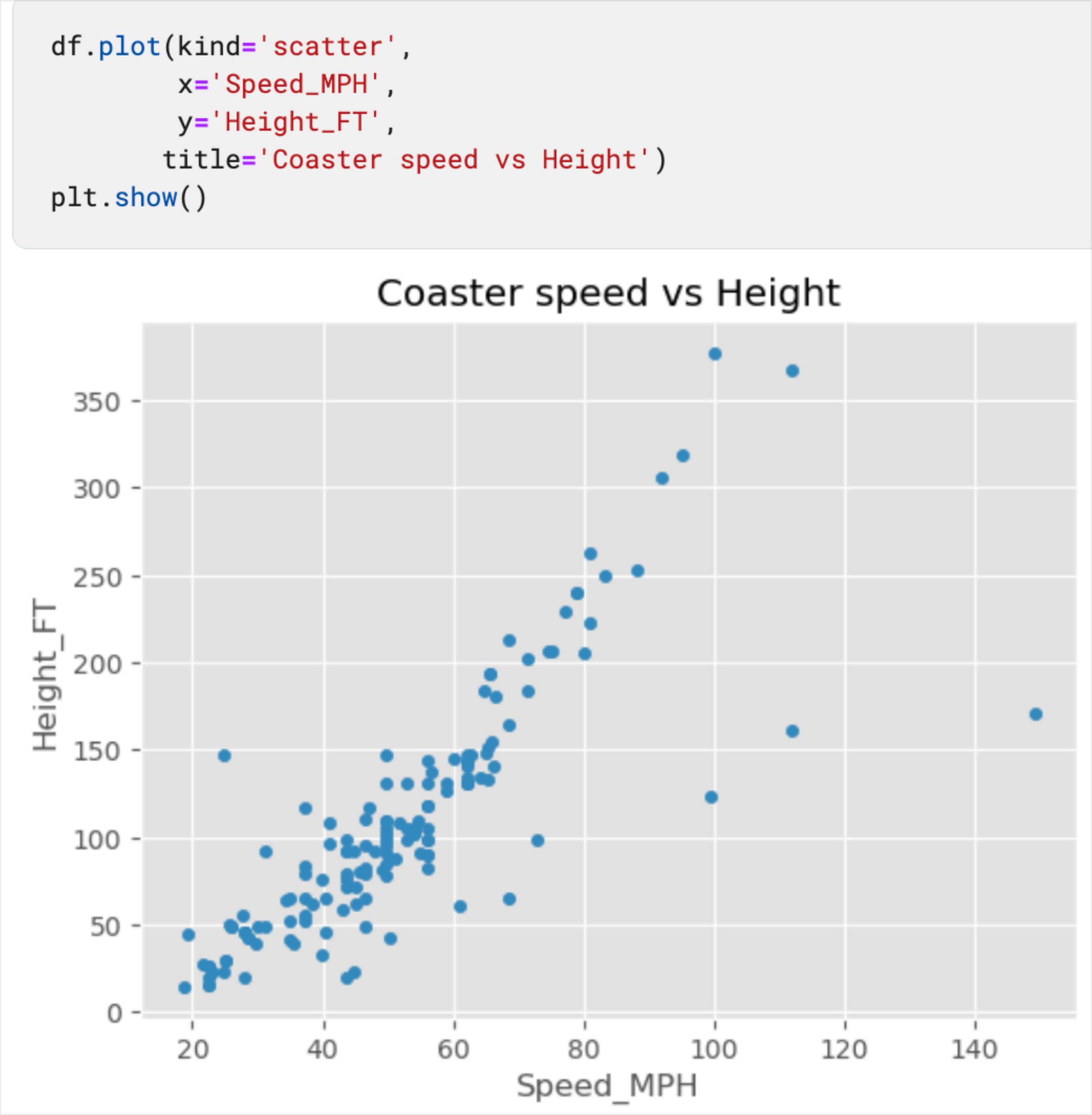
```
ax = df['Speed_MPH'].plot(kind='kde',
                          title="Roller coaster's speed (MPH)",
                          color='red')
ax.set_xlabel('Speed in MPH')
```

```
Text(0.5, 0, 'Speed in MPH')
```

Feature relationships

Now I want to look at how do the different features in my dataset relate to each other!

- I used a scatter diagram to show the relation between the speed of the coaster and the height of the coaster.

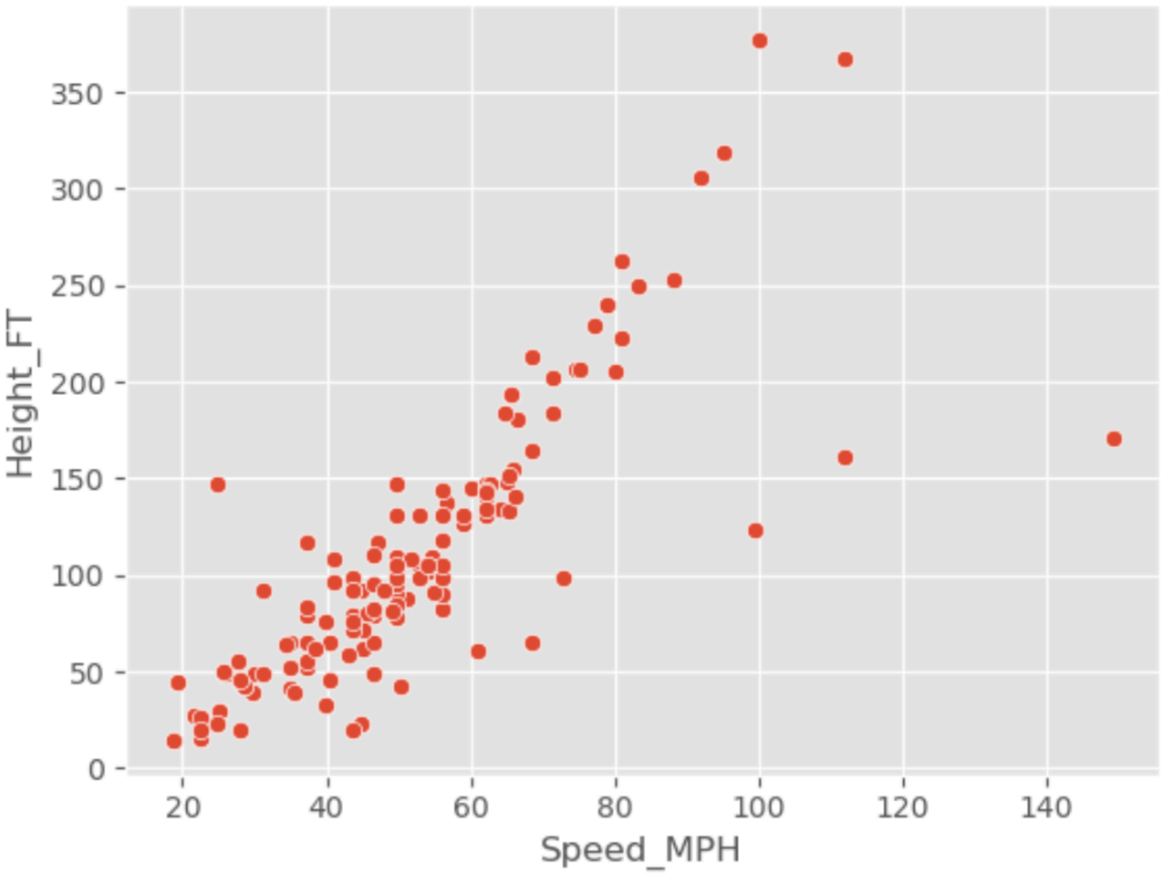


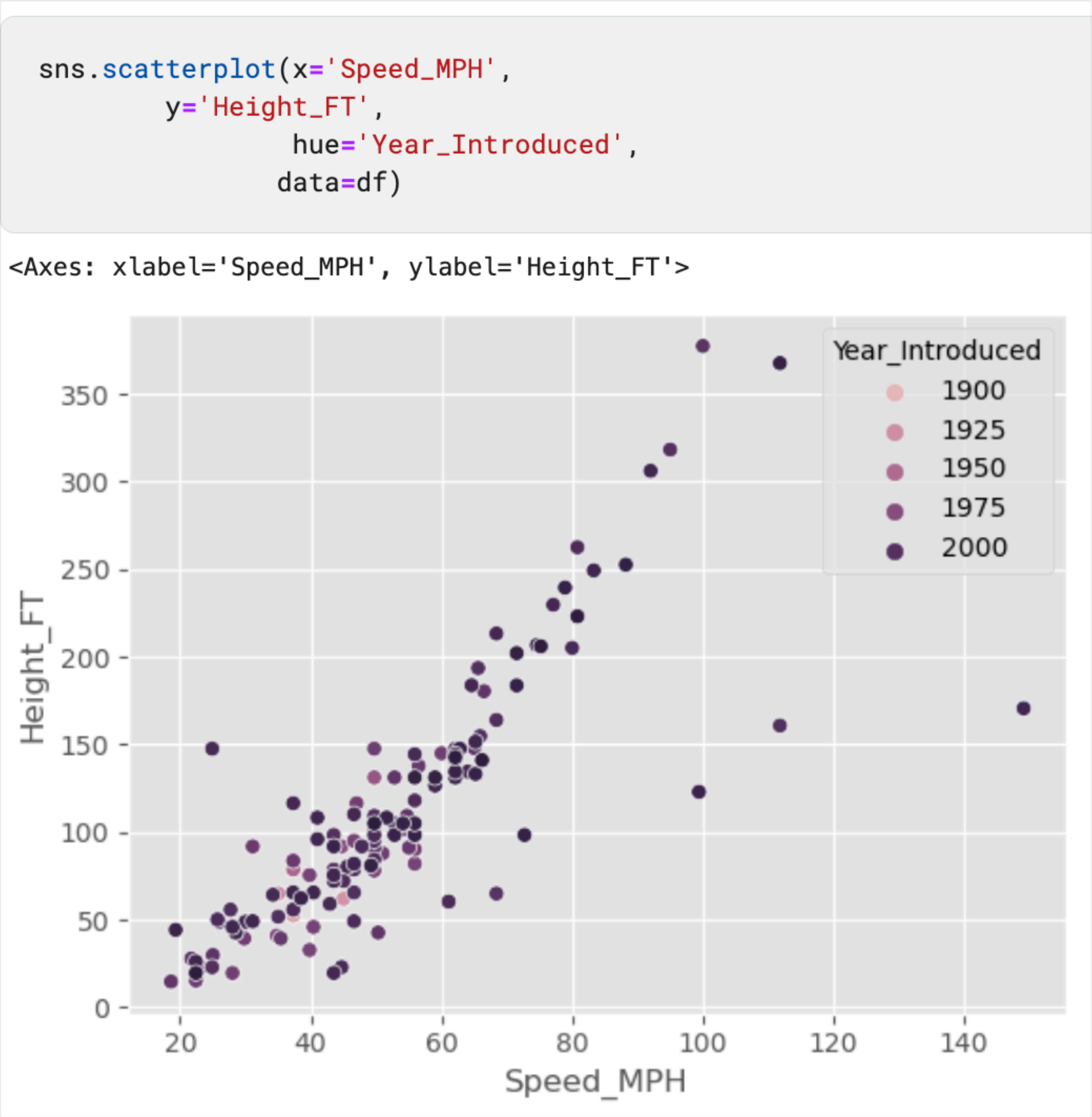
```
df.plot(kind='scatter',
        x='Speed_MPH',
        y='Height_FT',
       title='Coaster speed vs Height')
plt.show()
```

- Made the same scatter diagram using seaborn
- With this I was able to add colour to my scatter diagram based on another variable (Year_Introduced)

```
sns.scatterplot(x='Speed_MPH',
         y='Height_FT',
                data=df)
```

```
<Axes: xlabel='Speed_MPH', ylabel='Height_FT'>
```

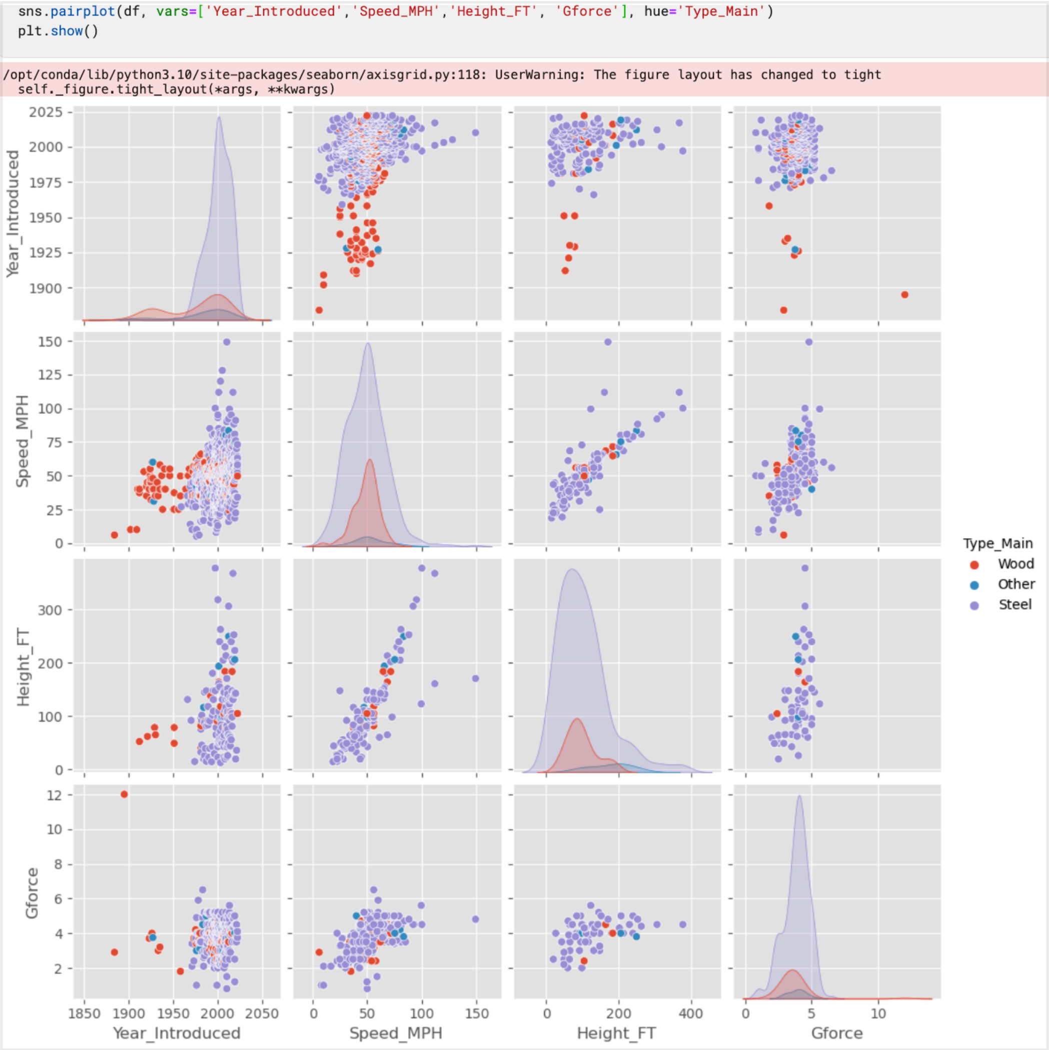




```
sns.scatterplot(x='Speed_MPH',
        y='Height_FT',
                hue='Year_Introduced',
               data=df)
```

```
<Axes: xlabel='Speed_MPH', ylabel='Height_FT'>
```

- I used seaborns feature, pairplot, so I can compare multiple features with each other.

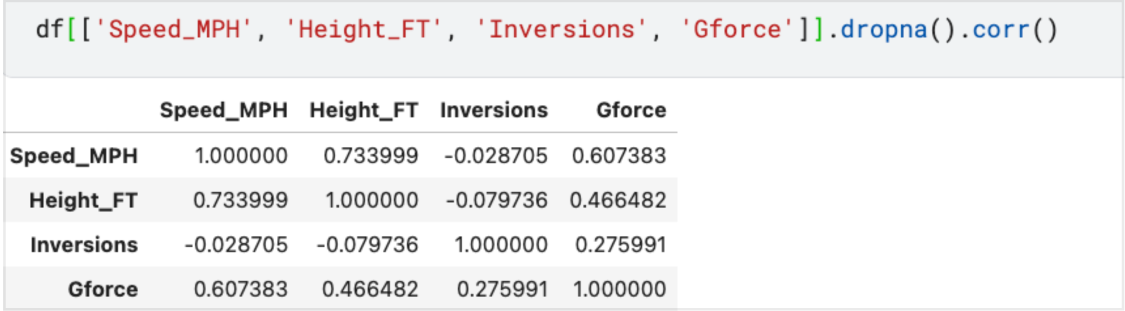


```
sns.pairplot(df, vars=['Year_Introduced','Speed_MPH','Height_FT', 'Gforce'], hue='Type_Main')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

- I was able to compare various variable with one another and learn new things:
- As the years went by, the coasters were being made faster and manufacturers moved away from using wooden materials to steel. Quite naturally, the G force rates increased over time as the rides being made were getting faster and faster. Overall it suggests that as technology was advancing, rides were advancing with it.

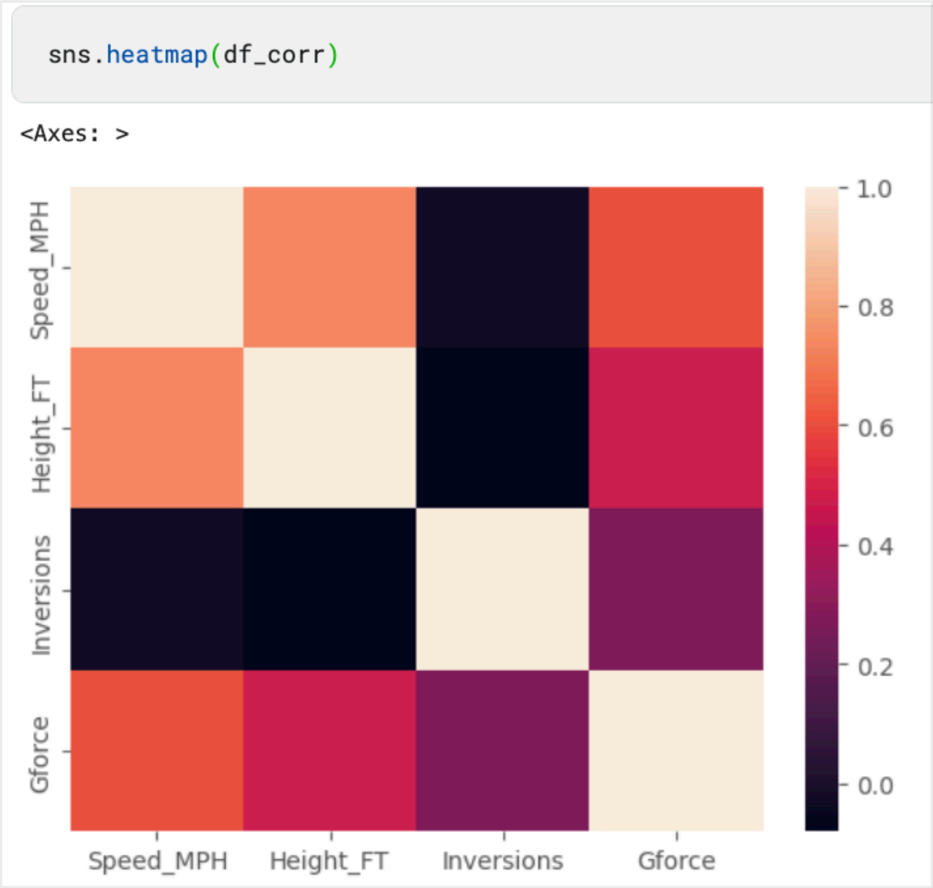
- Run the correlation function on a few of the numeric variables to see their correlation



```
df[['Speed_MPH', 'Height_FT', 'Inversions', 'Gforce']].dropna().corr()
```

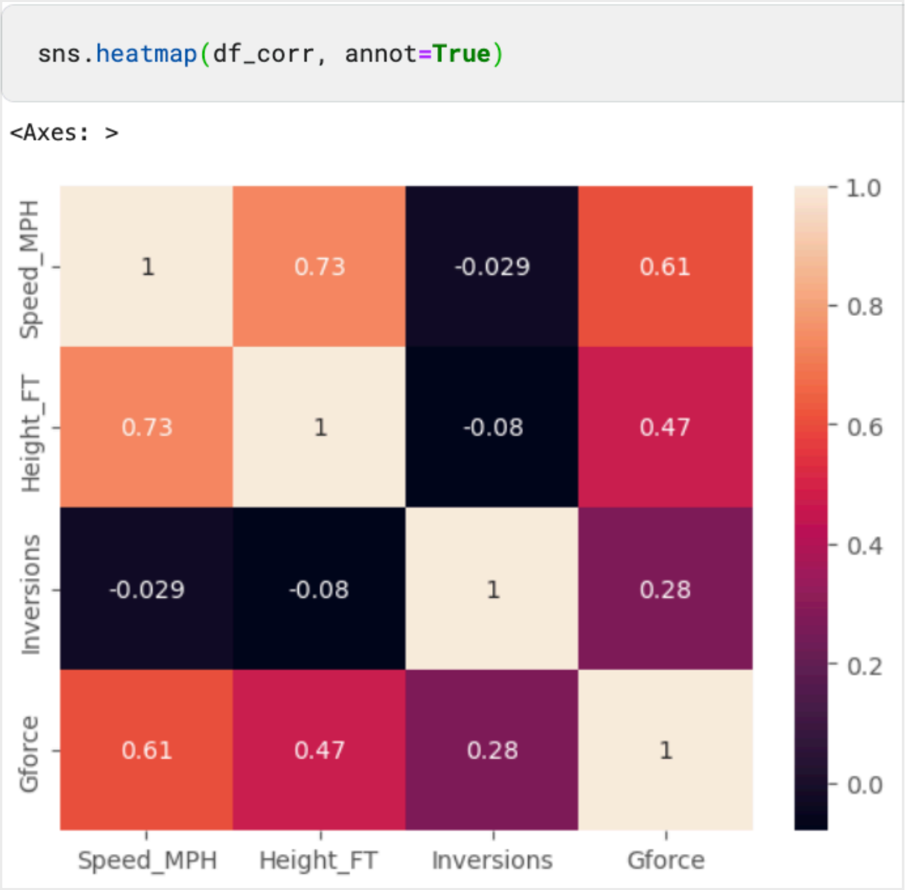
	Speed_MPH	Height_FT	Inversions	Gforce
Speed_MPH	1.000000	0.733999	-0.028705	0.607383
Height_FT	0.733999	1.000000	-0.079736	0.466482
Inversions	-0.028705	-0.079736	1.000000	0.275991
Gforce	0.607383	0.466482	0.275991	1.000000

- I used seaborns heat map feature and used the df correlation inside it.



```
sns.heatmap(df_corr)
```

```
<Axes: >
```



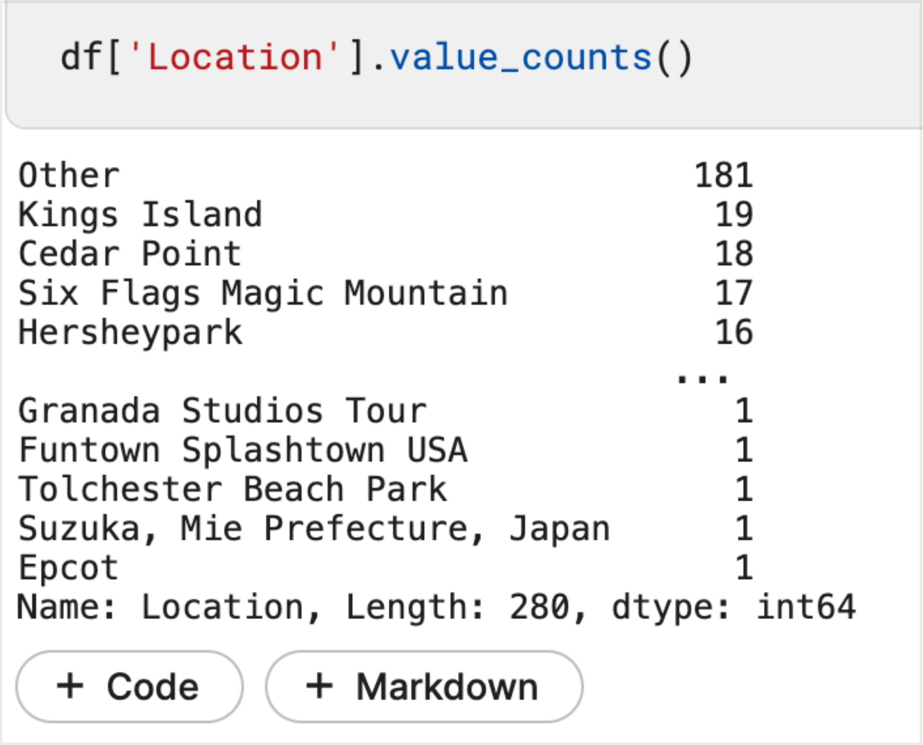
```
sns.heatmap(df_corr, annot=True)
```

```
<Axes: >
```

Ask a question about the data

A question: what are the locations with the fastest roller coasters(with a minimum of 10 roller coasters at that location)?

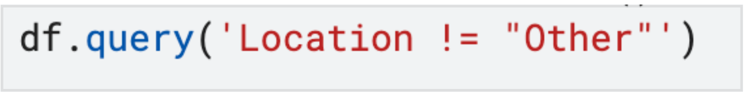
- First I did a count of how many rides there was at each location



```
df['Location'].value_counts()
```

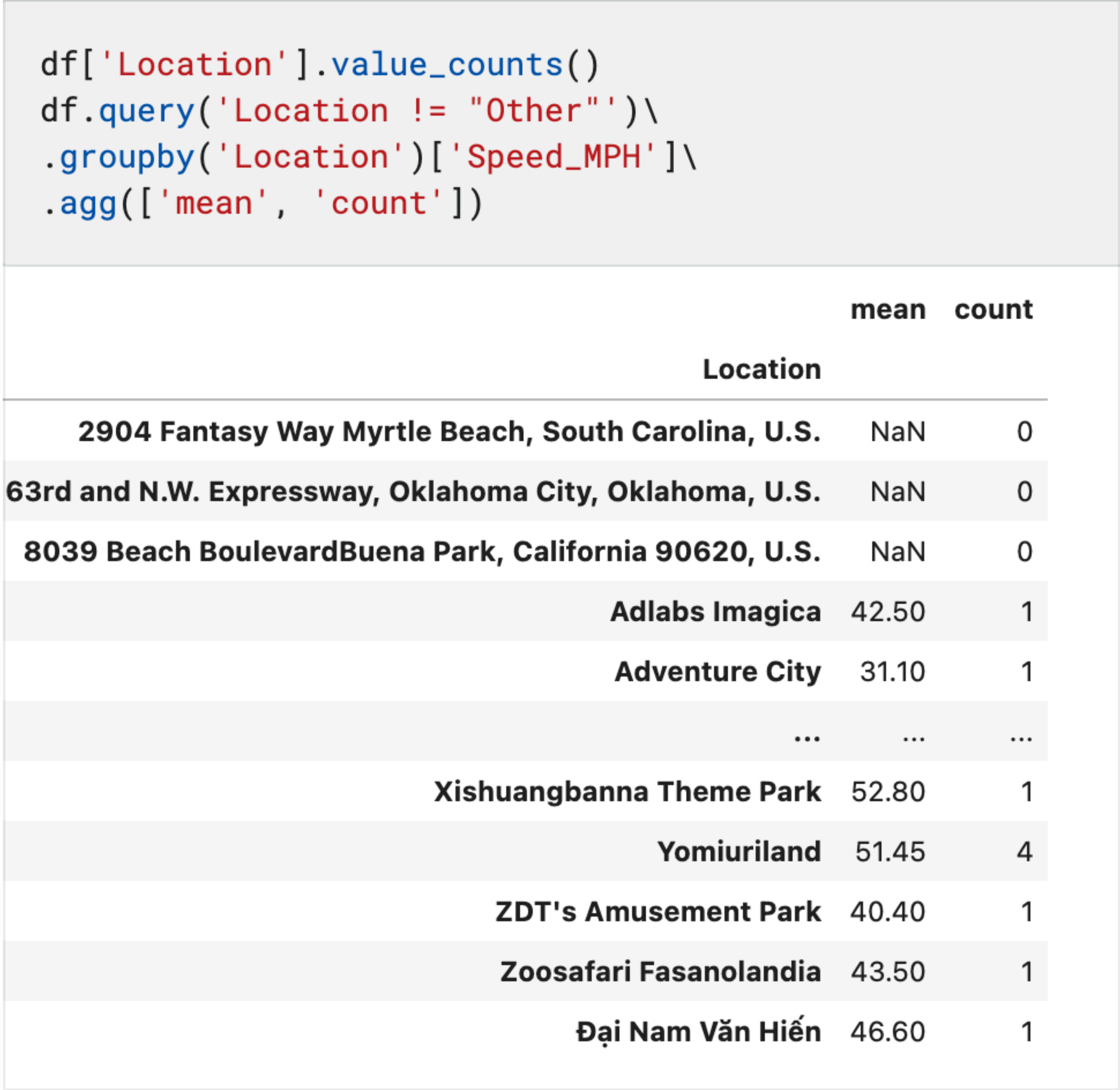
```
Other                              181
Kings Island                        19
Cedar Point                         18
Six Flags Magic Mountain            17
Hersheypark                         16
                                  ...
Granada Studios Tour                 1
Funtown Splashtown USA               1
Tolchester Beach Park                1
Suzuka, Mie Prefecture, Japan        1
Epcot                                1
Name: Location, Length: 280, dtype: int64
```

- There is 'other' which I did not need for this analysis so I queried it out so it only shows me those rides where location does not equal to 'other'.



```
df.query('Location != "Other"')
```

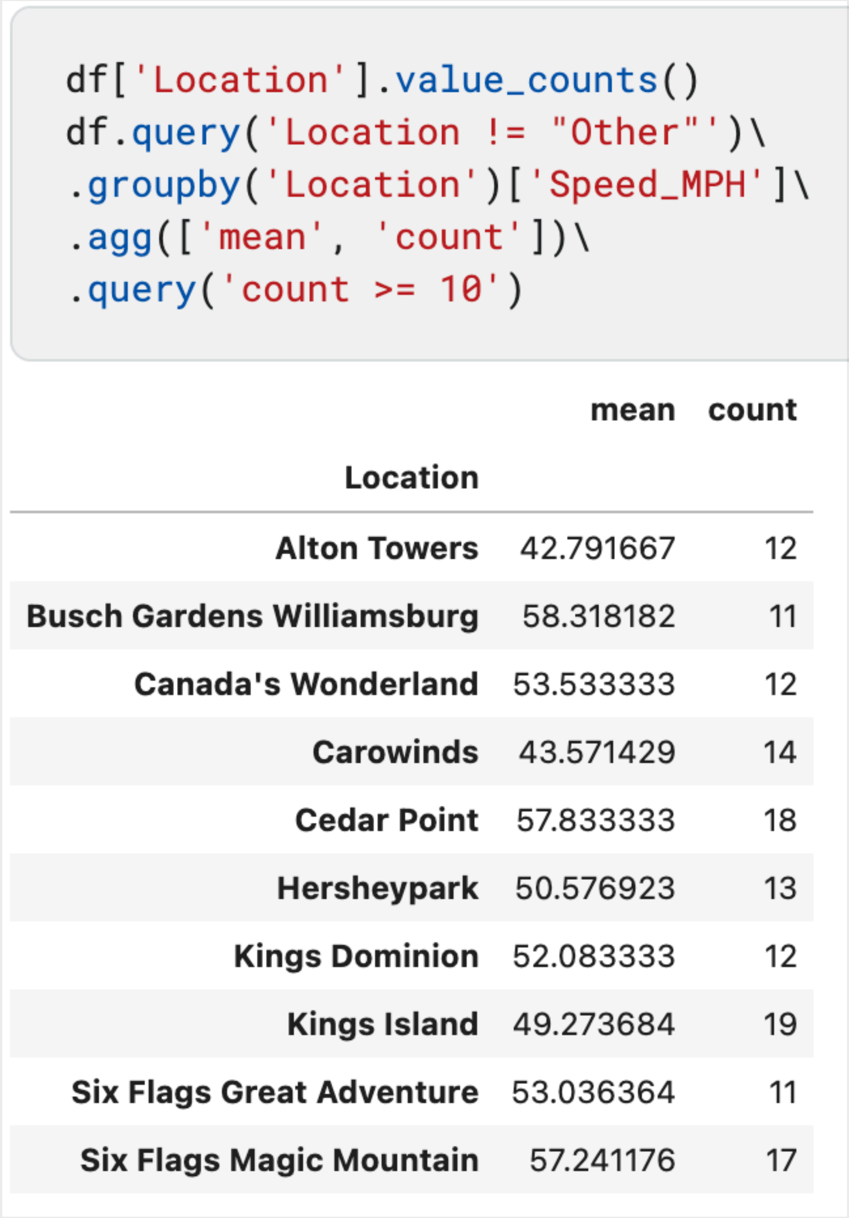
- And then I grouped by these locations and look at the speed in mph column
- Find out a few things about this 'per location' like what the average speed is, and what are the number of coasters in that location.



```
df['Location'].value_counts()
df.query('Location != "Other"')\
.groupby('Location')['Speed_MPH']\
.agg(['mean', 'count'])
```

	mean	**count**
Location		
2904 Fantasy Way Myrtle Beach, South Carolina, U.S.	NaN	0
63rd and N.W. Expressway, Oklahoma City, Oklahoma, U.S.	NaN	0
8039 Beach BoulevardBuena Park, California 90620, U.S.	NaN	0
Adlabs Imagica	42.50	1
Adventure City	31.10	1
...	...	...
Xishuangbanna Theme Park	52.80	1
Yomiuriland	51.45	4
ZDT's Amusement Park	40.40	1
Zoosafari Fasanolandia	43.50	1
Đại Nam Văn Hiến	46.60	1

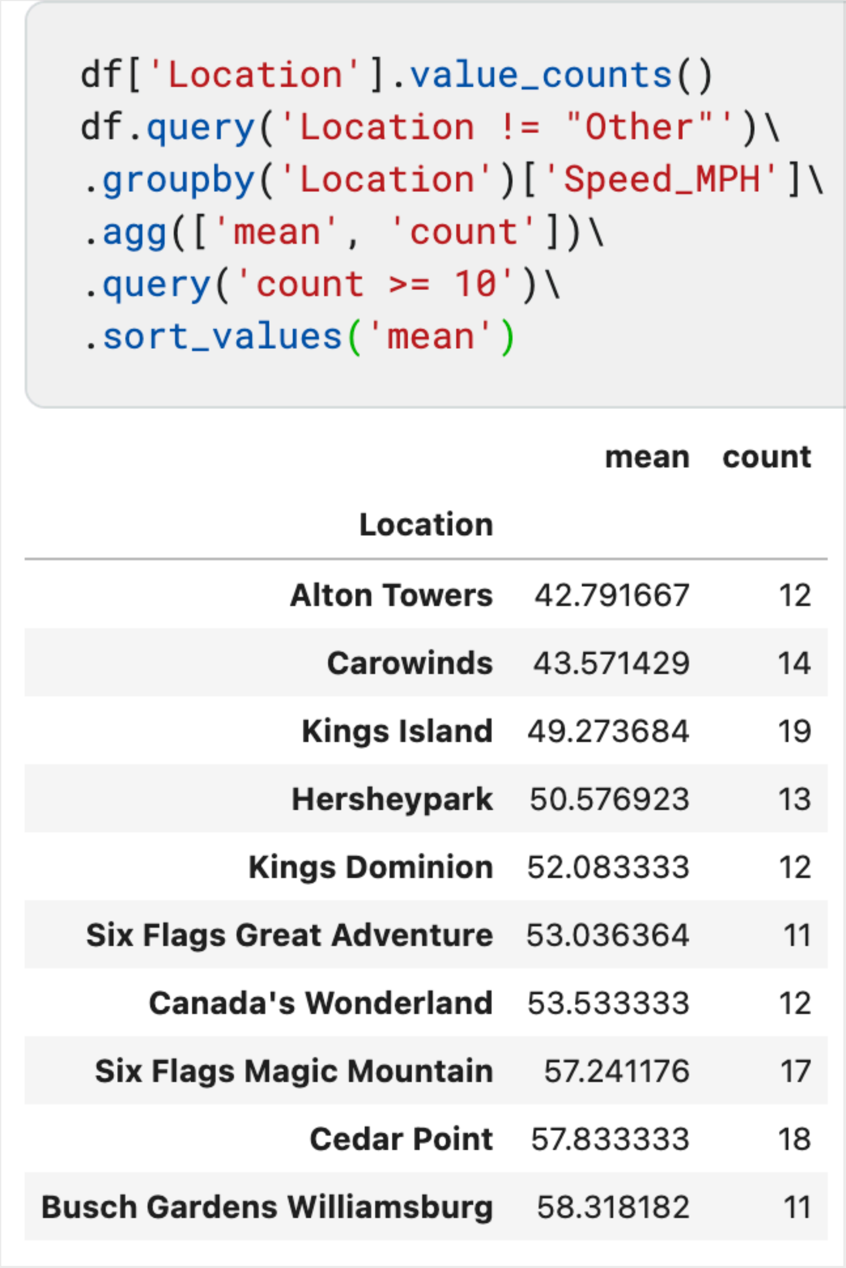
- e.g. we see that in location: Yomiuriland, there are 4 rides and the average speed is 51.45MPH
- Next I will query where the count is greater than or equal to 10 (minimum of 10 rides for each location showed)



```
df['Location'].value_counts()
df.query('Location != "Other"')\
.groupby('Location')['Speed_MPH']\
.agg(['mean', 'count'])\
.query('count >= 10')
```

	mean	count
Location		
Alton Towers	42.791667	12
Busch Gardens Williamsburg	58.318182	11
Canada's Wonderland	53.533333	12
Carowinds	43.571429	14
Cedar Point	57.833333	18
Hersheypark	50.576923	13
Kings Dominion	52.083333	12
Kings Island	49.273684	19
Six Flags Great Adventure	53.036364	11
Six Flags Magic Mountain	57.241176	17

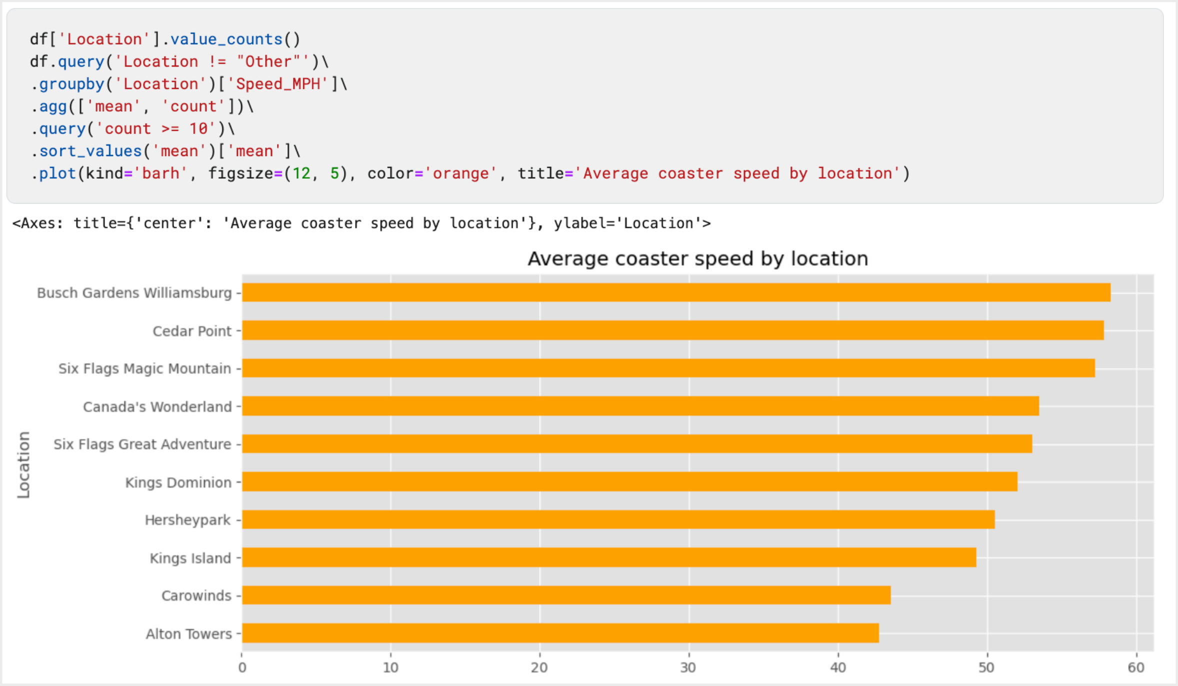
- Now I can see the locations with 10 or more rides and their average speeds. So in kings island, there are 19 rides with an average speed of 49.27MPH
- To make it cleaner, I sorted it by average speed (slowest to fastest).



```
df['Location'].value_counts()
df.query('Location != "Other"')\
.groupby('Location')['Speed_MPH']\
.agg(['mean', 'count'])\
.query('count >= 10')\
.sort_values('mean')
```

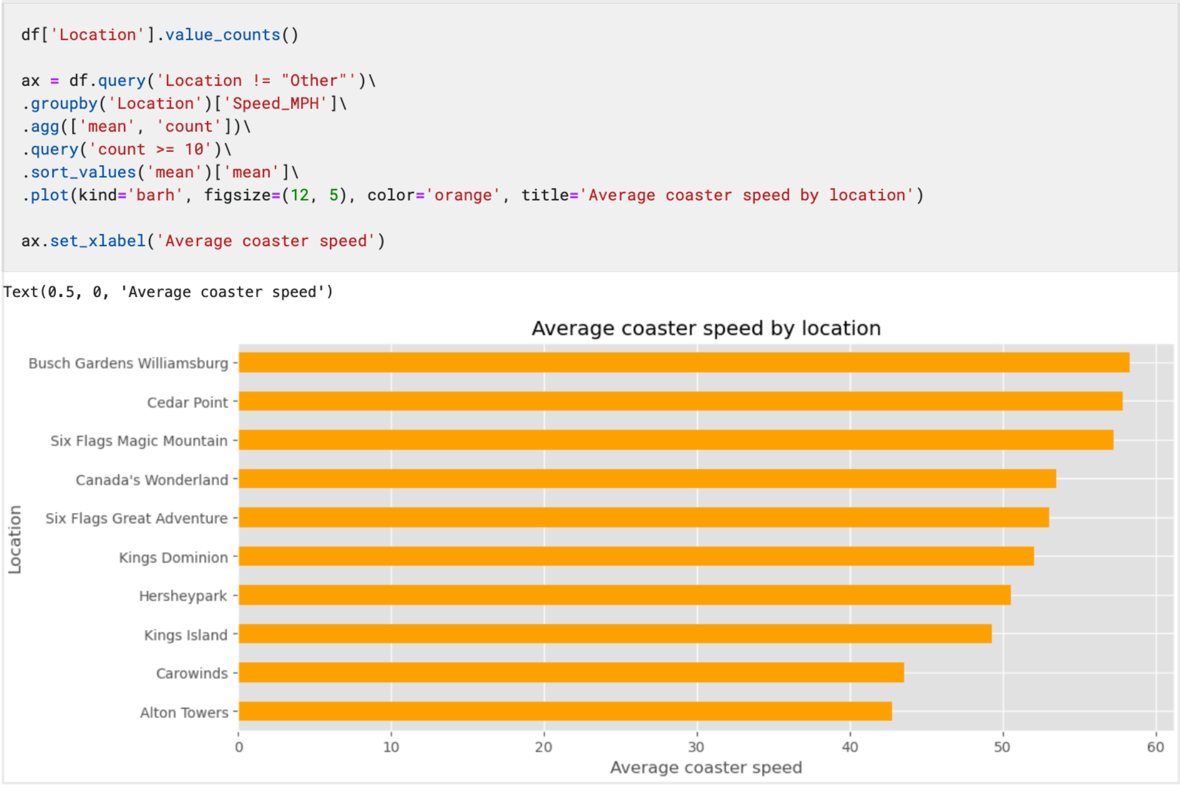
	mean	count
Location		
Alton Towers	42.791667	12
Carowinds	43.571429	14
Kings Island	49.273684	19
Hersheypark	50.576923	13
Kings Dominion	52.083333	12
Six Flags Great Adventure	53.036364	11
Canada's Wonderland	53.533333	12
Six Flags Magic Mountain	57.241176	17
Cedar Point	57.833333	18
Busch Gardens Williamsburg	58.318182	11

- I made a horizontal bar chart plot for the above:



```
df['Location'].value_counts()
df.query('Location != "Other"')\
.groupby('Location')['Speed_MPH']\
.agg(['mean', 'count'])\
.query('count >= 10')\
.sort_values('mean')['mean']\
.plot(kind='barh', figsize=(12, 5), color='orange', title='Average coaster speed by location')
```

```
<Axes: title={'center': 'Average coaster speed by location'}, ylabel='Location'>
```



```
df['Location'].value_counts()

ax = df.query('Location != "Other"')\
.groupby('Location')['Speed_MPH']\
.agg(['mean', 'count'])\
.query('count >= 10')\
.sort_values('mean')['mean']\
.plot(kind='barh', figsize=(12, 5), color='orange', title='Average coaster speed by location')

ax.set_xlabel('Average coaster speed')
```

```
Text(0.5, 0, 'Average coaster speed')
```