Part 2: Real Life Example of a Python/Pandas Data Analysis Project

- Demonstration of a real life data analysis project using Python, Pandas, SQL and Seaborn

```
In [4]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inLine
```

```
In [7]:
import os
```

```
In [9]:
# Ensure that you have the correct working directory.
# The file containing the data should be located within your current working directory.

#os.getcwd() -> This provides your current working directory.
#os.chdir('C:\\Users\\maullonv\\Desktop\\dataAnalysisPython')
    #-> Use this to change your working directory if needed.
```

```
In [13]:
# Load Data

# read the csv file into Python

sales = pd.read_csv(
    'sales_data.csv',
    parse_dates=['Date'])
```

In [14]:

```python
# The data at a glance:
sales.head()
```

	Date	Day	Month	Year	Customer_Age	Age_Group	Customer_Gender	Country	State	Product_Category	Sub_(
0	2013-11-26	26	November	2013	19	Youth (<25)	M	Canada	British Columbia	Accessories	Bike Ra
1	2015-11-26	26	November	2015	19	Youth (<25)	M	Canada	British Columbia	Accessories	Bike Ra
2	2014-03-23	23	March	2014	49	Adults (35-64)	M	Australia	New South Wales	Accessories	Bike Ra
3	2016-03-23	23	March	2016	49	Adults (35-64)	M	Australia	New South Wales	Accessories	Bike Ra
4	2014-05-15	15	May	2014	47	Adults (35-64)	F	Australia	New South Wales	Accessories	Bike Ra

sales.head()

- This is the first few lines of the data frame created
- A data frame is pretty much a csv representation, however has more things incorporated within it
    - Ex: each column has a certain data type
    - It’s a better format to use when conducting our analysis

```
In [15]: sales.shape

(113036, 18)

In [16]: sales.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113036 entries, 0 to 113035
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Date              113036 non-null  datetime64[ns]
 1   Day               113036 non-null  int64
 2   Month             113036 non-null  object
 3   Year              113036 non-null  int64
 4   Customer_Age      113036 non-null  int64
 5   Age_Group         113036 non-null  object
 6   Customer_Gender   113036 non-null  object
 7   Country           113036 non-null  object
 8   State             113036 non-null  object
 9   Product_Category  113036 non-null  object
 10  Sub_Category      113036 non-null  object
 11  Product           113036 non-null  object
 12  Order_Quantity    113036 non-null  int64
 13  Unit_Cost         113036 non-null  int64
 14  Unit_Price        113036 non-null  int64
 15  Profit            113036 non-null  int64
 16  Cost              113036 non-null  int64
 17  Revenue           113036 non-null  int64
dtypes: datetime64[ns](1), int64(9), object(8)
memory usage: 12.1+ MB
```

sales.shape()

- This gives (#rows, #columns) within the data

sales.info()

- After we load the data, we want to find out more about it
- In reference to the shape and other properties in the data we are working with
- Ex. this code / function displays the columns and its properties
    - Ex. Customer_Age column has object type integer / int
    - Have an idea about the size of the data
- This provides a better idea of the structure of the data

In [17]:

```
sales.describe()
```

	Day	Year	Customer_Age	Order_Quantity	Unit_Cost	Unit_Price	Profit	Cost
count	113036.000000	113036.000000	113036.000000	113036.000000	113036.000000	113036.000000	113036.000000	113036.000000
mean	15.665753	2014.401739	35.919212	11.901660	267.296366	452.938427	285.051665	469.318695
std	8.781567	1.272510	11.021936	9.561857	549.835483	922.071219	453.887443	884.866118
min	1.000000	2011.000000	17.000000	1.000000	1.000000	2.000000	-30.000000	1.000000
25%	8.000000	2013.000000	28.000000	2.000000	2.000000	5.000000	29.000000	28.000000
50%	16.000000	2014.000000	35.000000	10.000000	9.000000	24.000000	101.000000	108.000000
75%	23.000000	2016.000000	43.000000	20.000000	42.000000	70.000000	358.000000	432.000000
max	31.000000	2016.000000	87.000000	32.000000	2171.000000	3578.000000	15096.000000	42978.000000

sales.describe()

- This provides a better statistical summary & understanding of the data
- For all numerical fields, I can have an idea of all statistical properties
    - Ex. I know that the average customer age present within this data set is 36 years old and the maximum age of a customer is 87 years old, min 17 years old
        - In this case, the mean is very close to the median
        -

In [34]:

```
# Numerical Analysis and Visualization

sales['Unit_Cost'].describe()
```

```
count    113036.000000
mean        267.296366
std         549.835483
min           1.000000
25%           2.000000
50%           9.000000
75%          42.000000
max        2171.000000
Name: Unit_Cost, dtype: float64
```

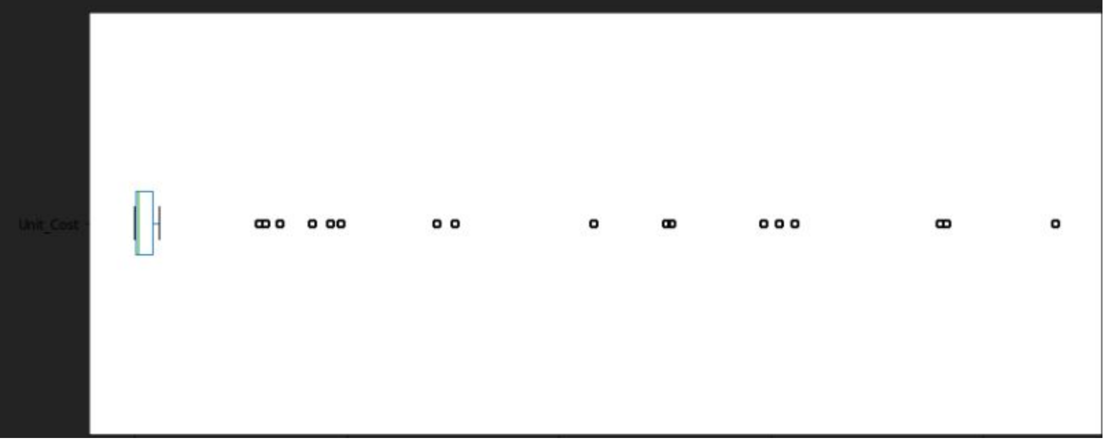
sales[‘Unit_Cost’].describe()

- This focuses on the Unit Cost column of the data frame

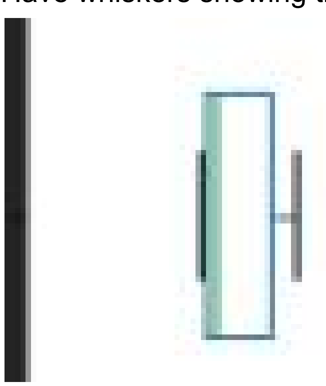
```
In [35]:  sales['Unit_Cost'].mean()

267.296365759581

In [36]:  sales['Unit_Cost'].median()

9.0
```

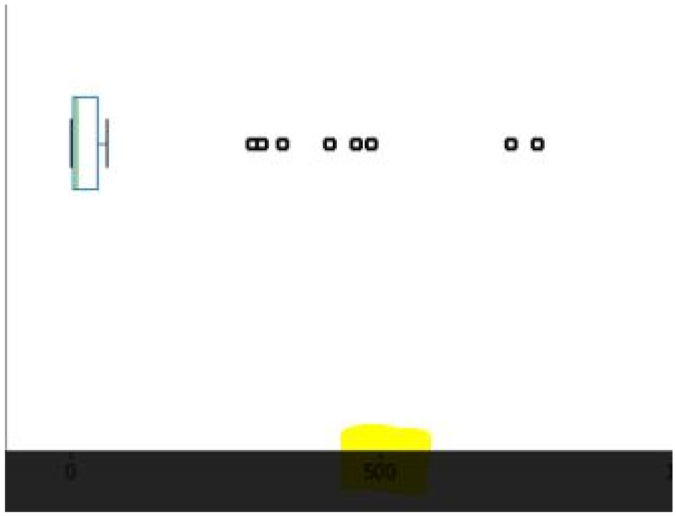
Mean and median are as shown previously when looked at statistical info regarding the entire dara frame

```
In [23]:  sales['Unit_Cost'].plot(kind = 'box', vert = False, figsize = (14, 6))

<AxesSubplot:>
```

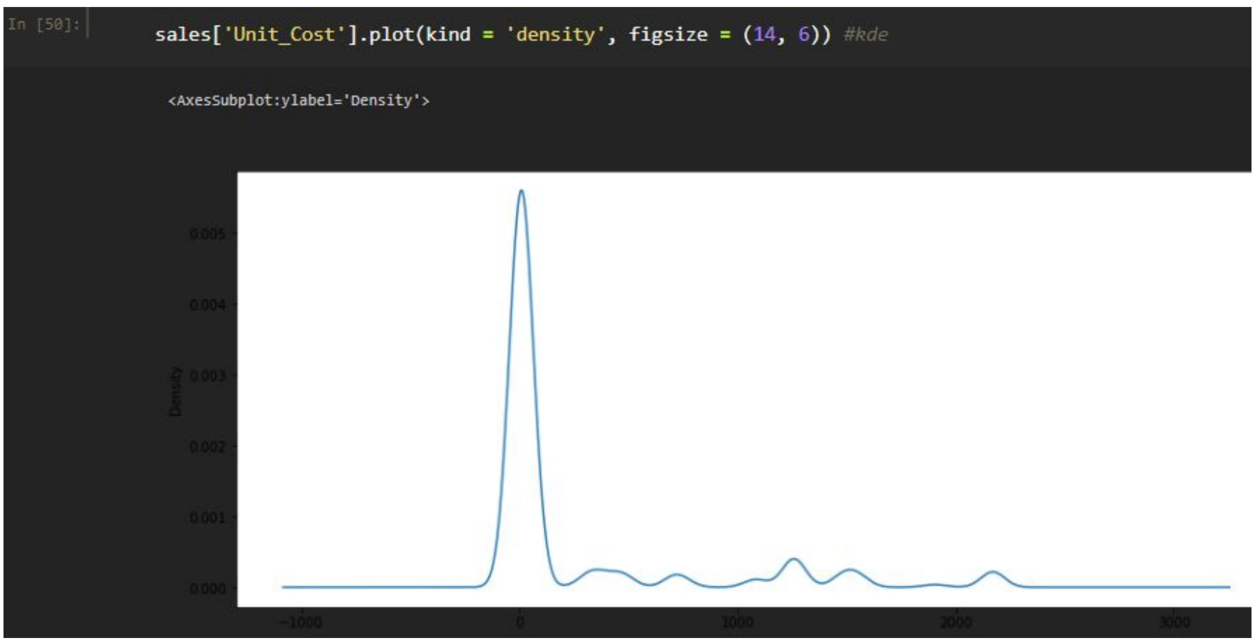


sales['Unit_Cost'].plot(kind = 'box', vert = False, figsize = (14, 6))

- This visualization is created using matplotlib, but we are doing it directly from pandas
- The box plot created is regarding Unit_Cost
    - Have whiskers showing the first and third quantile
    - 
    - And the median and the outliers to the right of the visualization

- 

    - Ex: a product that is $500 is considered to be an outlier

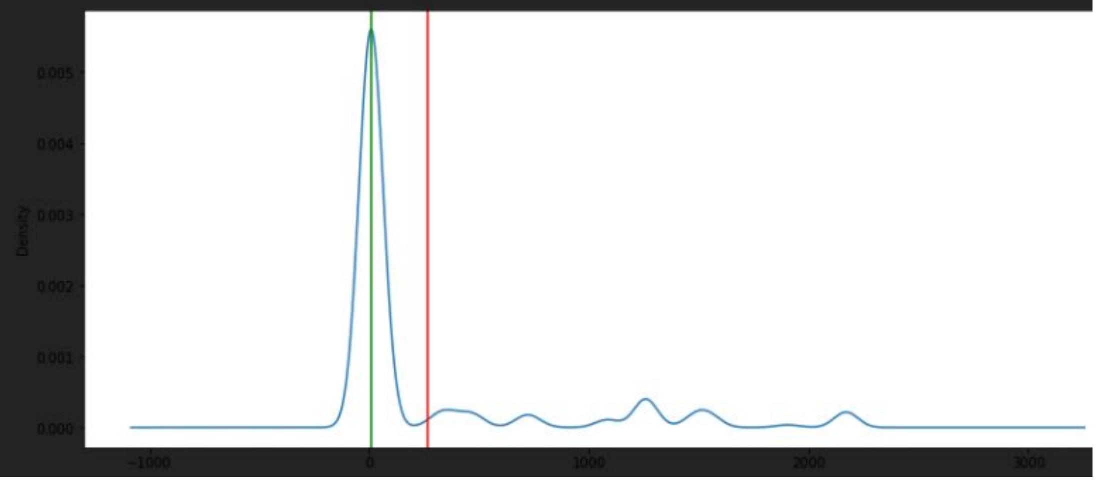


sales['Unit_Cost'].plot(kind = 'density', figsize = (14, 6))

- Density plot

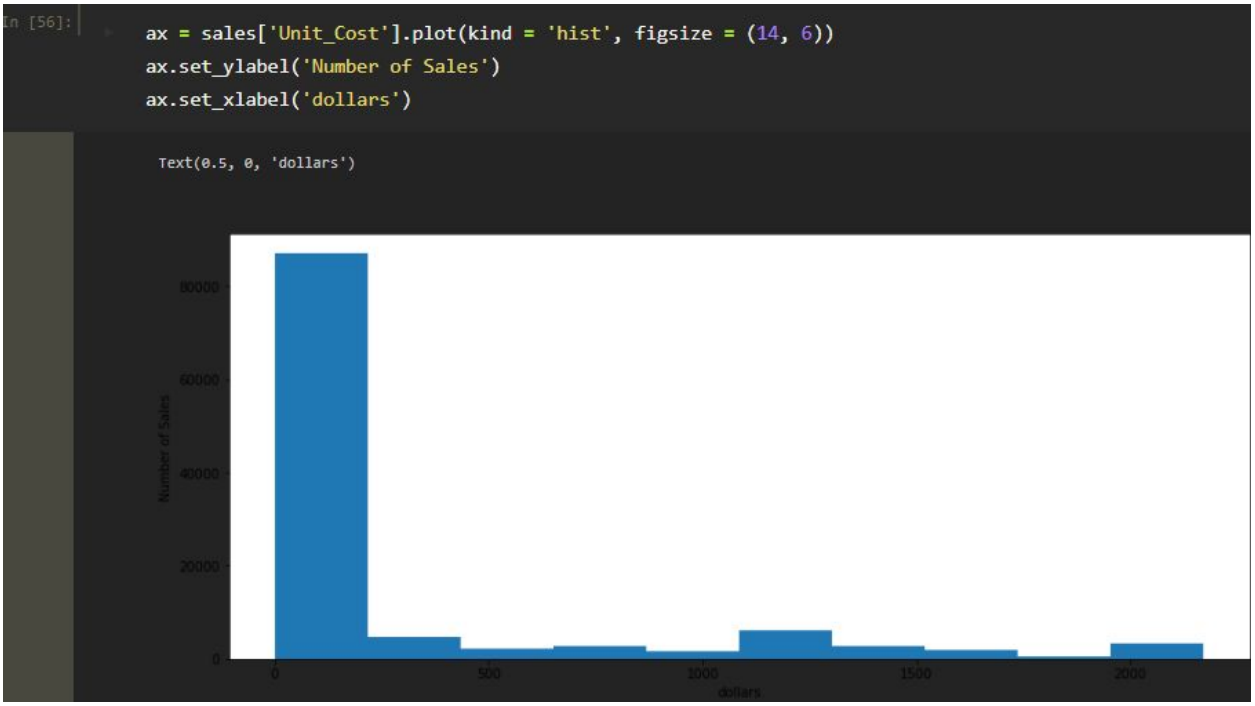
```
In [53]:  ax = sales['Unit_Cost'].plot(kind = 'density', figsize = (14, 6)) #kde
           ax.axvline(sales['Unit_Cost'].mean(), color = 'red')
           ax.axvline(sales['Unit_Cost'].median(), color = 'green')
```

```
<matplotlib.lines.Line2D at 0x10bafa30>
```



ax = sales[‘Unit_Cost’].plot(kind = ‘density’, figsize = (14, 6))
ax.axvline(sales[‘Unit_Cost’].mean(), color = ‘red’)
ax.axvline(sales[‘Unit_Cost’].median(), color = ‘green’)

- Shows mean and median within density plot




```
ax = sales['Unit_Cost'].plot(kind = 'hist', figsize = (14, 6))
ax.set_ylabel('Number of Sales')
ax.set_xlabel('dollars')
```

- Histogram

```
In [57]:
# Categorical Analysis and Visualization
# Analyze the Age_Group column
sales['Age_Group'].value_counts()
```

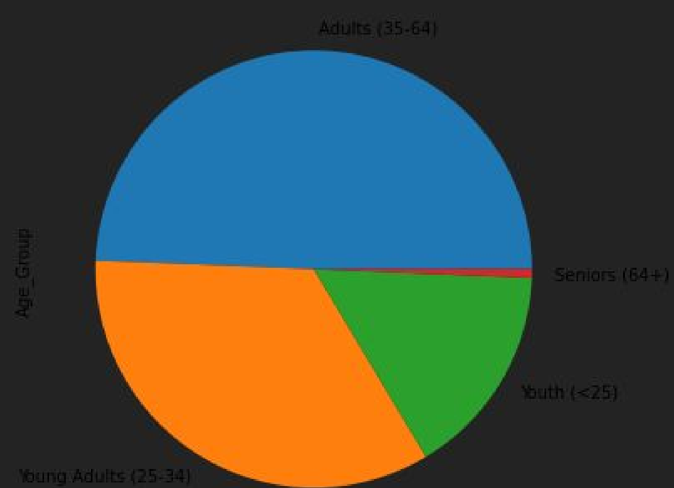
```
Adults (35-64)          55824
Young Adults (25-34)    38654
Youth (<25)             17828
Seniors (64+)             730
Name: Age_Group, dtype: int64
```

sales['Age_Group'].value_counts()

- Categories were created to better understand these groups

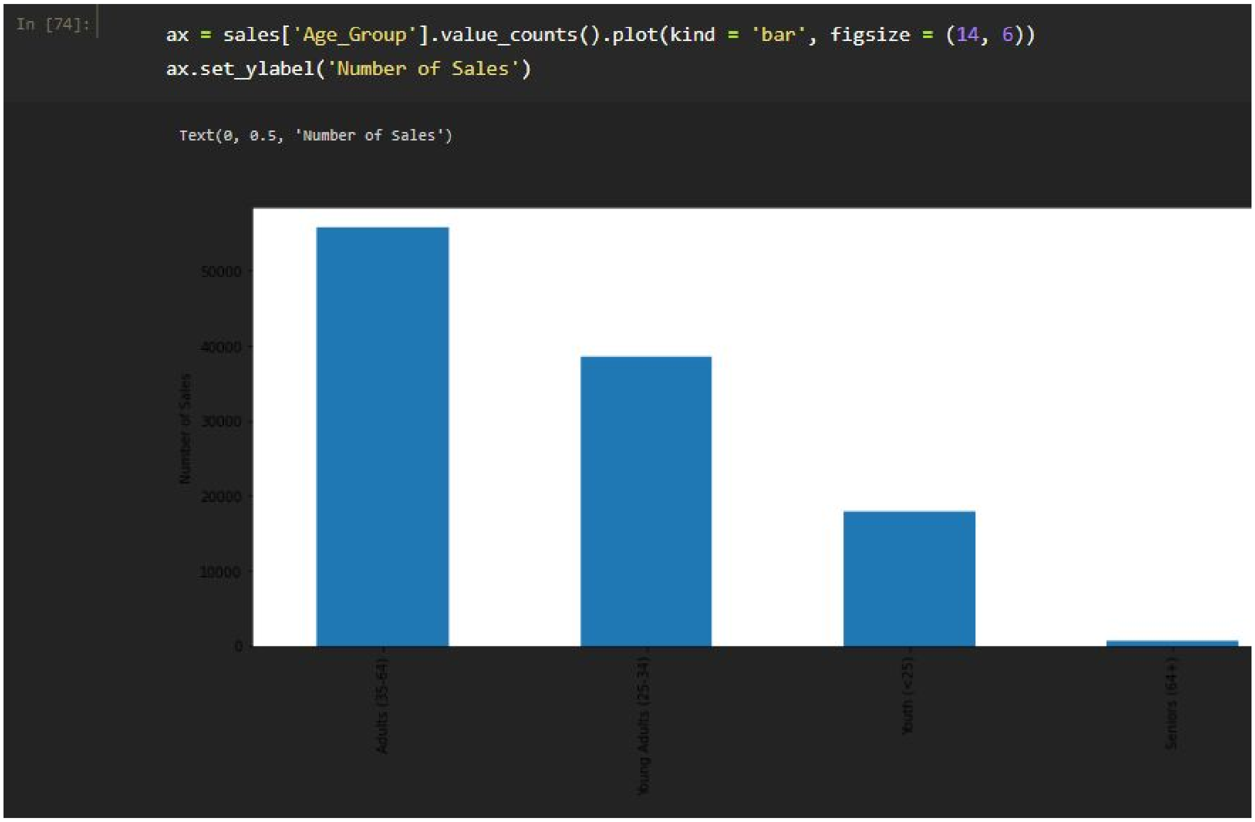
```
In [72]:
sales['Age_Group'].value_counts().plot(kind = 'pie', figsize=(6, 6))
```

```
<AxesSubplot:ylabel='Age_Group'>
```



sales['Age_Group'].value_counts().plot(kind = 'pie', figsize = (6, 6))

- Notice: adults are the largest group for our data



ax = sales[‘Age Group’].value_counts().plot(kind = ‘bar’, figsize = (14, 6))
ax.set_ylabel(‘Number of Sales’)

In [76]:

```
# Relationships between columns
corr = sales.corr()
corr
```

	Day	Year	Customer_Age	Order_Quantity	Unit_Cost	Unit_Price	Profit	Cost	Revenue
Day	1.000000	-0.007635	-0.014296	-0.002412	0.003133	0.003207	0.004623	0.003329	0.003853
Year	-0.007635	1.000000	0.040994	0.123169	-0.217575	-0.213673	-0.181525	-0.215604	-0.208673
Customer_Age	-0.014296	0.040994	1.000000	0.026887	-0.021374	-0.020262	0.004319	-0.016013	-0.009326
Order_Quantity	-0.002412	0.123169	0.026887	1.000000	-0.515835	-0.515925	-0.238863	-0.340382	-0.312895
Unit_Cost	0.003133	-0.217575	-0.021374	-0.515835	1.000000	0.997894	0.741020	0.829869	0.817865
Unit_Price	0.003207	-0.213673	-0.020262	-0.515925	0.997894	1.000000	0.749870	0.826301	0.818522
Profit	0.004623	-0.181525	0.004319	-0.238863	0.741020	0.749870	1.000000	0.902233	0.956572
Cost	0.003329	-0.215604	-0.016013	-0.340382	0.829869	0.826301	0.902233	1.000000	0.988758
Revenue	0.003853	-0.208673	-0.009326	-0.312895	0.817865	0.818522	0.956572	0.988758	1.000000

sales.corr()

- Ex: Profit and Unit_Cost have a correlation of 0.74

In [78]:

```
fig = plt.figure(figsize = (8, 8))
plt.matshow(corr, cmap = 'RdBu', fignum = fig.number)
plt.xticks(range(len(corr.columns)), corr.columns, rotation = 'vertical');
plt.yticks(range(len(corr.columns)), corr.columns);
```

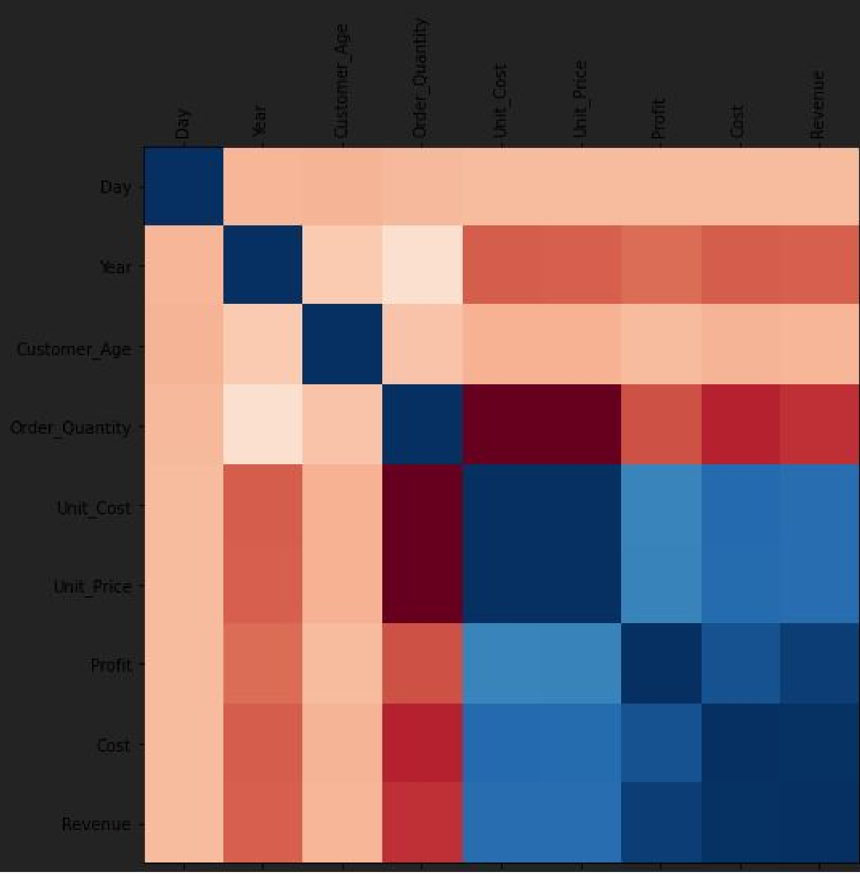


fig = plt.figure(figsize = (8,8))
plt.matshow(corr, cmap = ‘RdBu’, fignum = fig.number)
plt.xticks(range(len(corr.columns)), corr.columns, rotation = ‘vertical’);
plt.yticks(range(len(corr.columns)), corr.columns);

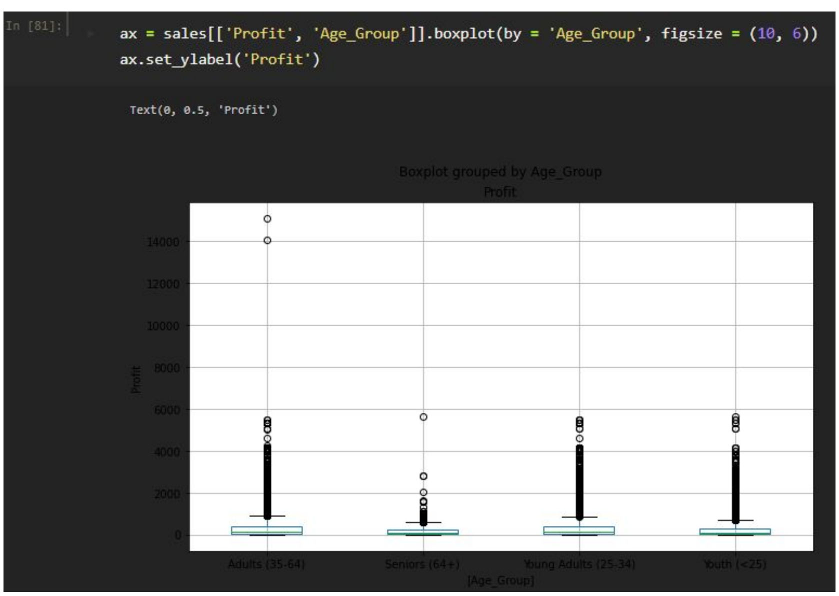
- Matrix correlation
- Blue shows high correlation while red shows low correlation (-ive correlation is shown by dark red)
- The blue diagonal shows columns with correlation of 1
- Notice: Profit has high correlation with Unit_Cost, Unit_Price, Cost, and Revenue
- In reference to the chart and the color map, notice that profit has negative correlation with Order_Quantity
- Profit has a high correlation with Revenue

sales.plot(kind = ‘scatter’, x = ‘Customer_Age’, y = ‘Revenue’, figsize = (6, 6))

- Checks the correlation between Customer_Age and Revenue

sales.plot(kind = ‘scatter’, x = ‘Revenue’, y = ‘Profit’, figsize = (6, 6))

- Checks the correlation between Revenue and Profit
- Can draw a linear diagonal
    - This shows linear dependency

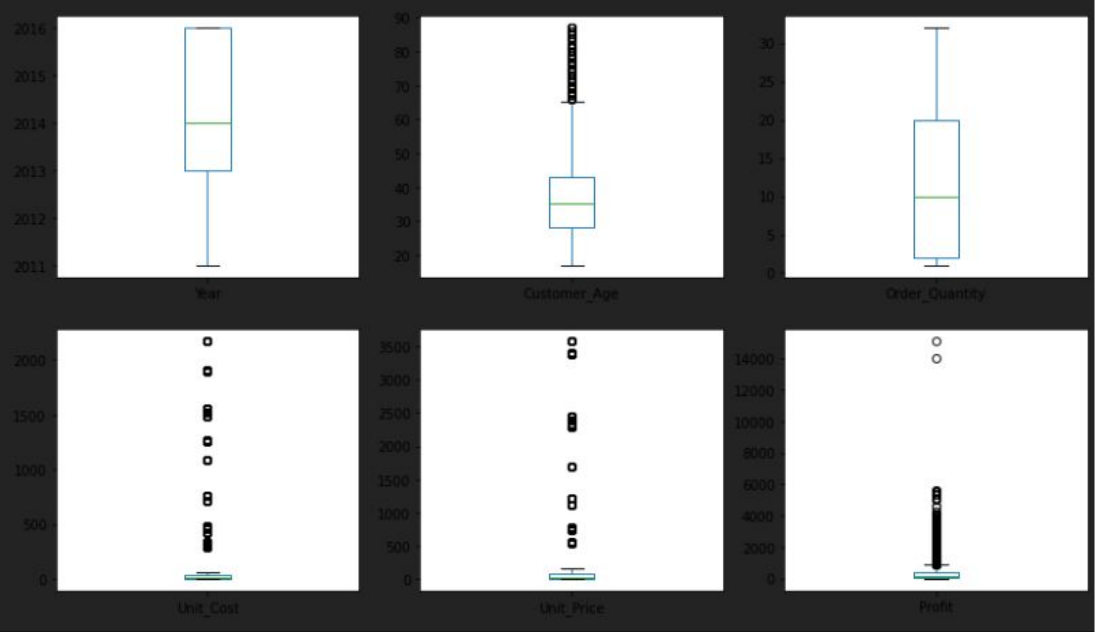


ax = sales[[‘Profit’, ‘Age_Group’]].boxplot(by = ‘Age_Group’, figsize = (10, 6))
ax.set_ylabel(‘Profit’)

- Box plots
- In this case this helps understand the profit per age group
    - Here, we see how the profit changes depending on the customer's age

In [82]:
```
boxplot_cols = ['Year', 'Customer_Age', 'Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Profit']
sales[boxplot_cols].plot(kind = 'box', subplots = True, layout = (2, 3), figsize = (14, 8))
```

```
Year                AxesSubplot(0.125,0.536818;0.227941x0.343182)
Customer_Age     AxesSubplot(0.398529,0.536818;0.227941x0.343182)
Order_Quantity   AxesSubplot(0.672059,0.536818;0.227941x0.343182)
Unit_Cost             AxesSubplot(0.125,0.125;0.227941x0.343182)
Unit_Price         AxesSubplot(0.398529,0.125;0.227941x0.343182)
Profit             AxesSubplot(0.672059,0.125;0.227941x0.343182)
dtype: object
```



boxplot_cols = ['Year', 'Customer_Age', 'Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Profit']
sales[boxplot_cols].plot(kind = 'box', subplots = True, layout = (2, 3), figsize = (14, 8))

In [83]:
```
# Column Wrangling
# Create new columns or modify existing ones

sales['Revenue_per_Age'] = sales['Revenue'] / sales['Customer_Age']
```

sales['Revenue_per_Age'] = sales['Revenue'] / sales['Customer_Age']

=R2/E2

D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Year	Customer_Age	Age_Group	Customer_Gend	Country	State	Product_Categor	Sub_Category	Product	Order_Quantity	Unit_Cost	Unit_Price	Profit	Cost	Revenue	Rev / Age
2013	19	Youth (<25)	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	8	45	120	590	360	950	50
2015	19	Youth (<25)	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	8	45	120	590	360	950	
2014	49	Adults (35-64)	M	Australia	New South Wale	Accessories	Bike Racks	Hitch Rack - 4-Bi	23	45	120	1366	1035	2401	
2016	49	Adults (35-64)	M	Australia	New South Wale	Accessories	Bike Racks	Hitch Rack - 4-Bi	20	45	120	1188	900	2088	
2014	47	Adults (35-64)	F	Australia	New South Wale	Accessories	Bike Racks	Hitch Rack - 4-Bi	4	45	120	238	180	418	
2016	47	Adults (35-64)	F	Australia	New South Wale	Accessories	Bike Racks	Hitch Rack - 4-Bi	5	45	120	297	225	522	
2014	47	Adults (35-64)	F	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	4	45	120	199	180	379	
2016	47	Adults (35-64)	F	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	2	45	120	100	90	190	
2014	35	Adults (35-64)	M	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	22	45	120	1096	990	2086	
2016	35	Adults (35-64)	M	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	21	45	120	1046	945	1991	
2013	32	Young Adults (25	F	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	8	45	120	398	360	758	
2015	32	Young Adults (25	F	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	8	45	120	398	360	758	
2013	34	Young Adults (25	M	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	7	45	120	349	315	664	
2015	34	Young Adults (25	M	Australia	Victoria	Accessories	Bike Racks	Hitch Rack - 4-Bi	7	45	120	349	315	664	
2013	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	5	45	120	369	225	594	
2015	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	7	45	120	517	315	832	28.68965517
2013	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	2	45	120	148	90	238	8.206896552
2015	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	1	45	120	74	45	119	4.103448276
2014	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	1	45	120	74	45	119	4.103448276
2016	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	1	45	120	74	45	119	4.103448276
2014	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	6	45	120	443	270	713	24.5862069
2016	29	Young Adults (25	M	Canada	British Columbia	Accessories	Bike Racks	Hitch Rack - 4-Bi	8	45	120	590	360	950	32.75862069

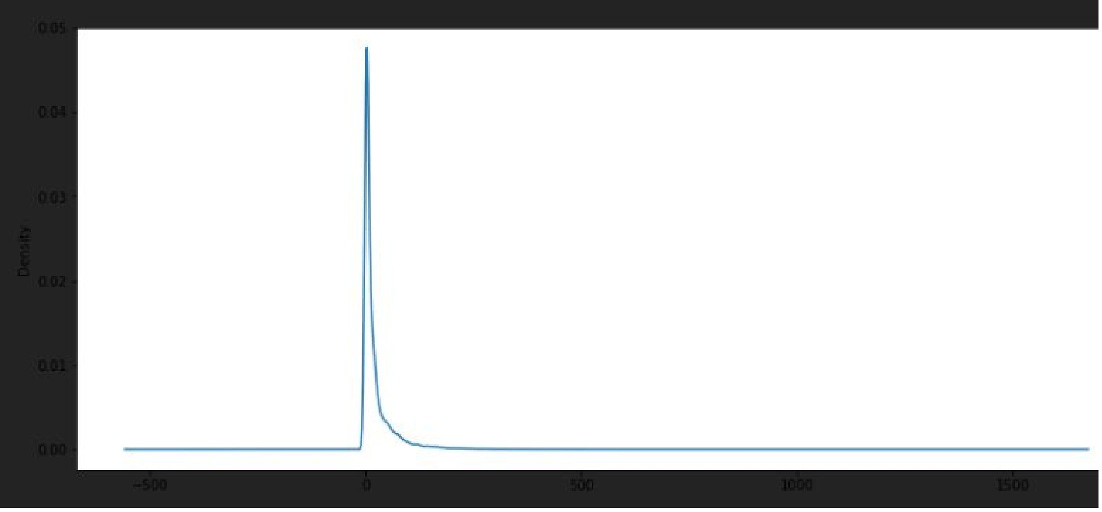
- Similar to excel

```
In [84]: sales['Revenue_per_Age'].head()

0    50.000000
1    50.000000
2    49.000000
3    42.612245
4     8.893617
Name: Revenue_per_Age, dtype: float64
```

sales['Revenue_per_Age'].head()

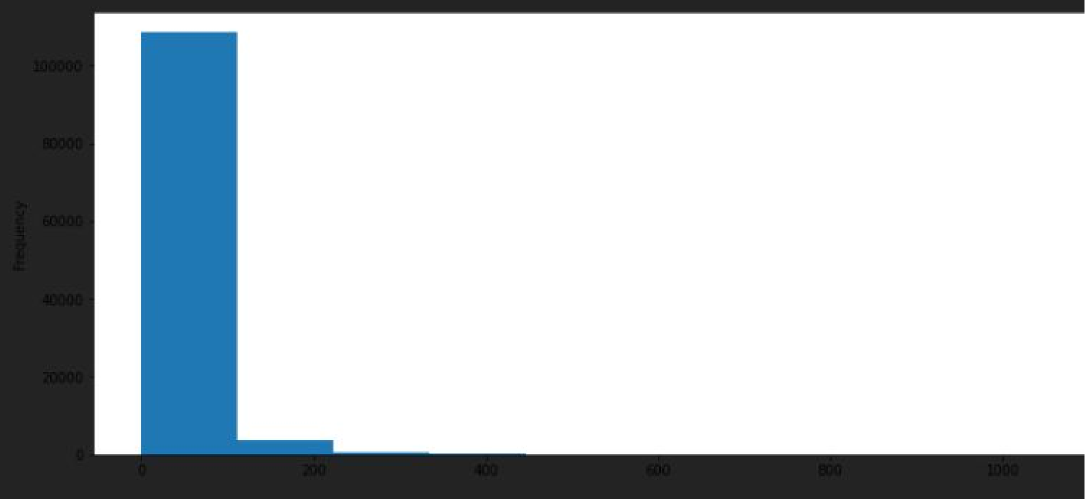
- Python is much more faster at calculating this than Excel

```
In [88]: sales['Revenue_per_Age'].plot(kind = 'density', figsize = (14,6))

<AxesSubplot:ylabel='Density'>
```



sales['Revenue_per_Age'].plot(kind = 'density', figsize = (14, 6))

```
In [89]:   sales['Revenue_per_Age'].plot(kind = 'hist', figsize = (14, 6))

<AxesSubplot:ylabel='Frequency'>
```



sales['Revenue_per_Age'].plot(kind = 'hist', figsize(14, 6))

Add and calculate new *Calculated_Cost* column
Calculated_Cost* = *Order_Quantity* * *Unit_Cost

```
# Use the following formula:
# Calculated_Cost = Order_Quantity * Unit_Cost

In [117]:   sales['Calculated_Cost'] = sales['Order_Quantity'] * sales['Unit_Cost']
            sales['Calculated_Cost'].head()

0     360
1     360
2    1035
3     900
4     180
Name: Calculated_Cost, dtype: int64
```

sales['Calculated_Cost'] = sales['Order_Quantity'] * sales['Unit_Cost']
sales['Calculated_Cost'].head()

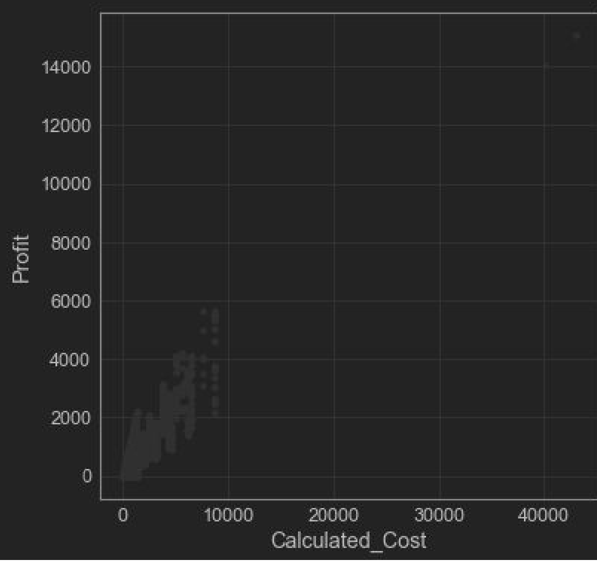
- Pretty much the orders * cost

```
In [121]:   (sales['Calculated_Cost'] != sales['Cost']).sum()

0
```

(sales['Calculated_Cost'] != sales['Cost']).sum()

- How many rows had a different value than what was provided by *Cost*?
- This checks if the *Cost* provided by the data set, at some point does <u>not</u> align with the *Calculated_Cost* (this is what we calculated above)
- This is good for checking if something went wrong during data entry

```
In [119]:
# Can look at the relationship between Cost and Profit using a scatter plot
sales.plot(kind = 'scatter', x = 'Calculated_Cost', y = 'Profit', figsize = (6, 6) )

<AxesSubplot:xlabel='Calculated_Cost', ylabel='Profit'>
```



sales.plot(kind = 'scatter', x = 'Calculated_Cost', y = 'Profit', figsize = (6, 6))

- This is a quick regression plot
- In this case, it shows that there exists <u>linear dependency</u> between *Calculated_Cost* and *Profit*

Add and calculate new *Calculated_Revenue* column

Revenue

In accounting, revenue is the income or increase in net assets that an entity has from its normal activities. Commercial revenue may also be referred to as sales or as turnover. Some companies receive revenue from interest, royalties, or other fees. Wikipedia

```
In [107]:  # Add and calculate a new Calculated_Revenue column

           # Use the following formula:
           # Calculated_Revenue = Cost + Profit

In [120]:  sales['Calculated_Revenue'] = sales['Cost'] + sales['Profit']
           sales['Calculated_Revenue'].head()

            0     950
            1     950
            2    2401
            3    2088
            4     418
            Name: Calculated_Revenue, dtype: int64
```

sales['Calculated_Revenue'] = sales['Cost'] + sales['Profit']
sales['Calculated_Revenue'].head()

```
In [122]:  (sales['Calculated_Revenue'] != sales['Revenue']).sum()

            0
```

(sales['Calculated_Revenue'] != sales['Revenue']).sum()

- Similar to Calculated_Cost above
- How many rows had a different value than what was provided by *Revenue*?
- This checks if the *Revenue* provided by the data set
    - Whether there exists a point that does not align with the *Calculated_Revenue* (this is what we calculated above)
- Again, this is good for checking if something went wrong during data entry

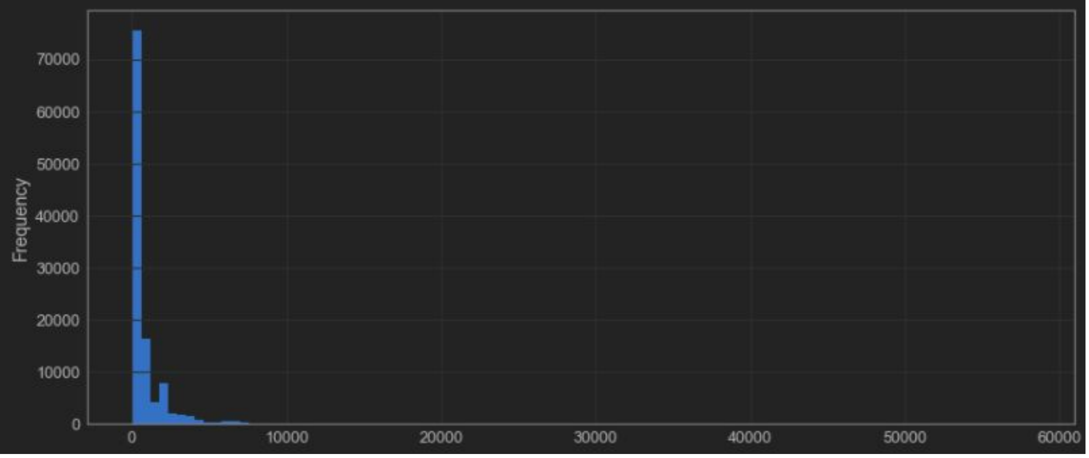
```
In [123]:  sales.head()
```

uct	Order_Quantity	Unit_Cost	Unit_Price	Profit	Cost	Revenue	Revenue_per_Age	Calculated_Cost	Calculated_Revenue
-4-	8	45	120	590	360	950	50.000000	360	950
-4-	8	45	120	590	360	950	50.000000	360	950
-4-	23	45	120	1366	1035	2401	49.000000	1035	2401
-4-	20	45	120	1188	900	2088	42.612245	900	2088
-4-	4	45	120	238	180	418	8.893617	180	418

sales.head()

- Notice: *Calculated_Cost* and *Calculated_Revenue* are now included within the *sales* data frame

```
In [126]: sales['Revenue'].plot(kind = 'hist', bins = 100, figsize = (14, 6))

<AxesSubplot:ylabel='Frequency'>
```



sales[‘Revenue’].plot(kind = ‘hist’, bins = 100, figsize = (14, 6))

- Histogram of the Revenue

Modify all *Unit_Price* values adding 3% tax to them

```
In [132]: sales['Unit_Price'].head()

0    120
1    120
2    120
3    120
4    120
Name: Unit_Price, dtype: int64
```

- sales[‘Unit_Price’].head()
- Values within the data <u>before</u> adding 3% tax to *Unit_Price*

```
In [134]: #sales['Unit_Price'] = sales['Unit_Price'] * 1.03
          sales['Unit_Price'] *= 1.03
```

sales[‘Unit_Price’] *= 1.03

- Adding 3% tax to values within *Unit_Price*
    - Increasing everything by .03

```
In [135]:    sales['Unit_Price'].head()

0    123.6
1    123.6
2    123.6
3    123.6
4    123.6
Name: Unit_Price, dtype: float64
```

sales['Unit_Price'].head()

- These are the values within *Unit_Price* after the 3% increase

Quick Filtering
Selection & Indexing

Get all the sales made in the state of Kentucky

Regarding Explanations, look at this:
https://medium.com/analytics-vidhya/basic-tools-to-learn-in-data-analysis-with-python-5b9b4a7a1b61

Review correlation

References

https://www.youtube.com/watch?v=r-uOLxNrNk8

https://github.com/ine-rmotr-curriculum/FreeCodeCamp-Pandas-Real-Life-Example

https://docs.google.com/presentation/d/1fDpjIyMiOMJyuc7_jMekcYLPP2XISl1eWw9F7yE7byk/edit#slide=id.g6fe1465eda_0_215