Exploratory Data Analysis (EDA) in Python -

Michelle Moloney King

- Apply practical Exploratory Data Analysis (EDA) techniques on any tabular dataset using Python packages such as Pandas and Numpy.

- Produce data visualisations using Seaborn and Matplotlib

- I used external Python packages such as Pandas, Numpy, Matplotlib, Seaborn etc. to conduct univariate analysis, bivariate analysis, correlation analysis and identify and handle duplicate/missing data.

- Link to data source: https://www.kaggle.com/aungpyaeap/supermarket-sales

Context

The growth of supermarkets in most populated cities is increasing and market competitions are high. This dataset contains the historical sales of a supermarket company which has recorded in 3 different branches for 3 months of data.

The directory is made up of Invoice id, Brach, City, Customer Type, Gendre, Product Line, Unit Price, Quantity, Tax, Total, Date, Time, payment, COGS (cost of goods sold), Gross margin percentage, Gross income, and Rating.

Steps taken

- Initial Data Exploration: Read in data, take a glimpse at a few rows, calculate some summary statistics.
- Univariate Analysis: Analyse continuous and categorical variables, one variable at a time.

  Bivariate Analysis: Looking at the relationship between two variables at a time.
- Identify and Handling Duplicate and Missing Data: Find and remove duplicate rows, and replace missing values with their mean and mode.
- Correlation Analysis: Looking at the correlation of numerical variables in the dataset and interpreting the numbers.

Tasks

Data Exploration

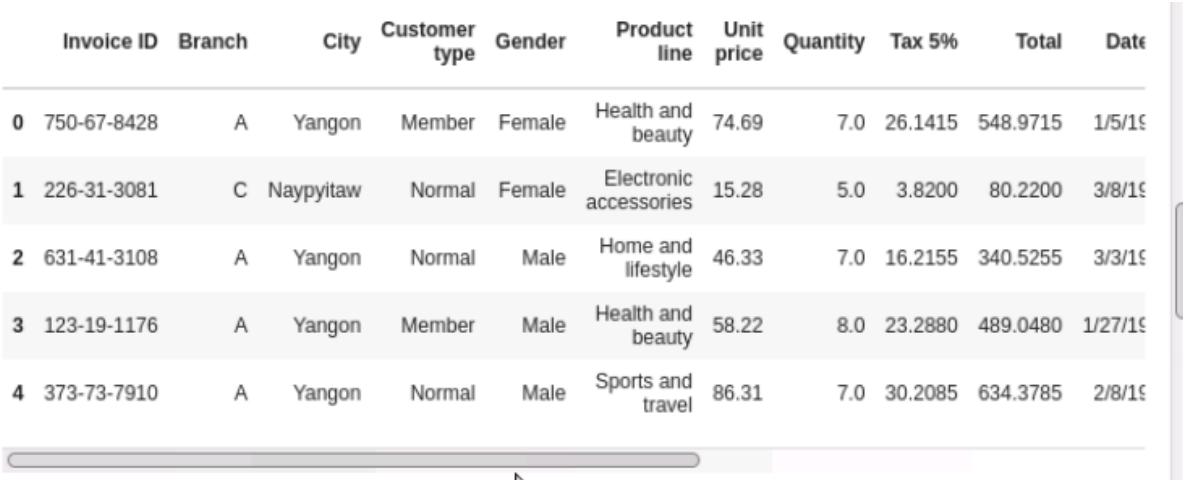
Set up a Juniper notebook, link the dataset, import these libraries to help run the data.

```
Import pandas as pd
Import numpy as np
Import matlotlib.pyplot as plt
Import seaborn as sns
Import calmap
From pandas_profiling Import  ProfileReport
```

I ran this python panda code to head the head of the database. I I wanted to read the end then I'd use df.TAIL

```
df = pd.read_csv('supermarket_sales.csv')
df.head()
```

We now see the first 5 rows



	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7.0	26.1415	548.9715	1/5/19
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5.0	3.8200	80.2200	3/8/19
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7.0	16.2155	340.5255	3/3/19
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8.0	23.2880	489.0480	1/27/19
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7.0	30.2085	634.3785	2/8/19

To learn more about what are the columns I used

```
df.columns
```

WHICH GIVES:

```
Index(['Invoice ID', 'Branch', 'City', 'Customer type', 'Gender',
       'Product line', 'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Date',
       'Time', 'Payment', 'cogs', 'gross margin percentage', 'gross income',
       'Rating'],
```

dtype='object')

Data Types

To learn what the data types are - Issued

df.dtypes

Which got:
Invoice ID object
Branch object
City object
Customer type object
Gender object
Product line object
Unit price float64
Quantity float64
Tax 5% float64
Total float64
Date object
Time object
Payment object
cogs float64
gross margin percentage float64
gross income float64
Rating float64
dtype: object

These columns are object or float. But the date is type object when it needs to be type date. To change this I used

df['Date'] = pd.to_datetime(df['Date'])

This will tell the column to change to a datetime type, to check this, run the query again
df.dtypes

```
27]: Invoice ID                      object
     Branch                          object
     City                            object
     Customer type                   object
     Gender                          object
     Product line                    object
     Unit price                     float64
     Quantity                       float64
     Tax 5%                         float64
     Total                          float64
     Date                    datetime64[ns]
     Time                            object
     Payment                         object
     cogs                           float64
     gross margin percentage        float64
     gross income                   float64
     Rating                         float64
     dtype: object
```

It worked. The type changed from object to datetime and from strings (1/3/19) when I queried df['Date'] and object data type when I queried the column datatypes with df.dtypes.

Using
df['Date'] = pd.to_datetime(df['Date']) and running this **df['Date']** shows me the dates are now in the format I need.

```
0      2019-01-05
1      2019-03-08
2      2019-03-03
3      2019-01-27
4      2019-02-08
          ...
998    2019-02-22
999    2019-02-18
1000   2019-02-18
1001   2019-03-10
1002   2019-01-26
Name: Date, Length: 1003, dtype: datetime64[ns]
```

Task 2: Univariate Analysis

Question 1: What does the distribution of customer ratings looks like?

Following convention I now need to set the date column as the index for the dataframe.

df.set_index('date',inplace=True)

I used true to ensure it is a permanent change.

To learn more about the summary statics of the data:

df.describe()

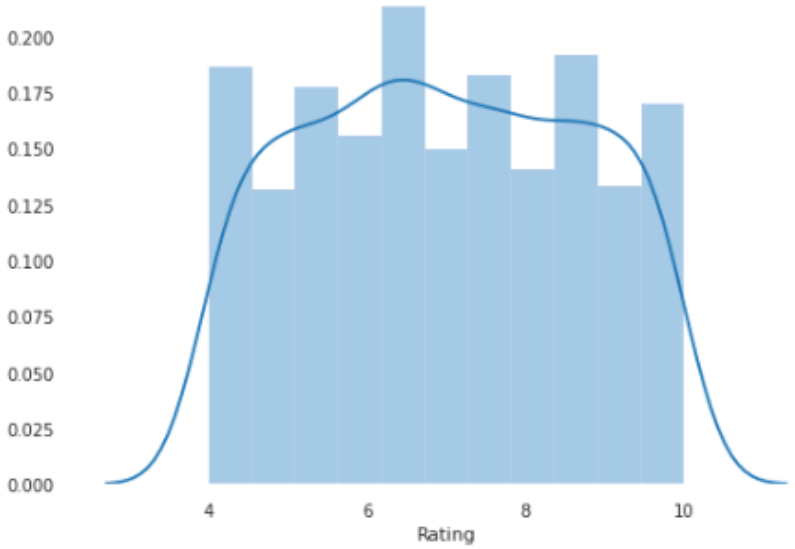
	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
count	996.000000	983.000000	1003.000000	1003.000000	1003.000000	1003.000000	1003.000000	1003.000000
mean	55.764568	5.501526	15.400368	323.407726	308.007358	4.761905	15.400368	6.972682
std	26.510165	2.924673	11.715192	246.019028	234.303836	0.000000	11.715192	1.717647
min	10.080000	1.000000	0.508500	10.678500	10.170000	4.761905	0.508500	4.000000
25%	33.125000	3.000000	5.894750	123.789750	117.895000	4.761905	5.894750	5.500000
50%	55.420000	5.000000	12.096000	254.016000	241.920000	4.761905	12.096000	7.000000
75%	78.085000	8.000000	22.539500	473.329500	450.790000	4.761905	22.539500	8.500000
max	99.960000	10.000000	49.650000	1042.650000	993.000000	4.761905	49.650000	10.000000

EDA is not straightforward so I am going to ask some questions and then use

Univariate Analysis

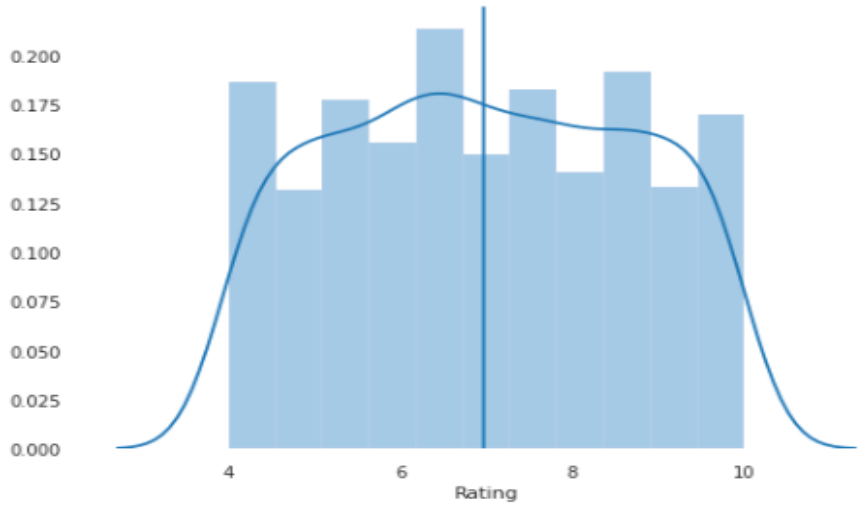
What does the distribution of customer ratings look like?
I used the Seaborn library (which I imported earlier) to plot out this distribution.
sns.distplot(df['Rating'])

```
: <AxesSubplot:xlabel='Rating'>
```



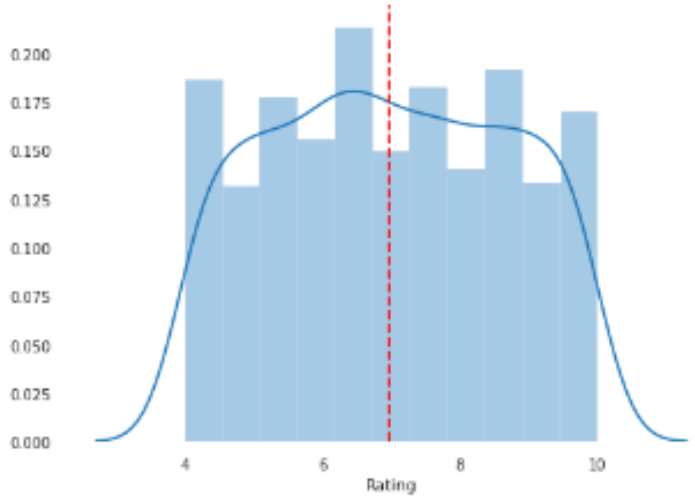
There is uniform distribution, so none of the rating numbers spike. TO plot the mean of the rating in this graph

plt.axvline(x=np.mean(df['Rating'])



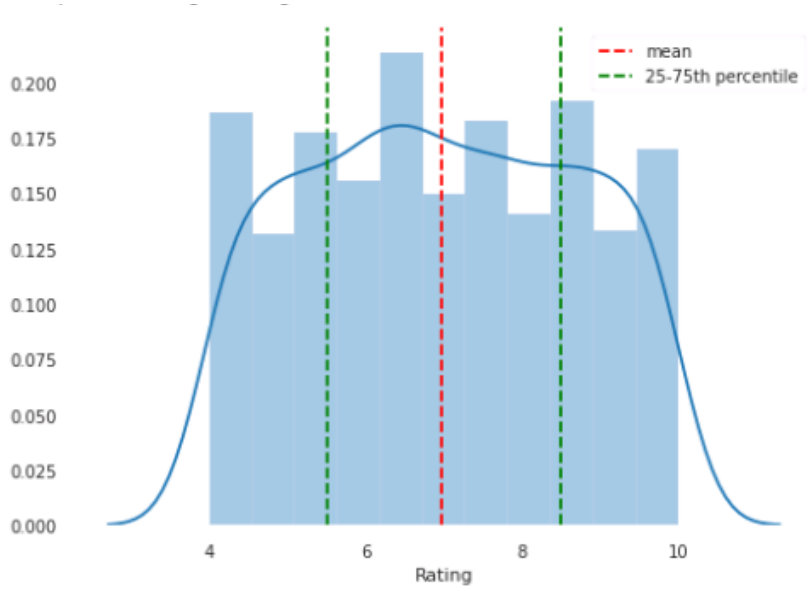
To change the line style and colour of the mean line to make it stand out more I added this to the above code: (ensuring I removed the last) and ended the code with the removed ‘)’

,c=’red’,ls=’--’)



I then added a 25% and 75% percentile and added labels to make it easier to understand. To show the labels I added a legend. It’s the same code as above with extra at the end:

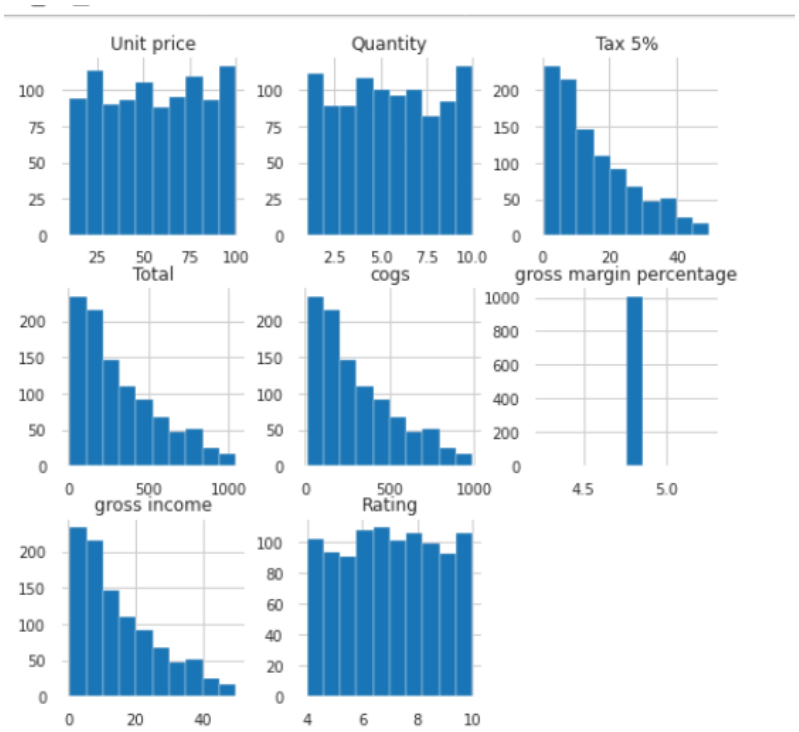
```
sns.distplot(df['Rating'])
plt.axvline(x=np.mean(df['Rating']),c='red',ls='--',label='mean')
plt.axvline(x=np.percentile(df['Rating'],25),c='green',ls='--',label='25-75th percentile')
plt.axvline(x=np.percentile(df['Rating'],75),c='green',ls='--')
plt.legend()
```



No, now I know that the distribution of customer ratings seems to be uniform and **there is no skew in either direction.**

Histograms

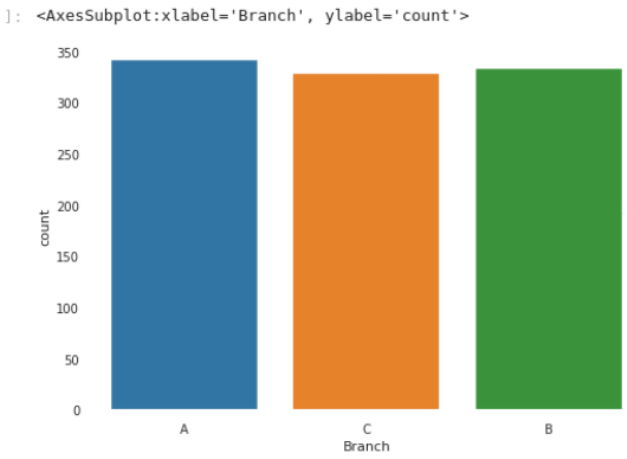
To run the histograms I use: **df.hist(figsize=(10,10)**



Do aggregate sales numbers differ by much between branches?

I use the Seaborn library to answer this. I use Branch as each row in the dataset represents each sale.

sns.countplot(df['Branch'])



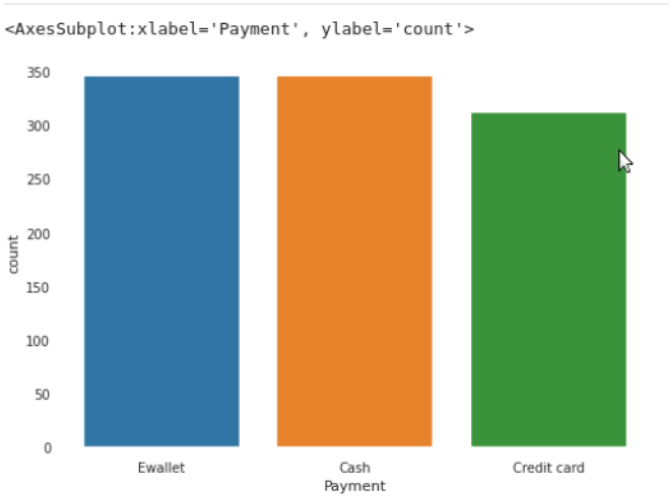
To get the exact number of sales I use:

df['Branch'].value_counts()
Which tells us each branches sales:

A 342
B 333
C 328

So now I want to investigate the users' payment methods.

sns.countplot(df['Payment'])



Ewallet is the most popular form of payment.

I looked at one variable at a time and now I will look at more than one variable; Bivariables, to understand the relationship between them.

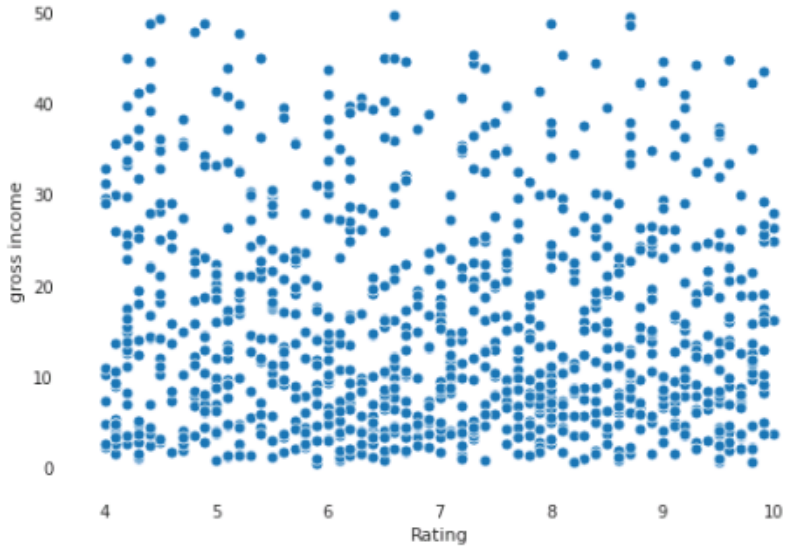
Bivariate Analysis - gross income and customer ratings

Is there a relationship between gross income and customer ratings?

I will use seaborn library and make a scatter plot to see what the relationship is between customer rating and gross income. Or how ratings are influenced by gross income. But as we see from the scatterplot there is no relationship.

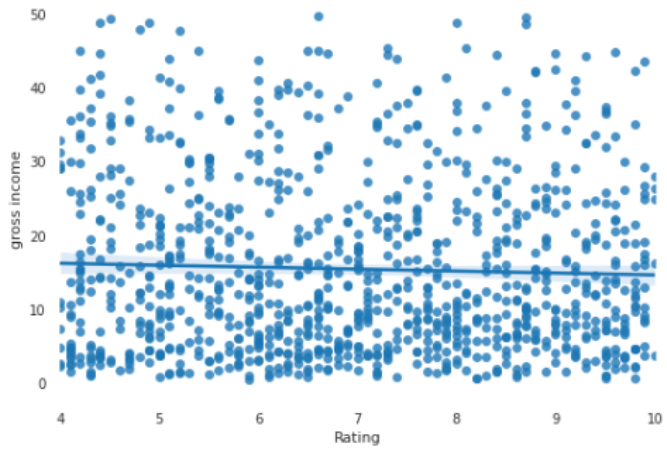
sns.scatterplot(df['Rating'],df['gross income'])

```
<AxesSubplot:xlabel='Rating', ylabel='gross income'>
```



To ensure my findings are correct I replaced scatter plot with regplot to see the trendline. As you can see below it is quite flat. So no relationship between customer rating and gross income.

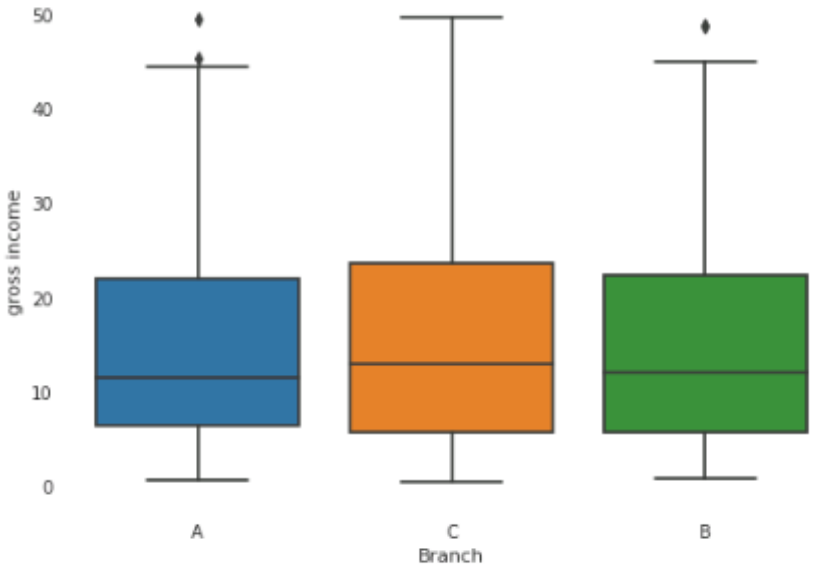
```
: <AxesSubplot:xlabel='Rating', ylabel='gross income'>
```



Is there a relationship between branch and total income? Again I use seaborn library and I'll plot a box plot to show the ranges. The medium line shows me that there isn't that much of a difference between the branches.

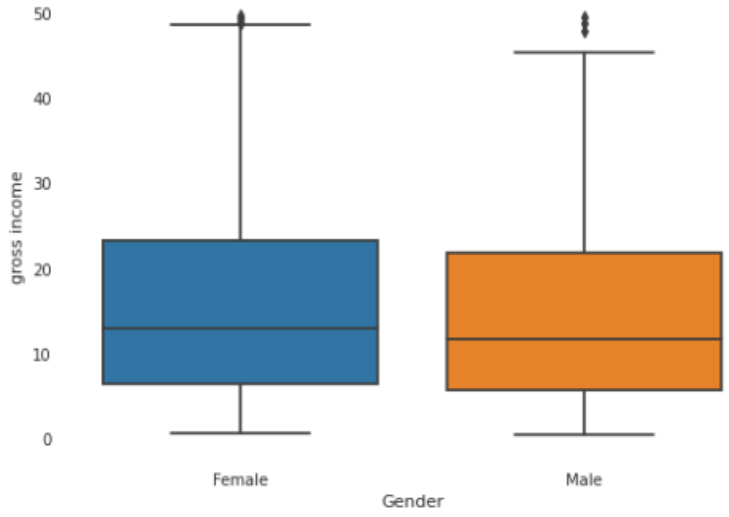
sns.boxplot(x=df['Branch'],y=df['gross income'])

```
: <AxesSubplot:xlabel='Branch', ylabel='gross income'>
```



I will now drill down into gender and research if there is a relationship between gender and gross income. I can use the same code but swap out Gender for Branch
sns.boxplot(x=df['Gender'],y=df['gross income'])

```
: <AxesSubplot:xlabel='Gender', ylabel='gross income'>
```



I can see there are no major differences, these results are similar.

Is there a noticeable time trend in gross income?

I need to change the data frame as there could be multiple customers at a given date, so I'll GROUPBY the dates to aggregate the data. The dates were changed to index so that term is used in my group by statement. I will group by the mean to get the mean of each date. Each row represents the average mean value for each day. 89 rows, in the below screenshot, represent 89 days.

df.groupby(df.index).mean()

Date	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
2019-01-01	54.995833	6.454545	18.830083	395.431750	376.601667	4.761905	18.830083	6.583333
2019-01-02	44.635000	6.000000	11.580375	243.187875	231.607500	4.761905	11.580375	6.050000
2019-01-03	59.457500	4.625000	12.369813	259.766062	247.396250	4.761905	12.369813	8.112500
2019-01-04	51.743333	5.333333	12.886417	270.614750	257.728333	4.761905	12.886417	6.516667
2019-01-05	61.636667	4.583333	14.034458	294.723625	280.689167	4.761905	14.034458	7.433333
...	...	...	...	...	...	...	...	...
2019-03-26	42.972308	4.000000	7.188692	150.962538	143.773846	4.761905	7.188692	6.623077
2019-03-27	56.841000	4.500000	13.822950	290.281950	276.459000	4.761905	13.822950	6.760000
2019-03-28	45.525000	4.800000	10.616200	222.940200	212.324000	4.761905	10.616200	7.050000
2019-03-29	66.346250	6.750000	23.947875	502.905375	478.957500	4.761905	23.947875	6.925000
2019-03-30	67.408182	5.888889	19.424500	407.914500	388.490000	4.761905	19.424500	6.800000

89 rows × 8 columns

To plot this out I will use Seaborn library and the line plot with mean on the x axis and gross income on the y axis.

```
sns.lineplot(x=df.groupby(df.index).mean().index,
             y=df.groupby(df.index).mean()['gross income'])
```

There doesn't seem to be any particular trend here. There are some spikes but nothing to suggest a noted correlation.

Plotting BiVariants

To plot all the Bivariants possible I'll use pairplot. This will plot, in pair, for every pair in the dataset. (This can only be done for a small dataset.)
sns.pairplot(df)

Dealing With Duplicate Rows and Missing Values

df.duplicated().sum()
This will show us duplicate rows, we can see what they are.
df(df.duplicated()==True)

Date	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	
2019-02-18	849-09-3807	A	Yangon	Member	Female	Fashion accessories	88.34	7.0	30.919	6
2019-03-10	745-74-0715	A	Yangon	Normal	Male	Electronic accessories	NaN	2.0	5.803	1
2019-01-26	452-04-8808	B	Mandalay	Normal	Male	Electronic accessories	87.08	NaN	30.478	6

To drop duplicated data
df.drop_duplicates(inplace=True)

Then verify that this change was flushed through with
df.duplicated().sum()

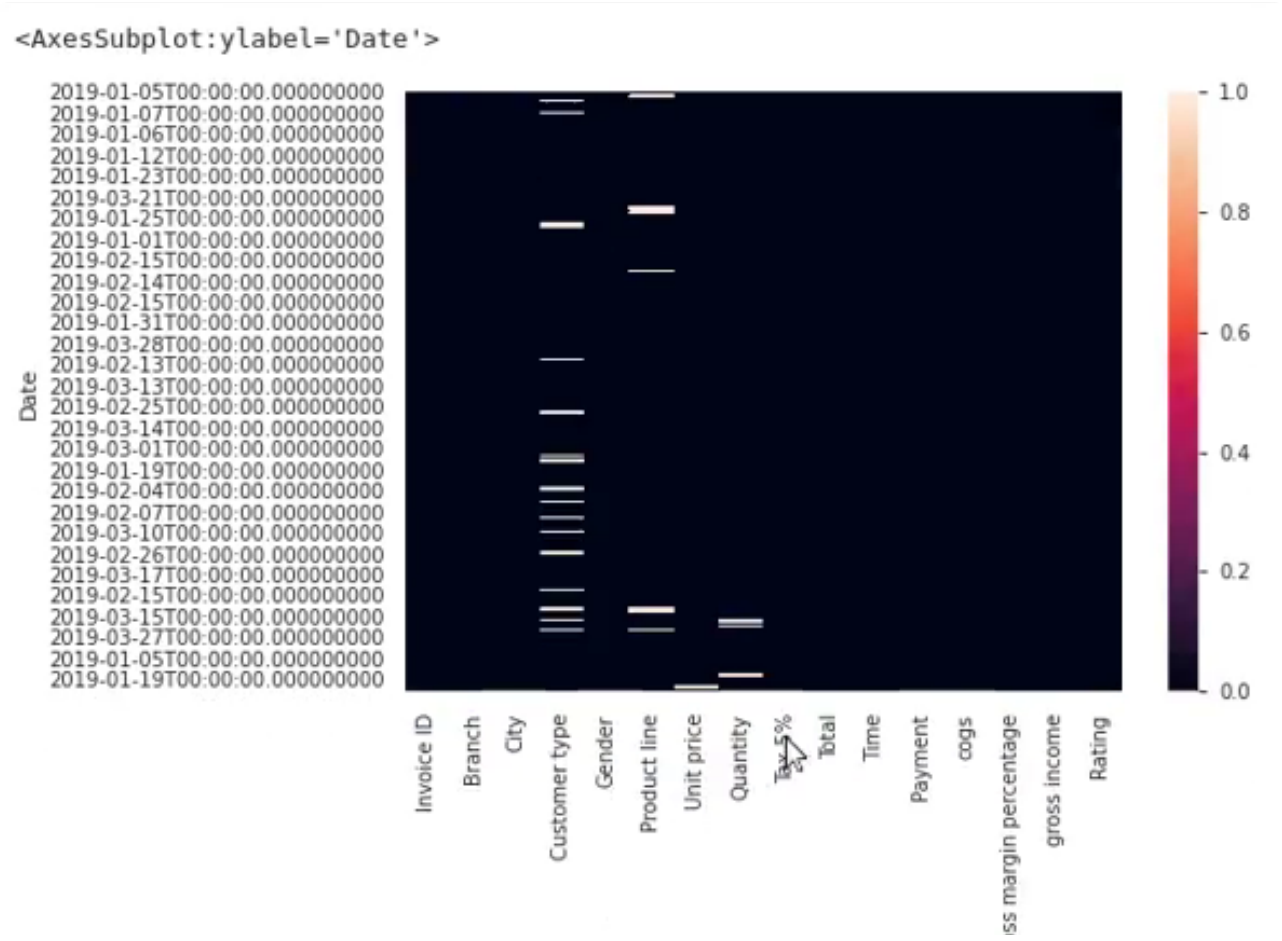
To see the total number of missing values per column
df.isna().sum()

```
Invoice ID                 0
Branch                     0
City                       0
Customer type             79
Gender                     0
Product line              43
Unit price                 6
Quantity                  19
Tax 5%                     0
Total                      0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

Visualising Missing Data Values

To visualise this missing data I will use a heatmap
sns.heatmap(df.isnull(),cbar=false)

The white spaces show where there is missing data.

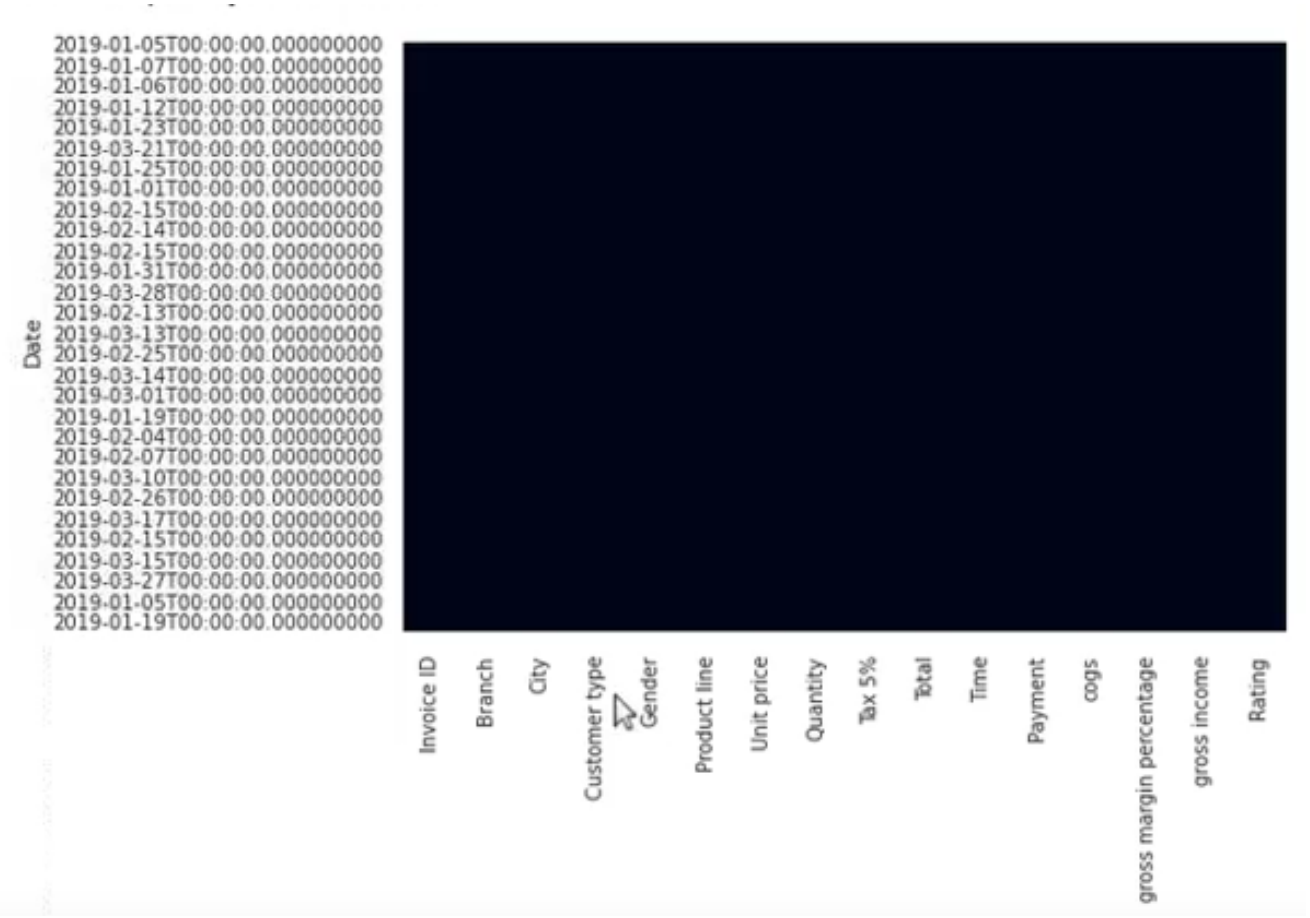


Find/Replace Missing Values

To find or create data for these missing values, we can use the mean of the column (for numeric columns) and not a categorical value. For those columns I need a mode for those columns.

df.fillna(df.mean(),inplace=True)
df.fillna(df.mode(),iloc(0),inplace=True
I use zero in this as I want to start this at the zero index location and move outwards.

And now to check if this worked, I'll replot the heatmap with missing data
sns.heatmap(df.isnull(),cbar=false)

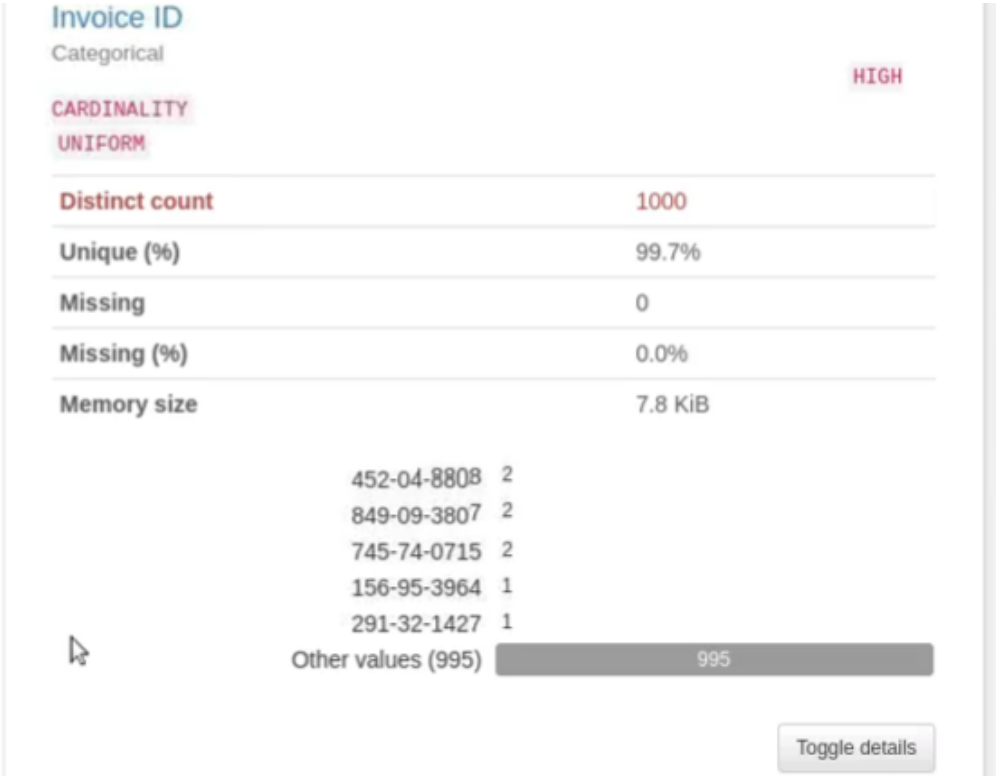


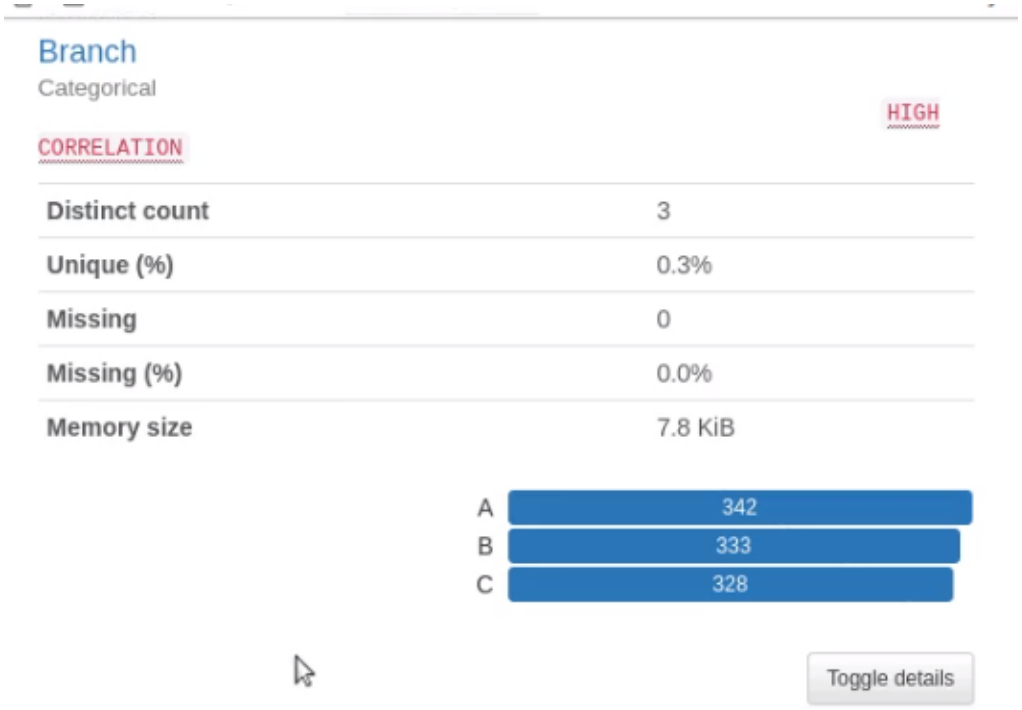
There are no white spaces and so the mean or mode of data has been created and there are no missing values.

Easier Way - PANDA'S PROFILER

BUT - there is an easier way to work through this EDA: PANDA'S PROFILER!!

```
Dataset = pd,read_csv('supermarket_sales.csv')
prof = ProfileReport(dataset)
prof
```





And there is so much more

Quantity

Real number ($\mathbb{R}_{\geq 0}$)

`MISSING`

Distinct count	10
Unique (%)	1.0%
Missing	20
Missing (%)	2.0%
Infinite	0
Infinite (%)	0.0%

Correlation Analysis

np.corrcoef(df['gross income'],df['Rating'])[1][0]

But it is easier to look at a correlation matrix instead. ((sns.heatmap() can be used to visualise the directional magnitudes in a correlation matrix.)

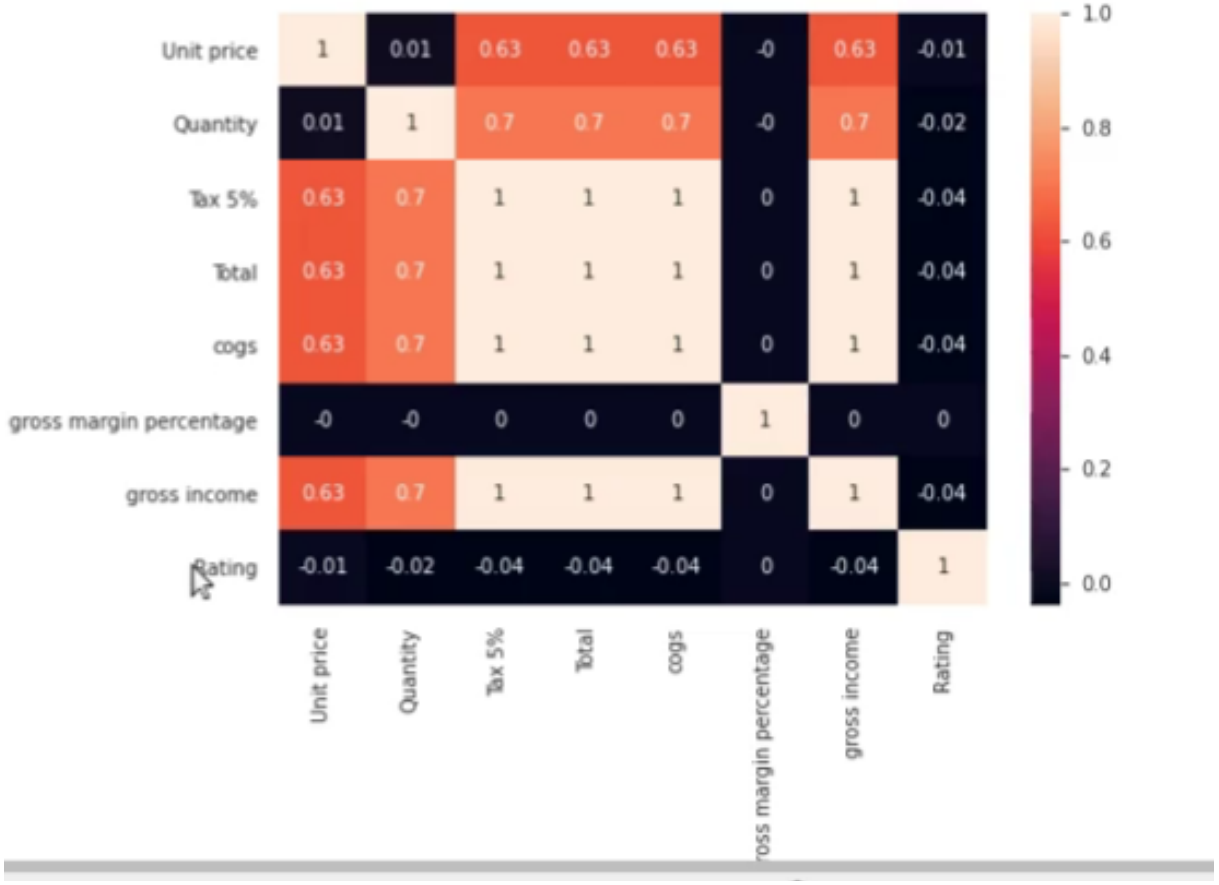
np.round(df.corr(),2)

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
Unit price	1.00	0.01	0.63	0.63	0.63	-0.0	0.63	-0.01
Quantity	0.01	1.00	0.70	0.70	0.70	-0.0	0.70	-0.02
Tax 5%	0.63	0.70	1.00	1.00	1.00	0.0	1.00	-0.04
Total	0.63	0.70	1.00	1.00	1.00	0.0	1.00	-0.04
cogs	0.63	0.70	1.00	1.00	1.00	0.0	1.00	-0.04
gross margin percentage	-0.00	-0.00	0.00	0.00	0.00	1.0	0.00	0.00
gross income	0.63	0.70	1.00	1.00	1.00	0.0	1.00	-0.04
Rating	-0.01	-0.02	-0.04	-0.04	-0.04	0.0	-0.04	1.00

Correlation Analysis as Heatmap

To view this as a heatmap:

sns.heatmap(np.round(df.corr(),2),annot=True)



Customer rating has no relationship with any other data, so their shopping experience has nothing to do with anything else.

I can spot click trends and issues at a glance thanks to the colour-coded nature of heatmaps (beige means the most interaction, black the least).

And this is my Python - Panda EDA complete.

About Michelle Moloney King

Michelle King, based in Co. Tipperary, Ireland, is a skilled professional with a diverse background in teaching and data analytics. With a passion for leveraging data to drive strategic decision-making, Michelle is now seeking to transition from her teaching career into the field of data analytics.

Michelle's website showcases her expertise in SQL, Tableau, Python data analytics, data visualisation, pivot tables, Excel, visual communication, and design. Her strong analytical skills, problem-solving abilities, and customer-focused service make her a valuable asset in the field of data analytics.
Having a degree in Tech and having completed the Google Data Analytics Professional Certificate on Coursera, Michelle has gained in-depth knowledge in data analytics methodologies and tools. Additionally, her Google UX Design Professional Certificate demonstrates her understanding of user-centric design principles.

Now, with a strong foundation in data analytics and a proven track record of leveraging data to drive results, Michelle is eager to transition into a career in data analytics. Her diverse skill set, passion for data-driven insights, and ability to handle multiple projects make her a valuable candidate in the field of data analytics.

Links:

Linkedin
Twitter
Website