```
# Mobile Price Precicton (Capstone Project) - Charlotte Burrows (u32
The Capstone project is to understand and create a predictive model
dataset. The one provide is come from the website Kaggle  centered a
mobile price prediction. This document will go through the steps fro
looks of the data to the full functioning final product at the end e
the code and any related information.
```

Mobile Price Precicton (Capstone Project) - Charlotte Burrows (u3284174)

The Capstone project is to understand and create a predictive model around a dataset. The one provide is come from the website Kaggle centered around a mobile price prediction. This document will go through the steps from the first looks of the data to the full functioning final product at the end explaining the code and any related information.

Section 1

In this section we will be taking an inside look into the mobile phone data set. This is done through these steps:

1. General Code set up.
2. First look at the data
   - 2a. Full data set (15 at a time)
   - 2b. Price graph
   - 2c. Top 5
   - 2d. Bottom 5
   - 2e. Generalized data.
   - 2f. Data type
   - 2g. Understanding no data set
   - 2h. Unique data
   - 2i. General review
   - 2j. assigning data type

General Code Set-Up

The general set up code is common coding practice for good usability and organizational purposes. This allows for libraries downloads and directory paths to be easily set and changed if necessary:

Connection to Google drive:

This Code is importing google.colab to allow access and storage of folders and files to Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Imports for section 1

This section provides all of the imports used within section one, allowing a brief discussion of the code imports and a link to the website to provide additional information if necessary.

```
# Set up imports
from google.colab import drive
import warnings
# Data related imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Pandas

User Guide - https://pandas.pydata.org/docs/user_guide/index.html Pandas allows you to read data from various file formats (CSV, Excel, SQL databases, etc.) into Data Frames.

```
CellData=pd.read_csv
```

An example of calling on the pandas library used to read the csv file containing the cell phone data.

Numpy

User Guide - https://numpy.org/doc/stable/user/index.html#user NumPy provides tools for generating random numbers and sampling from various distributions. For this project it is mainly used for random selection of data.

Matplotlib.pyplot

User Guide - https://matplotlib.org/stable/users/index.html# Matplotlib is a library that allows you to create various types of plots, charts, and graphs. For this project this will be the main library that will be called to create graphs

```
%matplotlib inline
CellData['Price'].hist()
```

This is calling on matplotlib and building a histogram based on the CellData

Checking file path:

This is running a check to make sure the path is correct.

```
%cd /content/drive/MyDrive/UNI/Software.Tec.1/CELLPHONE_CAPSTONE_PROJECT
```

```
/content/drive/MyDrive/UNI/Software.Tec.1/CELLPHONE_CAPSTONE_PROJECT
```

Understanding the directory:

When this code is run it outputs what is within the given directory path.

```
!ls /content/drive/MyDrive/UNI/Software.Tec.1/CELLPHONE_CAPSTONE_PROJECT
```

```
'C:\CSP_CellData\archive\MLData.pkl'   DataForML.pkl         MLData.pkl
 Cellphone.csv                         Final_XGB_Model.pkl   MLR.jpg
```

Warnings

The import helps support warnings, the second sets up a filer to ignore lines

```
import warnings
warnings.filterwarnings('ignore')
```

Initial Data Inspection

Importting dataset

Then goes through and prints the shape (columns and rows of the table) then delates any duplicates and prints the shape again to understand whether there were any duplicate rows within the data.

1. reads the CSV file which is where the CellData set is stored
2. Then goes through and prints the shape (columns and rows of the table)
3. then delates any duplicates
4. prints the shape again a. this will help identify any duplicate rows
5. rounds all data to 3 decimal places. a. simplification of data noting all data is less than 3 decimal places anyway.

6. Printing the first 15 rows of the table

```
CellData=pd.read_csv('/content/drive/MyDrive/UNI/Software.Tec.1/CELLPHONE_CAPSTONE_PROJECT/Cellphone.csv', encoding='latin')
print('Shape before deleting duplicate values:', CellData.shape)
CellData=CellData.drop_duplicates()
print('Shape After deleting duplicate values:', CellData.shape)
CellData.round(3)
CellData.head(15)
```

```
Shape before deleting duplicate values: (161, 14)
Shape After deleting duplicate values: (161, 14)
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	Re
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	
4	880	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	
5	947	2137	12	150.0	5.5	401	4	2.30	16.0	2.000	
6	774	1238	13	134.1	4.0	233	2	1.20	8.0	1.000	
7	947	2137	13	150.0	5.5	401	4	2.30	16.0	2.000	
8	99	1315	14	118.5	4.0	233	2	1.30	4.0	0.512	
9	1103	2580	15	145.0	5.1	432	4	2.50	16.0	2.000	
10	289	2438	16	162.0	5.3	277	8	1.50	32.0	4.000	
11	605	2006	16	161.0	5.5	200	8	1.40	4.0	1.000	
12	622	2174	16	140.0	5.0	294	4	1.30	16.0	1.000	
13	1058	2744	16	174.0	5.6	524	4	2.70	32.0	3.000	

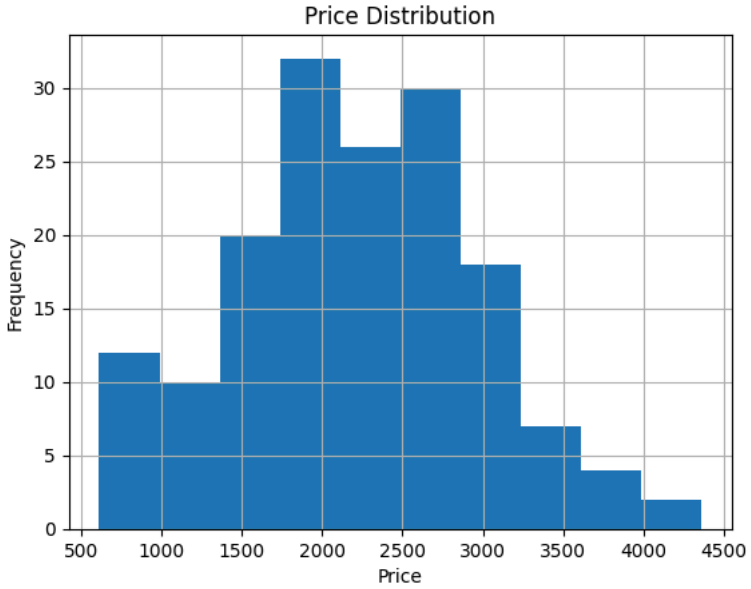
Next steps: Generate code with `CellData` View recommended plots

Price Histogram

This is the creation of the price histogram and the labeling of the axis

```
fig, ax = plt.subplots(1, 1)
ax.set_title("Price Distribution")
ax.set_xlabel('Price')
ax.set_ylabel('Frequency')
%matplotlib inline
CellData['Price'].hist()
```

```
<Axes: title={'center': 'Price Distribution'}, xlabel='Price', ylabel='Frequency'>
```



Top 5

This outputs the top 5 of the data set

```
CellData.head()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	Rea
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	1
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	1
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	

Next steps: Generate code with CellData | View recommended plots

Bottom 5

This outputs the bottom 5 of the data set

```
CellData.tail()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	Re
156	1206	3551	4638	178.0	5.46	538	4	1.875	128.0	6.0	
157	1296	3211	8016	170.0	5.50	534	4	1.975	128.0	6.0	
158	856	3260	8809	150.0	5.50	401	8	2.200	64.0	4.0	
159	1296	3211	8946	170.0	5.50	534	4	1.975	128.0	6.0	

Generalized Data

Each of the following columns show:

- Count: total number of observations
- Mean: average value of the data
- Standard deviation: the dispersion of data
- Minimum: smallest valued data
- 25%: The value below which 25% of the data falls

- 50%: the middle value (the median)
- 75%: The value below which 75% of the data falls
- Maximum: the largest valued data This give insight into the range and variability of data.

```
CellData.describe()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cp
count	161.000000	161.000000	161.000000	161.000000	161.000000	161.000000	161.(
mean	675.559006	2215.596273	621.465839	170.426087	5.209938	335.055901	4.8
std	410.851583	768.187171	1546.618517	92.888612	1.509953	134.826659	2.4
min	10.000000	614.000000	10.000000	66.000000	1.400000	121.000000	0.(
25%	237.000000	1734.000000	37.000000	134.100000	4.800000	233.000000	4.(
50%	774.000000	2258.000000	106.000000	153.000000	5.150000	294.000000	4.(
75%	1026.000000	2744.000000	382.000000	170.000000	5.500000	428.000000	8.(

Data type

This section is reading the data from the dataset and outputting various types of data e.g. (int64 & float64). As well as the memory usage

```
CellData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161 entries, 0 to 160
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Product_id    161 non-null    int64  
 1   Price         161 non-null    int64  
 2   Sale          161 non-null    int64  
 3   weight        161 non-null    float64
 4   resoloution   161 non-null    float64
 5   ppi           161 non-null    int64  
 6   cpu core      161 non-null    int64  
 7   cpu freq      161 non-null    float64
 8   internal mem  161 non-null    float64
 9   ram           161 non-null    float64
 10  RearCam       161 non-null    float64
 11  Front_Cam     161 non-null    float64
 12  battery       161 non-null    int64  
 13  thickness     161 non-null    float64
dtypes: float64(8), int64(6)
memory usage: 17.7 KB
```

Understanding no Data set

This is understanding if there are any null data that could skew with the data.

```
CellData.isnull().sum()
```

```
Product_id      0
Price           0
Sale            0
weight          0
resoloution     0
ppi             0
cpu core        0
cpu freq        0
internal mem    0
ram             0
RearCam         0
Front_Cam       0
battery         0
thickness       0
dtype: int64
```

Unique Data

Looking at any unique values within each of the column

```
CellData.nunique()
```

```
Product_id       83
Price            81
Sale            125
weight           62
resoloution      24
ppi              45
cpu core          6
cpu freq         28
internal mem     10
ram              12
RearCam          18
Front_Cam        15
battery          55
thickness        49
dtype: int64
```

General Review

Looking at this out there were no duplicate columns or rows. There are 161 row & 14 columns within our given data set. The Columns are as follows:

- Product Id - ID of each cellphone
- Price - Price of each cellphone
- Sale - Sales number
- Weight - Weight of each cellphone
- Resolution - Resolution of each cellphone
- PPI - Phone Pixel Density
- CPU core - type of CPU core in each cellphone
- CPU frequency - CPU Frequency in each cellphone in Hertz (Hz)
- internal memory - Internal memory of each cellphone both read-only memory (ROM) and random-access memory (RAM)
- Ram - Random-Access Memory of each cellphone
- Rear Camera - The camera is located on the back of a device.
- Front Camera - The camera is located on the front of a device.
- battery - The power source for electronic devices
- thickness - The measurement of how thin or thick a device.

Section 2

---

what will be covered in the section is:

- Imports for section 2
- Categorical Data
    - Histogram
    - Scatterplot matrix/pair plot
    - Bar Chart
    - Findings
- Continuous Data
    - Histogram
    - scatterplot matrix/pair plot
- Comaprison
    - continuois data vs catagorical data box plot
    - everything vs price
- Outerlier
    - identifying the outliers
    - removal & Comparison of outlires
- ANOVA Test
- Data Preperation For Machine Learning

Imports for section 2

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
#For AV Test
from scipy.stats import f_oneway
```

Categorical Data

This section contains all of the categorical data from the data set:

- CPU Core
- Rear Camara
- Frount Camara
- Resolutoution

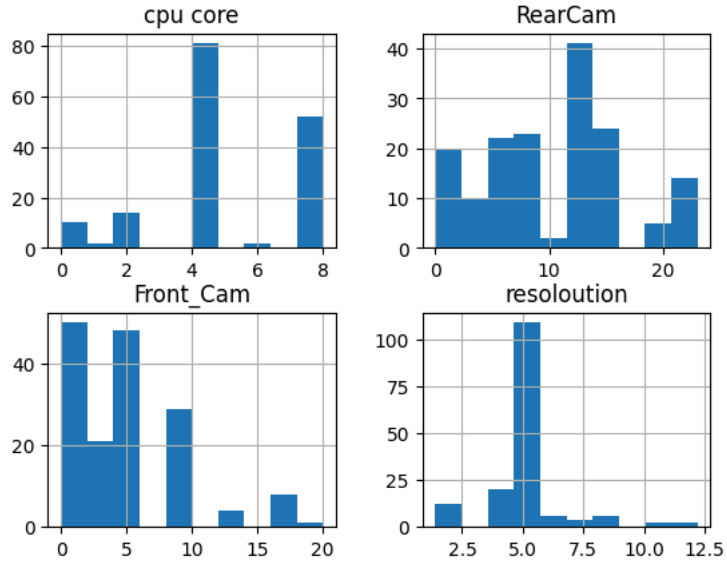
This data will be turned into different graphs to allow for a better understanding and comparison.

Histogram

Histograms for this data helps in identifying where most of the data points fall, the range of the data, and can provide insights into the central tendency, variability, and presence of any outliers in each feature.

```
CellData[['cpu core', 'RearCam', 'Front_Cam', 'resoloution']].hist()
```

```
array([[<Axes: title={'center': 'cpu core'}>,
        <Axes: title={'center': 'RearCam'}>],
       [<Axes: title={'center': 'Front_Cam'}>,
        <Axes: title={'center': 'resoloution'}>]], dtype=object)
```

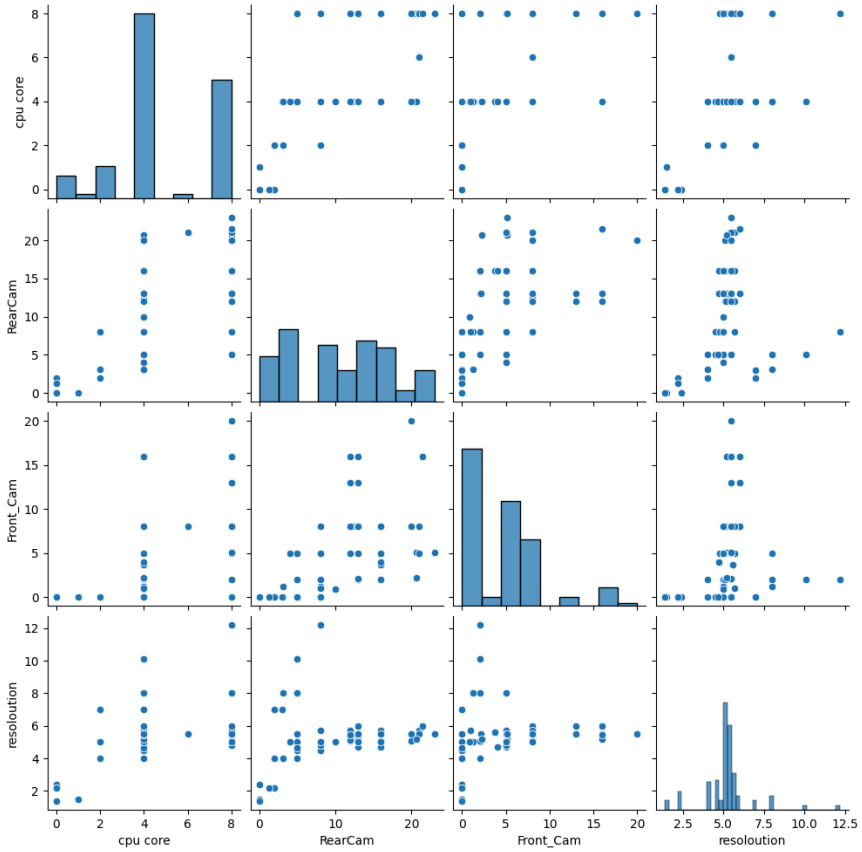


Scatterplot Matrix/Pair Plot

This visualization is a matrix of scatterplots that allows you to see the relationship between each pair of variables in a dataset. It's an effective tool for a quick overview of how each variable relates to the others. This also encapsulates the histogram data from above.

```
sns.pairplot(CellData[['cpu core', 'RearCam', 'Front_Cam', 'resoloution']])
```

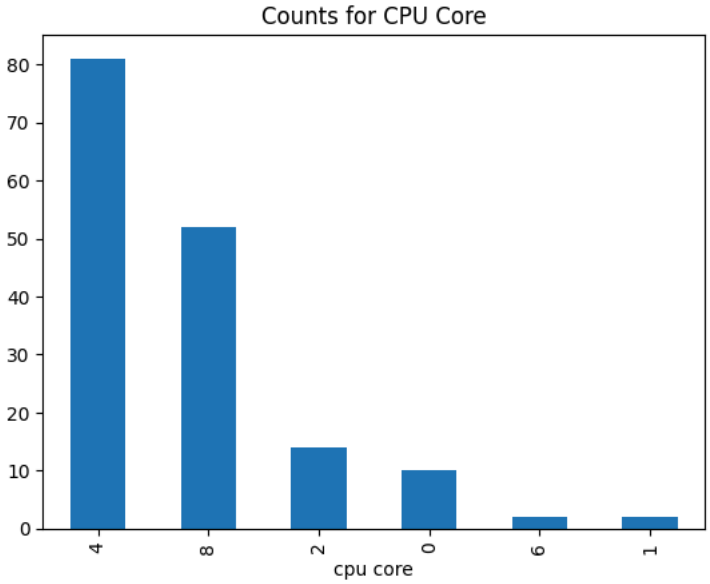
```
<seaborn.axisgrid.PairGrid at 0x7eb8bdd62e90>
```

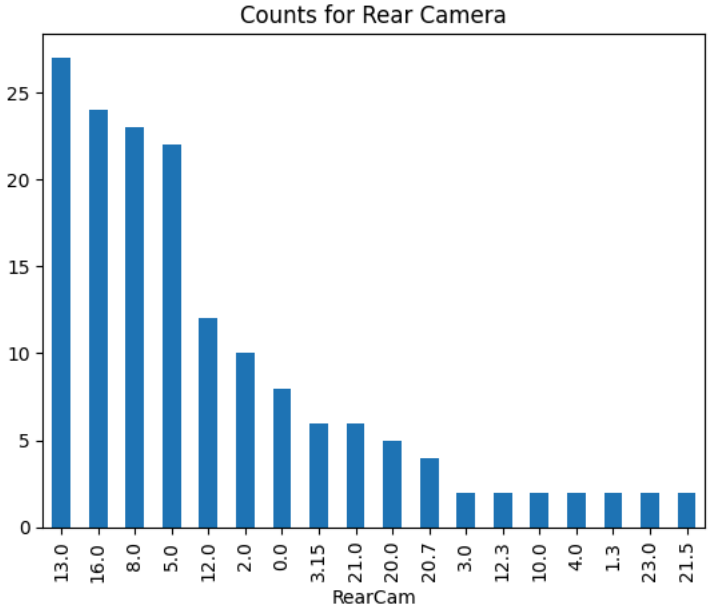


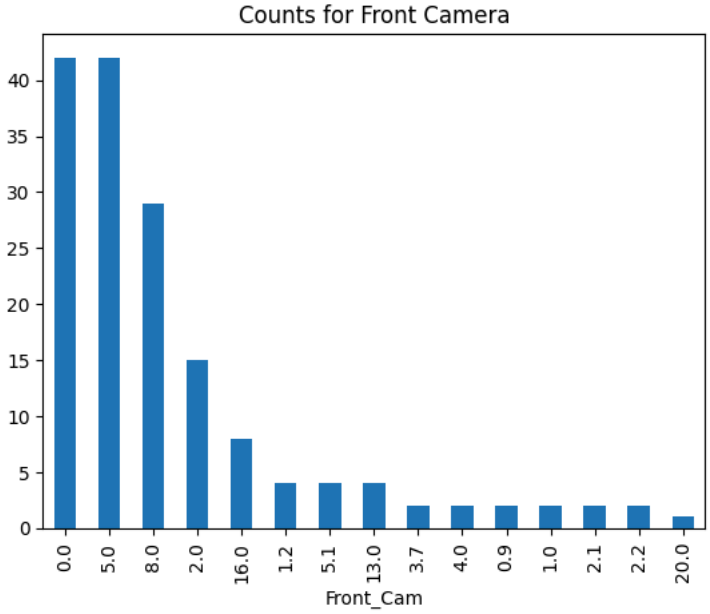
Bar Chat

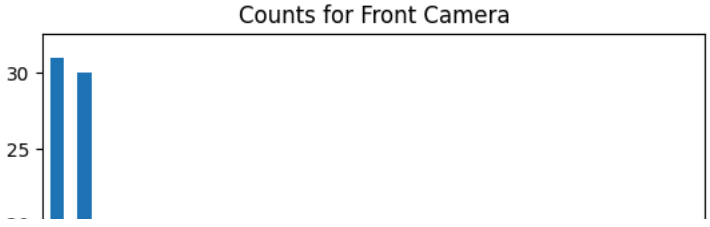
Double-click (or enter) to edit

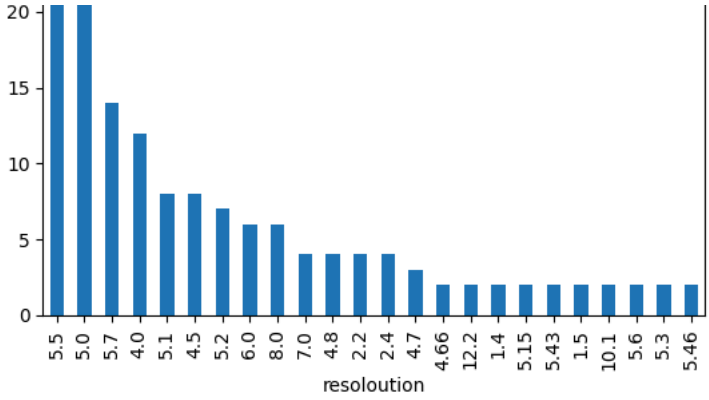
```
import matplotlib.pyplot as plt

# Create a bar chart for 'cpu core' counts
plt.figure()
CellData['cpu core'].value_counts().plot(kind='bar', title='Counts for CPU Core')
plt.show()

# Create a bar chart for 'RearCam' counts
plt.figure()
CellData['RearCam'].value_counts().plot(kind='bar', title='Counts for Rear Camera')
plt.show()

# Create a bar chart for 'Front_Cam' counts
plt.figure()
CellData['Front_Cam'].value_counts().plot(kind='bar', title='Counts for Front Camera')
plt.show()

# Create a bar chart for 'resolution' counts
plt.figure()
CellData['resoloution'].value_counts().plot(kind='bar', title='Counts for Front Camera')

plt.show()
```


Counts for CPU Core
80
70
60
50
40
30
20
10
0
4
8
2
0
6
1
cpu core


Counts for Rear Camera
25
20
15
10
5
0
13.0
16.0
8.0
5.0
12.0
2.0
0.0
3.15
21.0
20.0
20.7
3.0
12.3
10.0
4.0
1.3
23.0
21.5
RearCam


Counts for Front Camera
40
35
30
25
20
15
10
5
0
0.0
5.0
8.0
2.0
16.0
1.2
5.1
13.0
3.7
4.0
0.9
1.0
2.1
2.2
20.0
Front_Cam


Counts for Front Camera
30
25



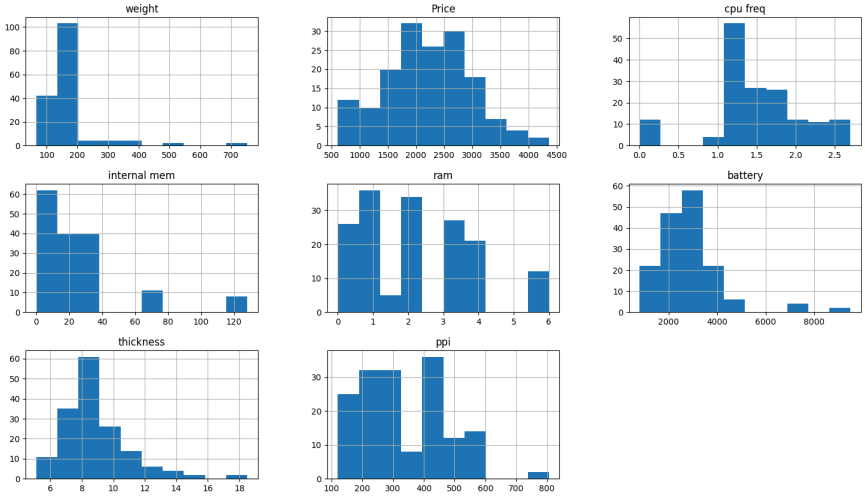
Continuous Data

Histogram

Histograms for this data helps in identifying where most of the data points fall, the range of the data, and can provide insights into the central tendency, variability, and presence of any outliers in each feature.

```
CellData.hist(["weight","Price","cpu freq","internal mem","ram","battery","thickness","ppi"], figsize=(18,10))
```

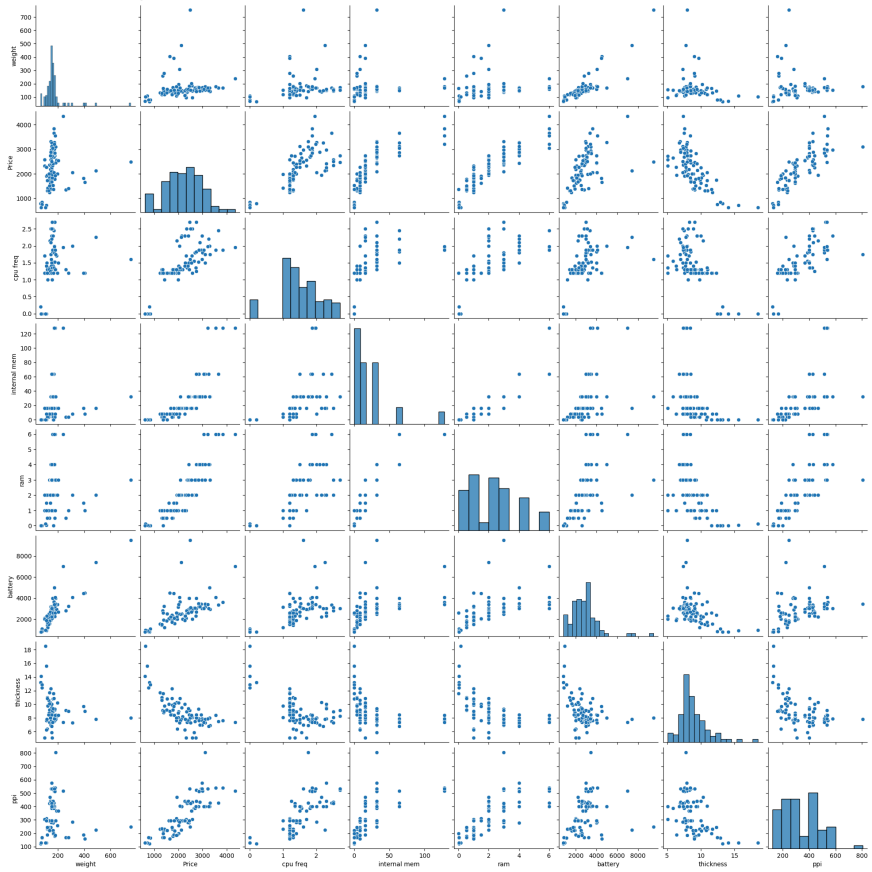
```
array([[<Axes: title={'center': 'weight'}>,
        <Axes: title={'center': 'Price'}>,
        <Axes: title={'center': 'cpu freq'}>],
       [<Axes: title={'center': 'internal mem'}>,
        <Axes: title={'center': 'ram'}>,
        <Axes: title={'center': 'battery'}>],
       [<Axes: title={'center': 'thickness'}>,
        <Axes: title={'center': 'ppi'}>, <Axes: >]], dtype=object)
```



Scatterplot Matrix/Pair Plot

```
sns.pairplot(CellData[["weight","Price","cpu freq","internal mem","ram","battery","thickness","ppi"]])
```

```
<seaborn.axisgrid.PairGrid at 0x7eb8bcffeaa0>
```



Comparison

Continuous Data vs Catagorical data table

The data very simlar to a heat map shows how each of the different coloumbs interact with each other in relation to the mean.

```
# Calculating correlation matrix
ContinuousCols=["weight","Price","cpu freq","internal mem","ram","battery","thickness","ppi"]

# Creating the correlation matrix
CorrelationData=CellData[ContinuousCols].corr()
CorrelationData
```

	weight	Price	cpu freq	internal mem	ram	battery	thickness	
weight	1.000000	0.144555	0.222730	0.098849	0.149283	0.833783	-0.185262	-0.0
Price	0.144555	1.000000	0.727383	0.776738	0.896915	0.559946	-0.716773	0.8
cpu freq	0.222730	0.727383	1.000000	0.441400	0.633547	0.473137	-0.614458	0.7
internal mem	0.098849	0.776738	0.441400	1.000000	0.875354	0.461506	-0.367412	0.6
ram	0.149283	0.896915	0.633547	0.875354	1.000000	0.541001	-0.521074	0.7
battery	0.833783	0.559946	0.473137	0.461506	0.541001	1.000000	-0.412682	0.3
thickness	-0.185262	-0.716773	-0.614458	-0.367412	-0.521074	-0.412682	1.000000	-0.4

Next steps: Generate code with `CorrelationData` View recommended plots

Price Correlation

This data can be useful in understanding how different features of a product relate to its price.

```
CorrelationData['Price'][abs(CorrelationData['Price']) > 0.5 ]
```

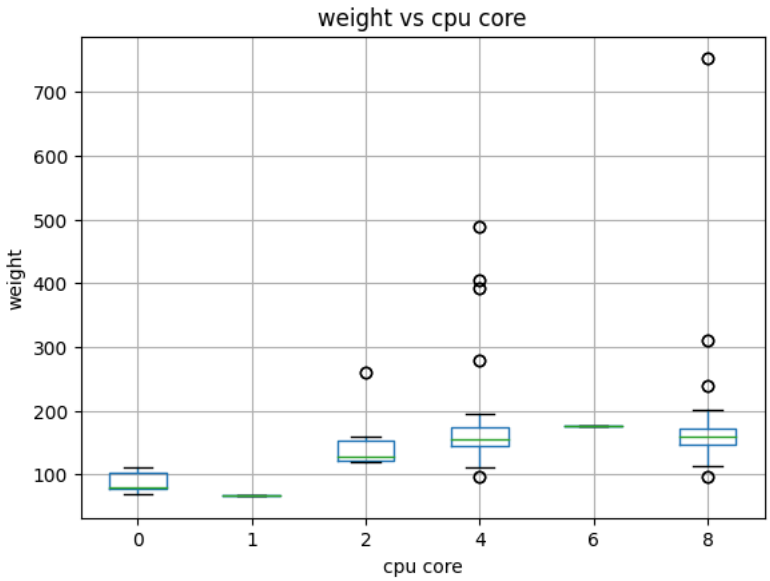
```
Price          1.000000
cpu freq       0.727383
internal mem   0.776738
ram            0.896915
battery        0.559946
thickness     -0.716773
ppi            0.817614
Name: Price, dtype: float64
```

Continuous Data vs Catagorical data Box plot

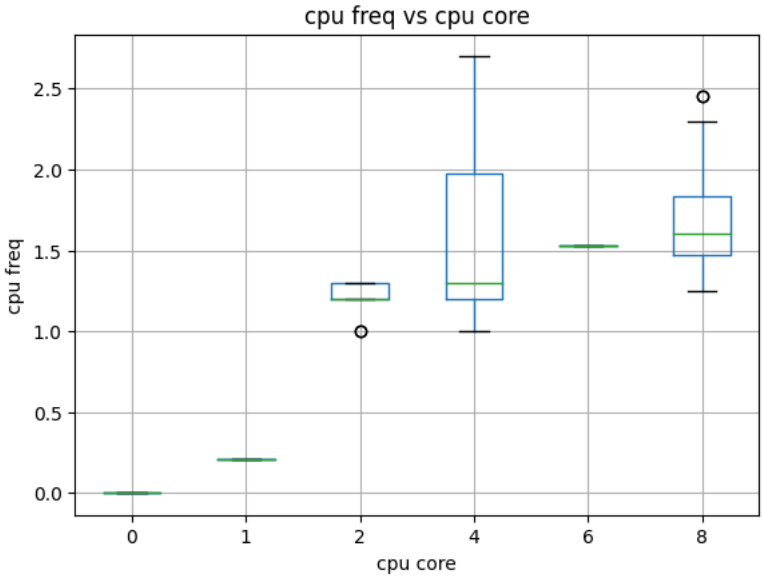
the bollow show how the different types of contiuous data relate to the Catagorical data

```
import matplotlib.pyplot as plt

# Assuming 'CellData' is your DataFrame and it's already loaded with data.

# List of continuous variables
ContinuousVars = ["weight", "cpu freq", "internal mem", "ram", "battery", "thickness", "ppi"]

# List of categorical variables
CategoricalVars = ['cpu core', 'RearCam', 'Front_Cam', 'resoloution']

# Plotting boxplot for each continuous variable vs each categorical variable
for categorical in CategoricalVars:
    for continuous in ContinuousVars:
        plt.figure(figsize=(10,5))
        CellData.boxplot(column=continuous, by=categorical)
        plt.title(f'{continuous} vs {categorical}')
        plt.suptitle('')  # Suppress the default title to avoid overlapping with the title we set
        plt.xlabel(categorical)
        plt.ylabel(continuous)
        plt.show()
```

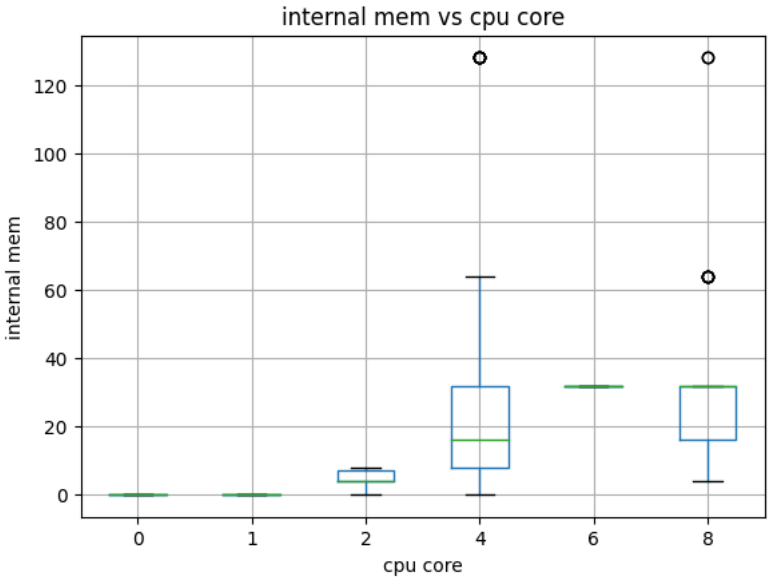
<Figure size 1000x500 with 0 Axes>



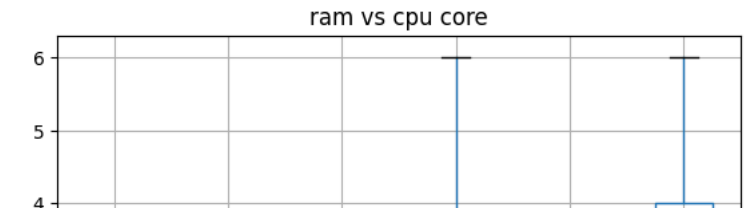
<Figure size 1000x500 with 0 Axes>

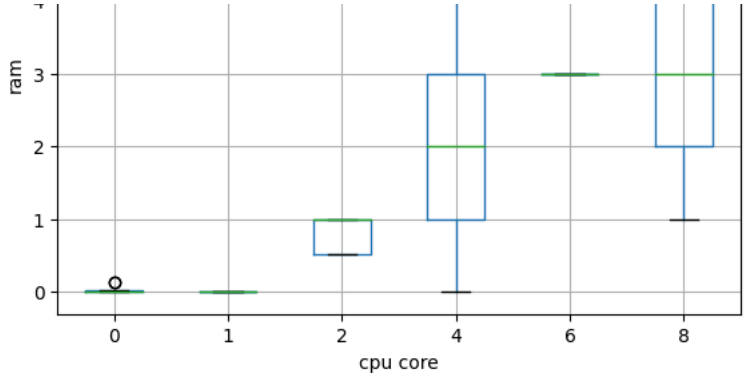


<Figure size 1000x500 with 0 Axes>

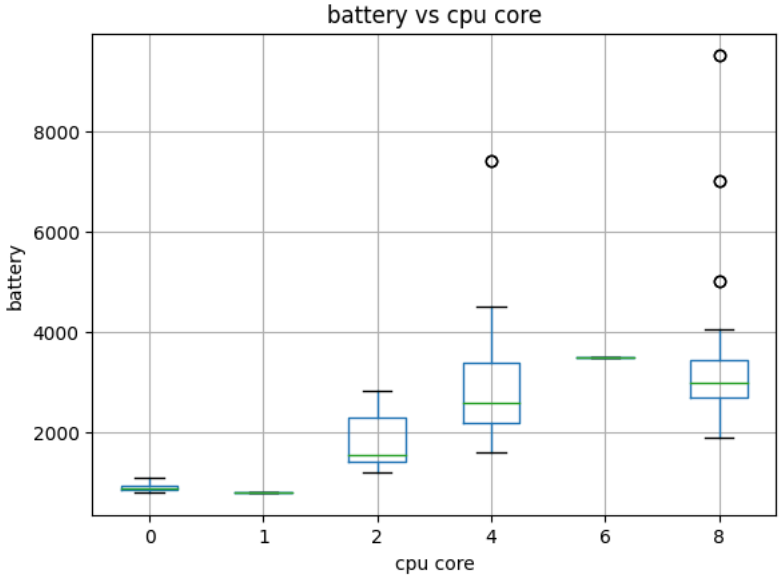


<Figure size 1000x500 with 0 Axes>

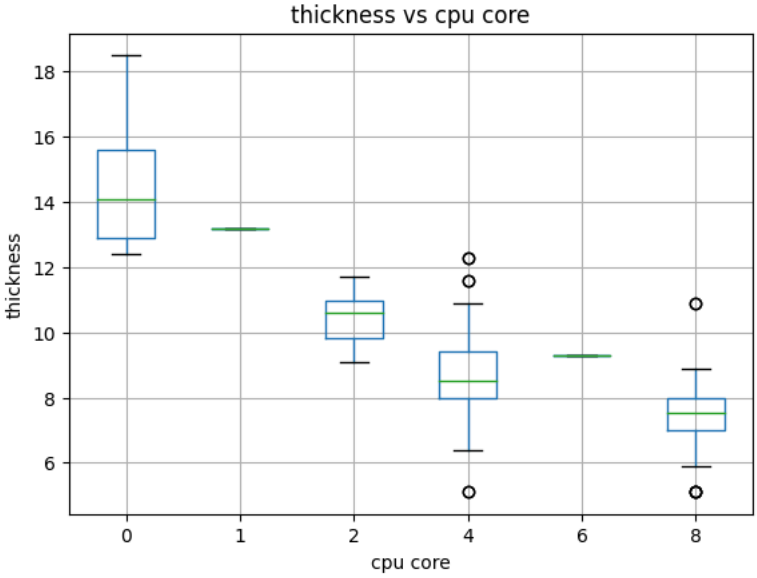




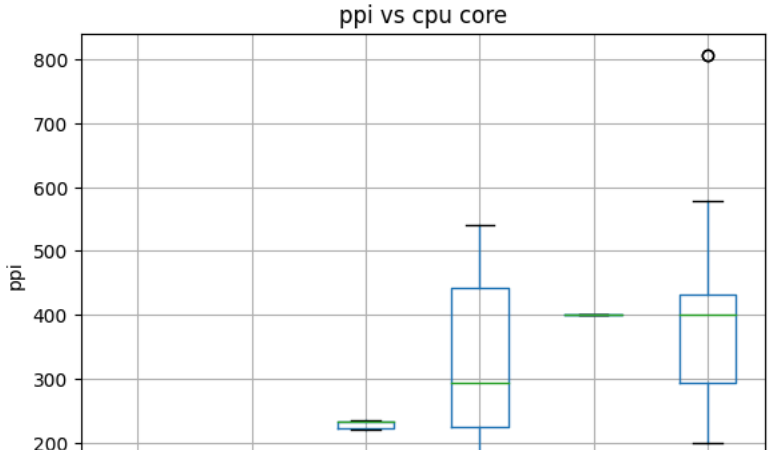
<Figure size 1000x500 with 0 Axes>

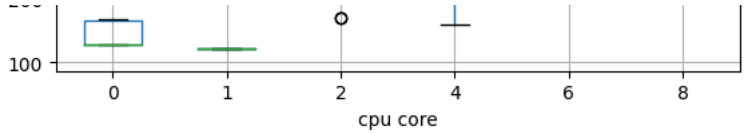


<Figure size 1000x500 with 0 Axes>

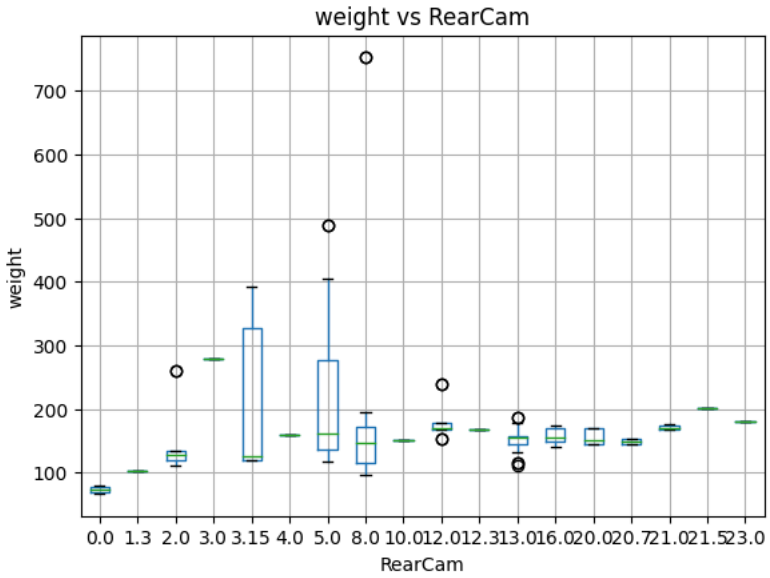


<Figure size 1000x500 with 0 Axes>

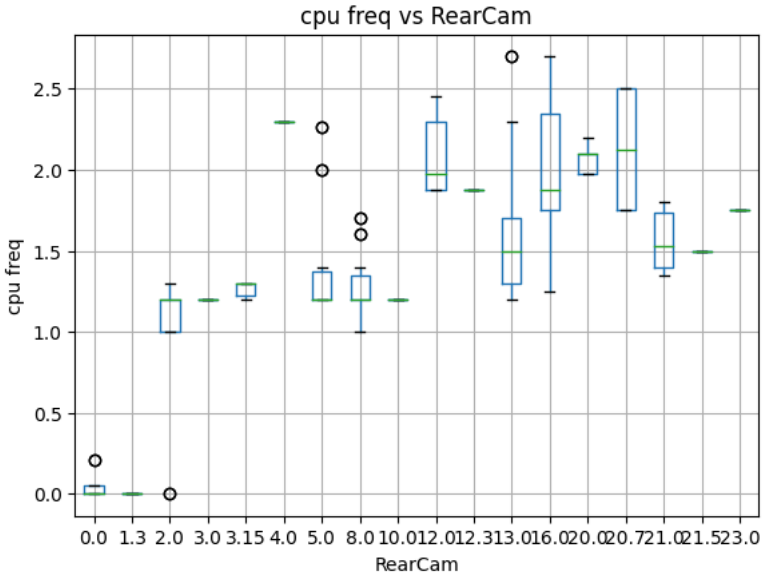




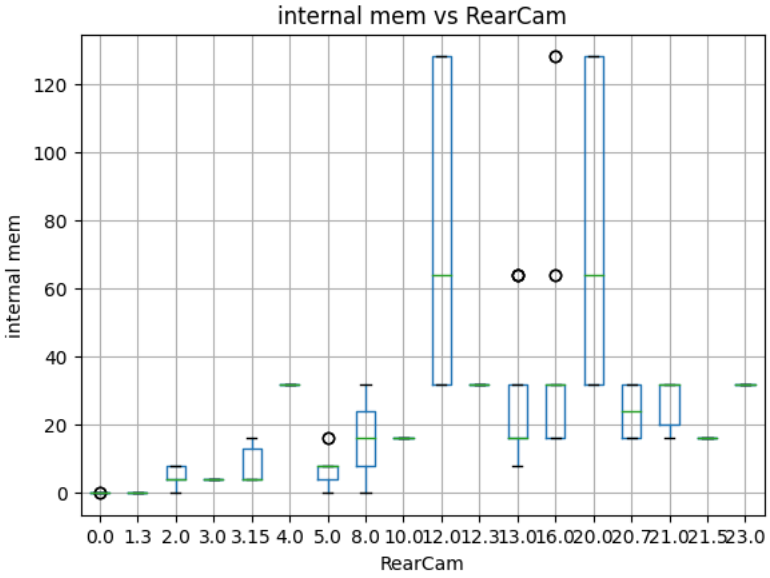
<Figure size 1000x500 with 0 Axes>



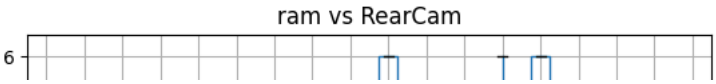
<Figure size 1000x500 with 0 Axes>

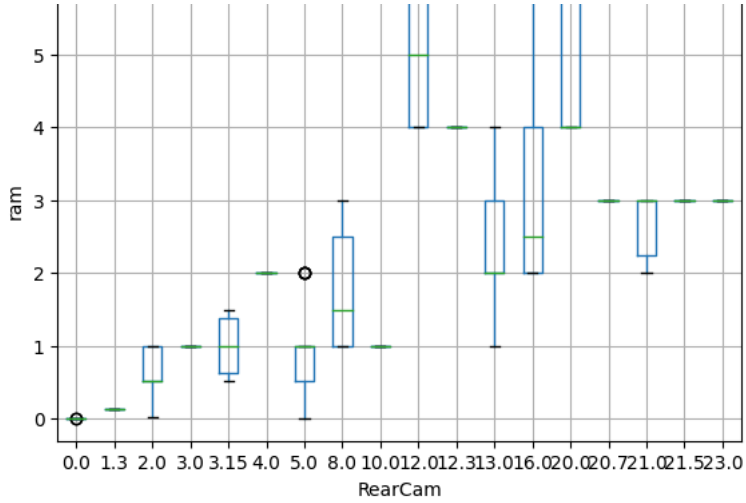


<Figure size 1000x500 with 0 Axes>

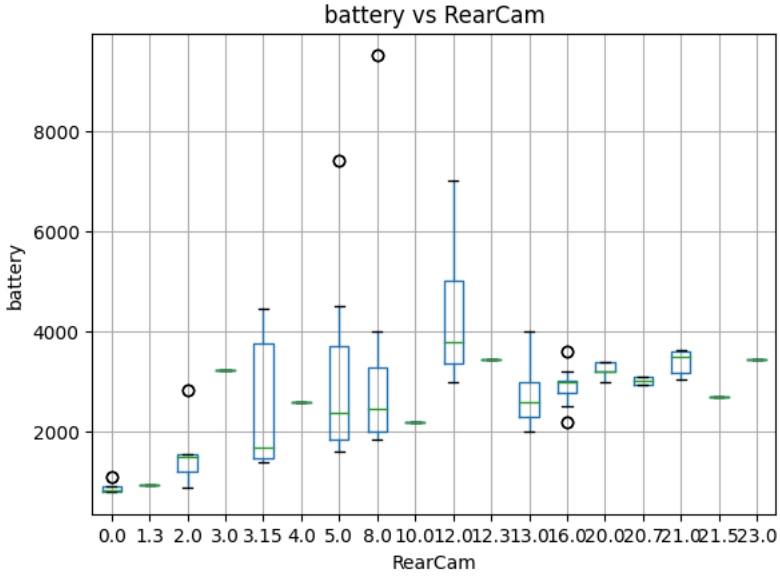


<Figure size 1000x500 with 0 Axes>

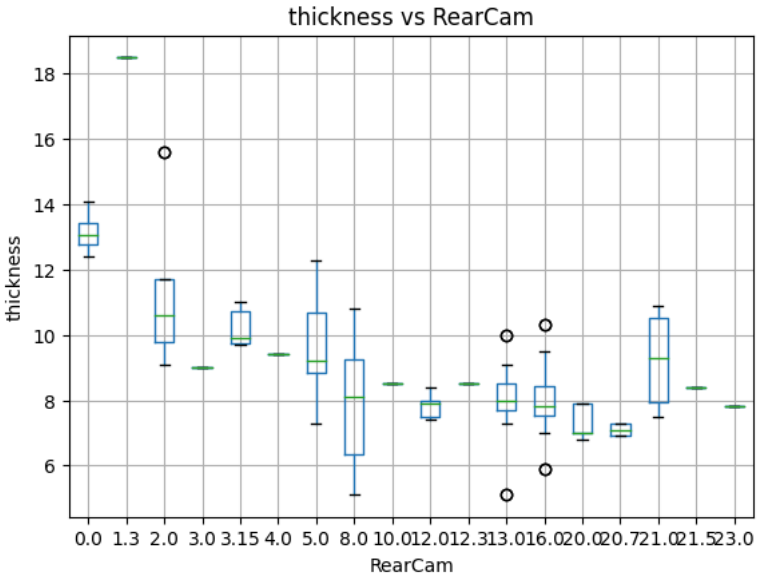




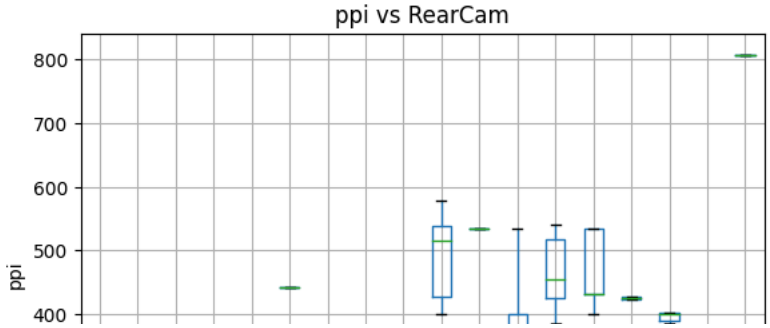
<Figure size 1000x500 with 0 Axes>

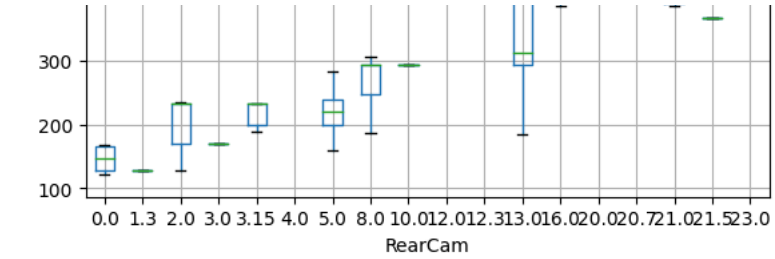


<Figure size 1000x500 with 0 Axes>

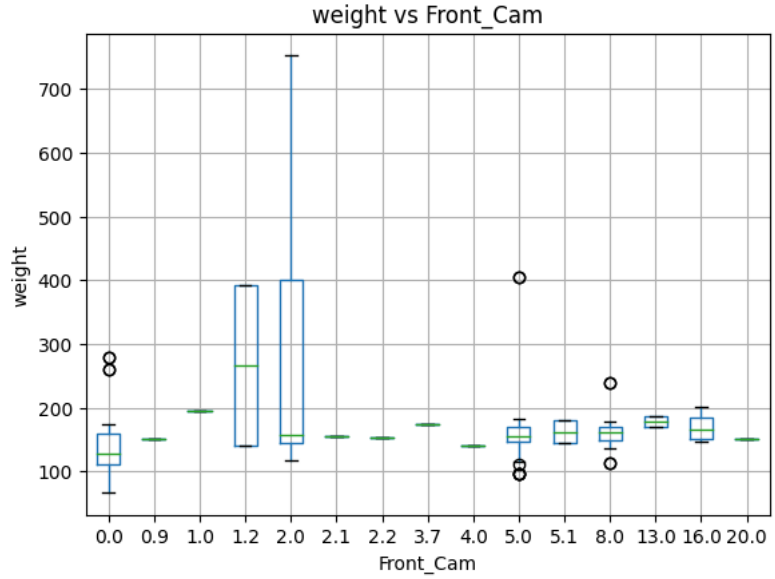


<Figure size 1000x500 with 0 Axes>

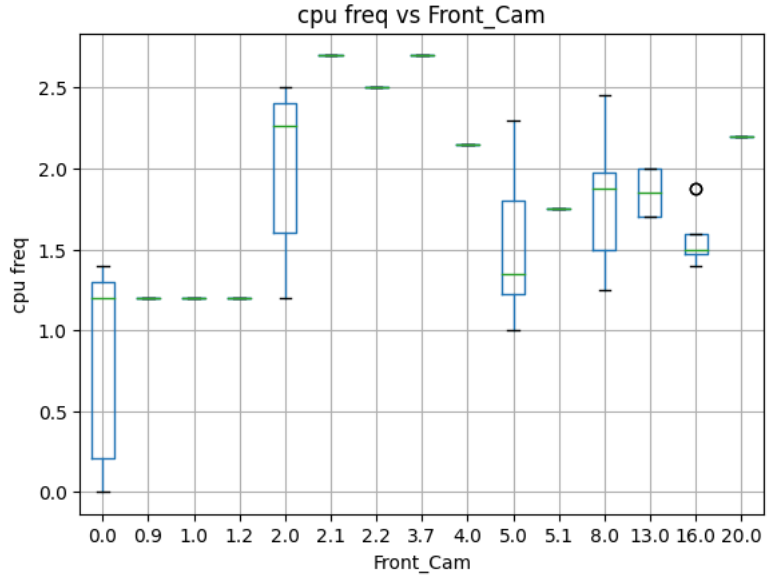




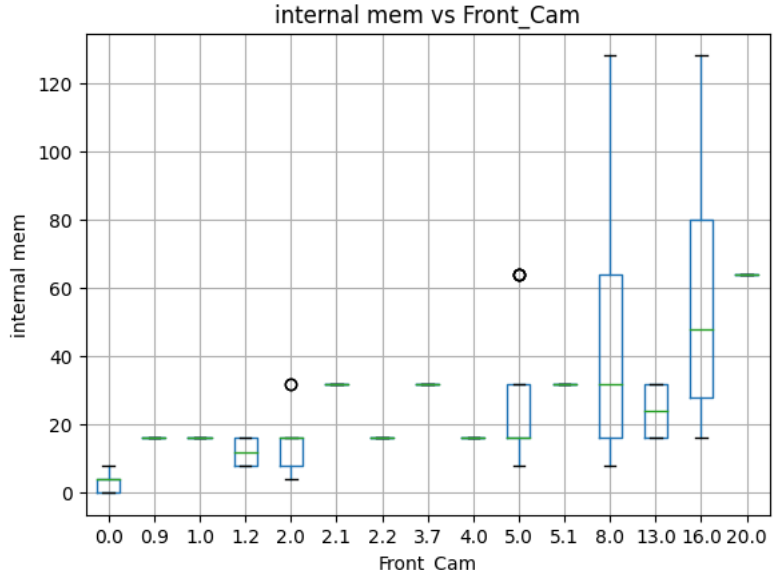
<Figure size 1000x500 with 0 Axes>



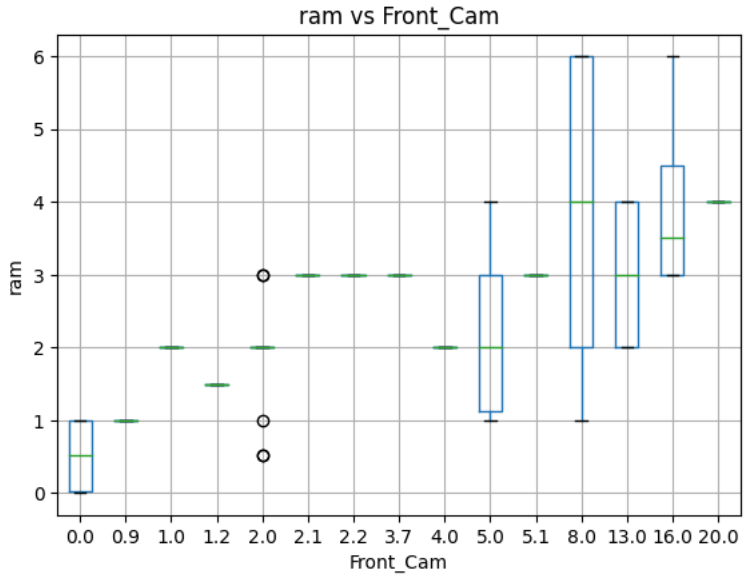
<Figure size 1000x500 with 0 Axes>



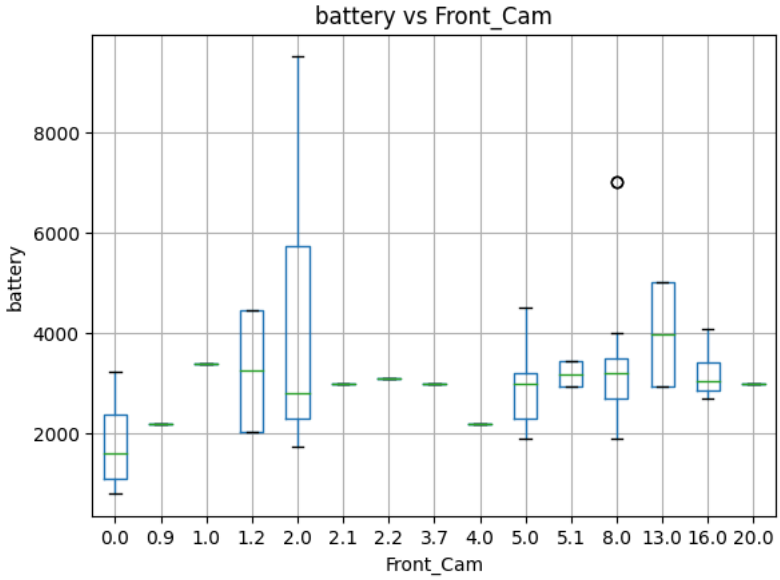
<Figure size 1000x500 with 0 Axes>



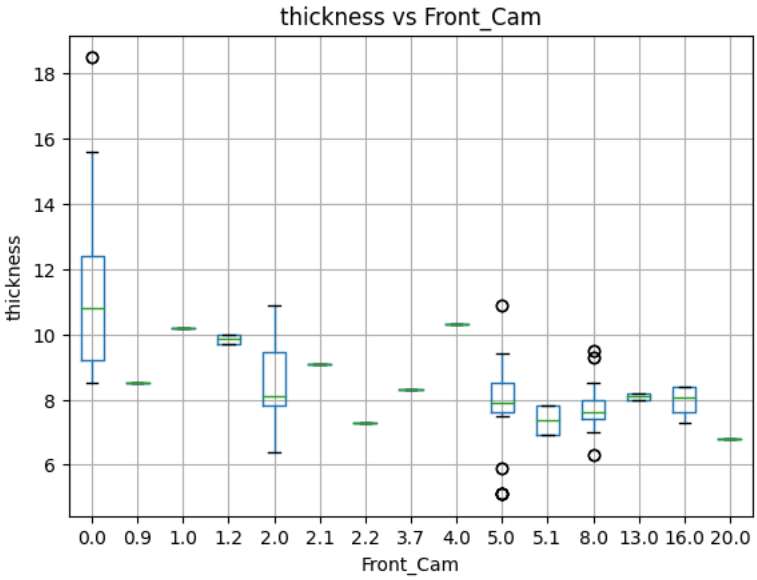
<Figure size 1000x500 with 0 Axes>



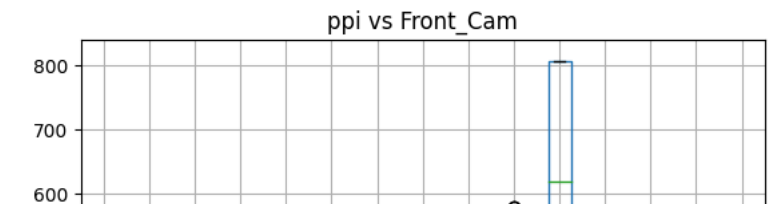
<Figure size 1000x500 with 0 Axes>

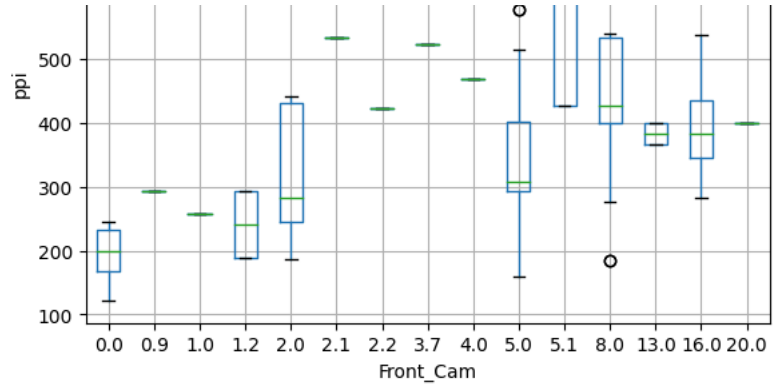


<Figure size 1000x500 with 0 Axes>

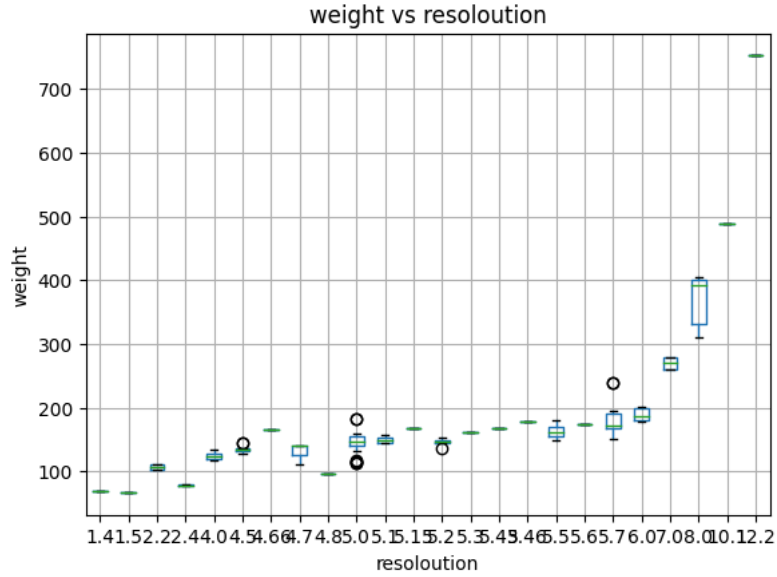


<Figure size 1000x500 with 0 Axes>

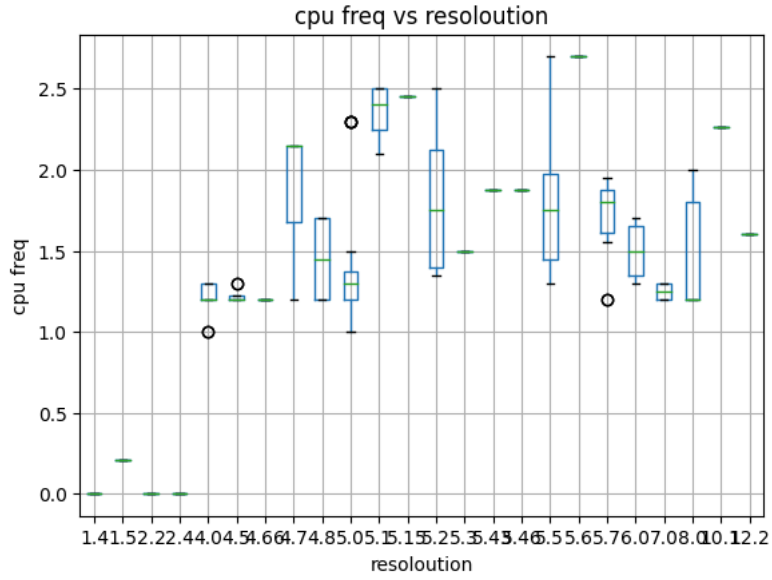




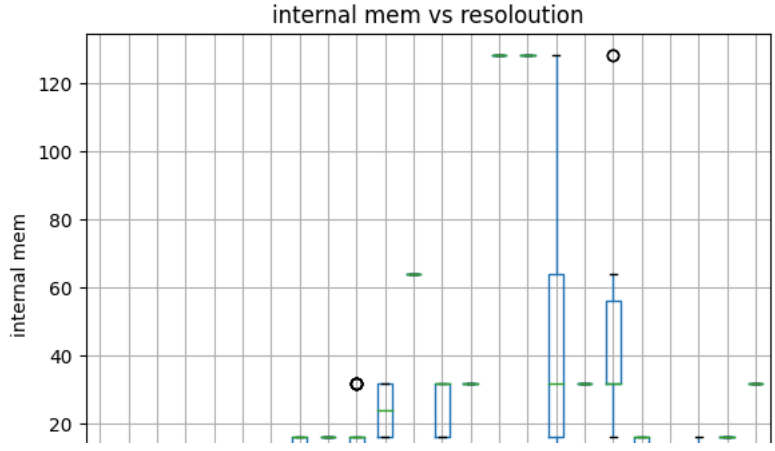

```
<Figure size 1000x500 with 0 Axes>
```

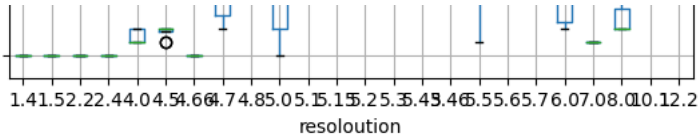



```
<Figure size 1000x500 with 0 Axes>
```

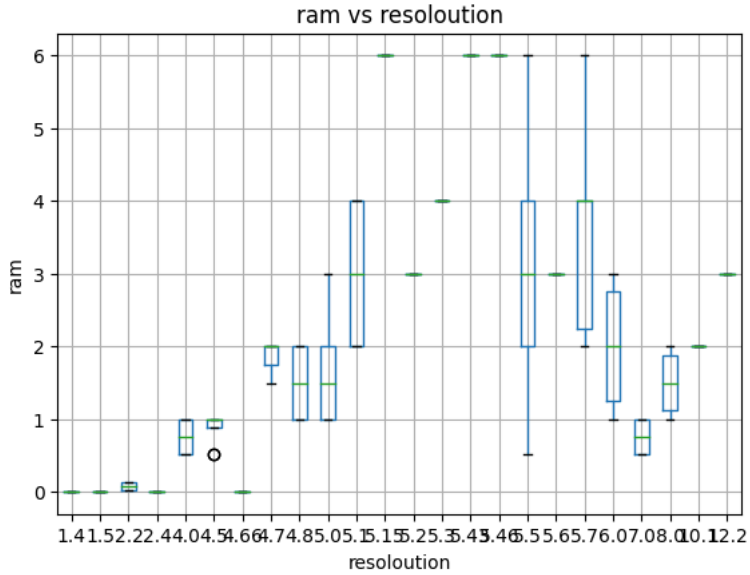



```
<Figure size 1000x500 with 0 Axes>
```

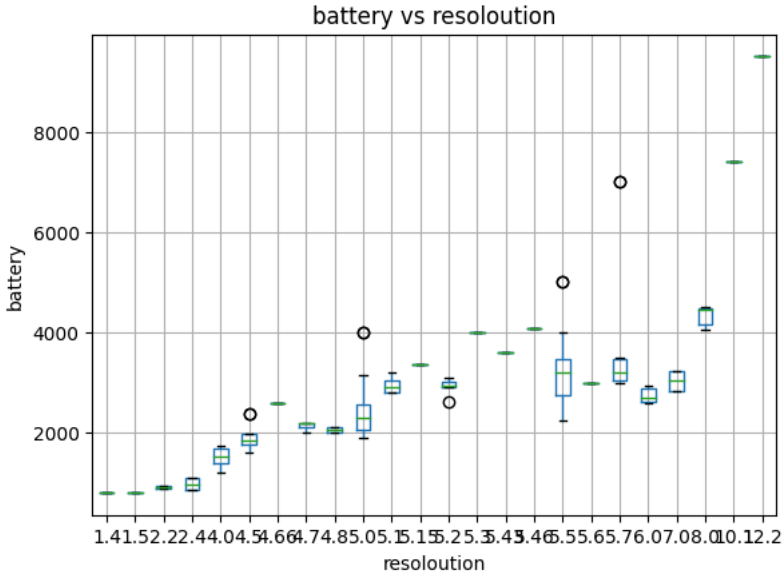




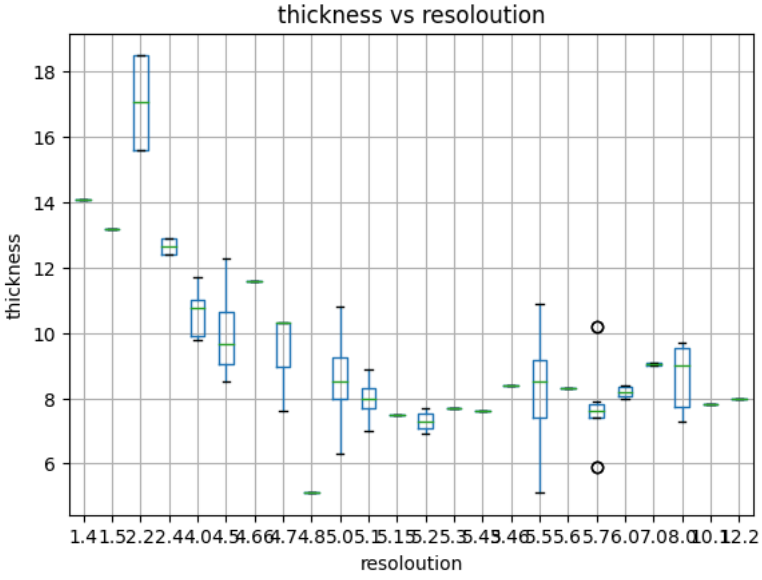
<Figure size 1000x500 with 0 Axes>



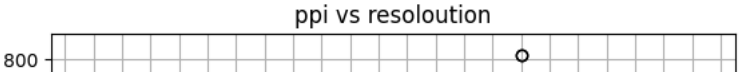
<Figure size 1000x500 with 0 Axes>

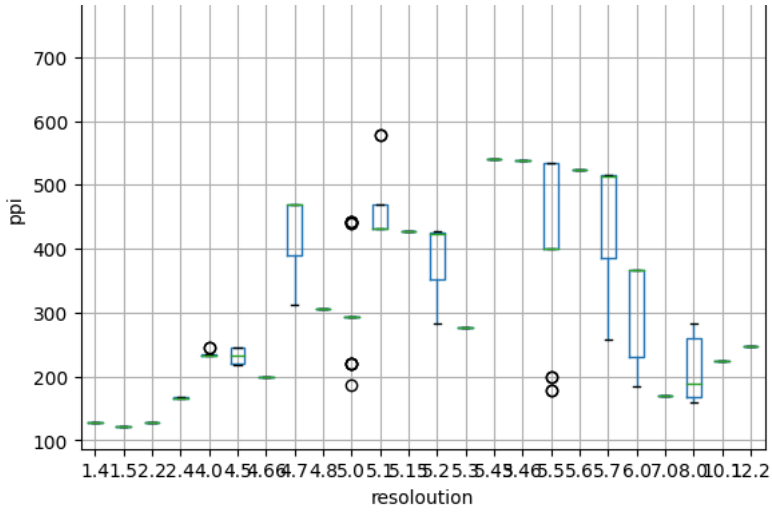


<Figure size 1000x500 with 0 Axes>



<Figure size 1000x500 with 0 Axes>

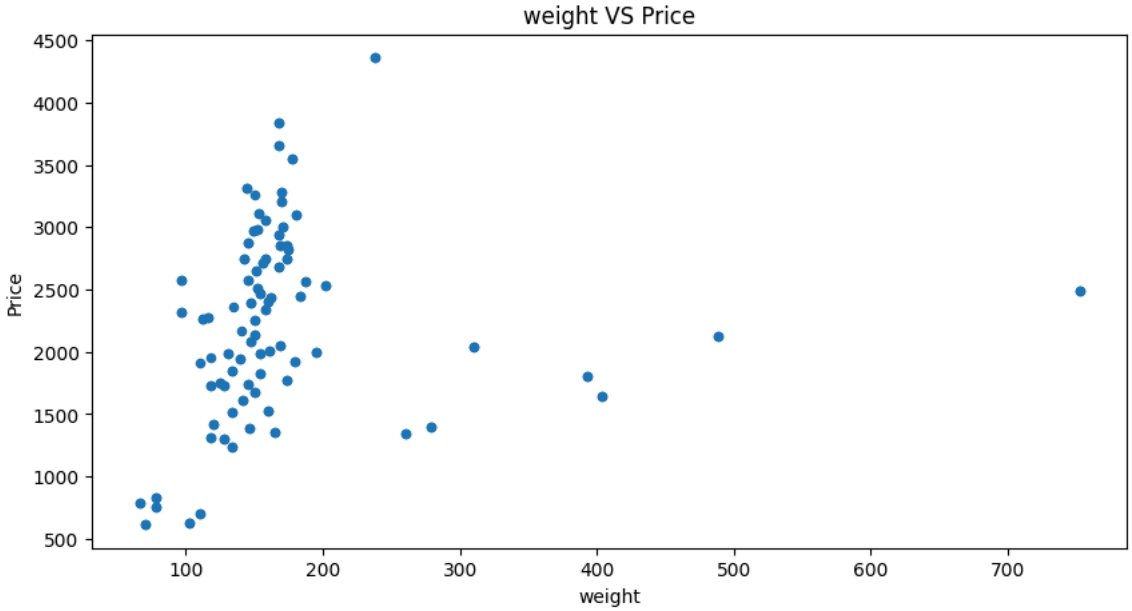


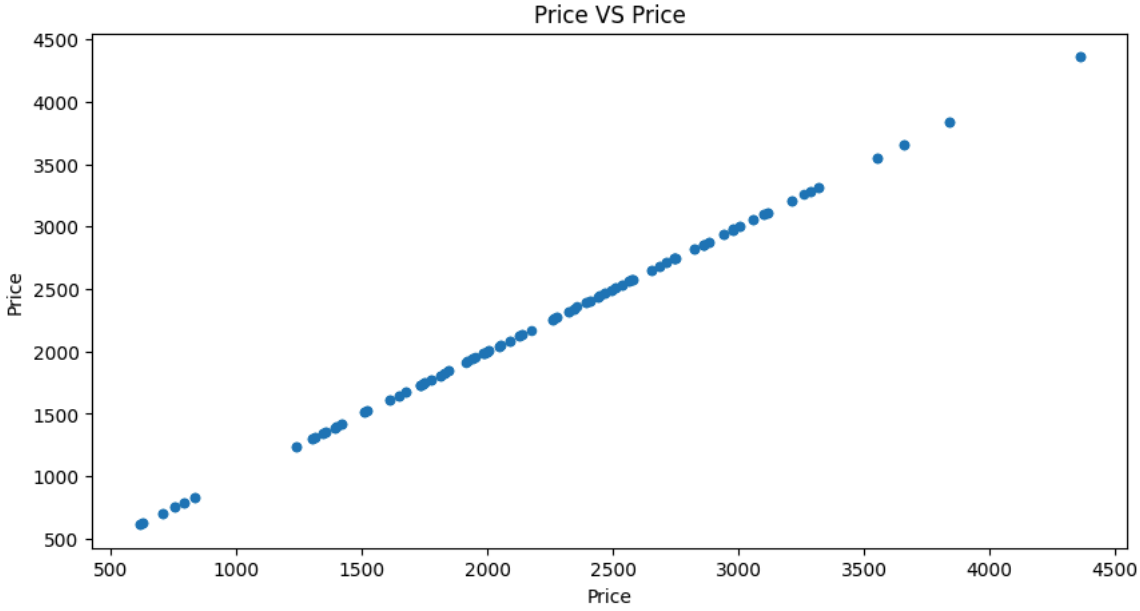


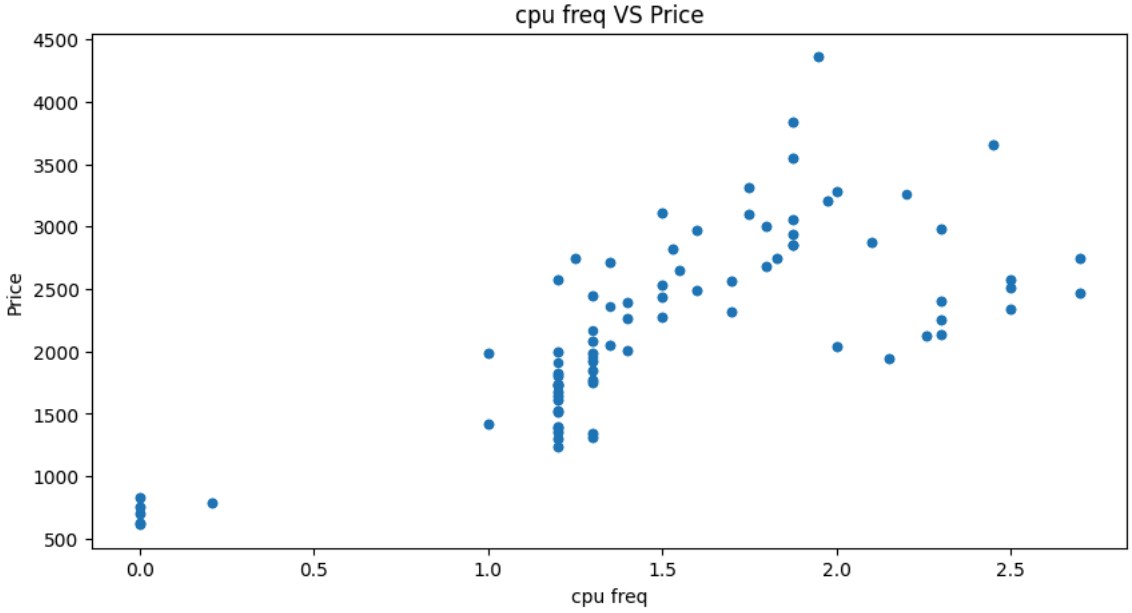
Everything vs Price

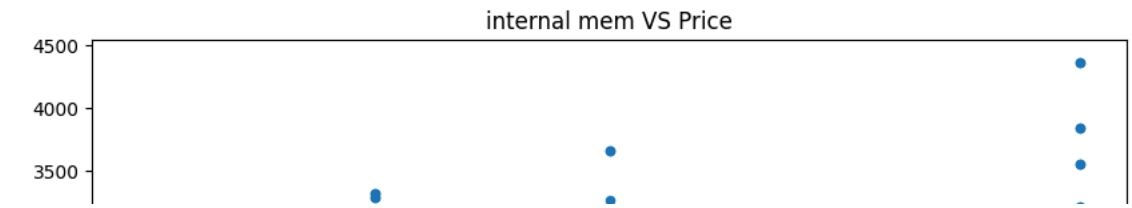
Double-click (or enter) to edit

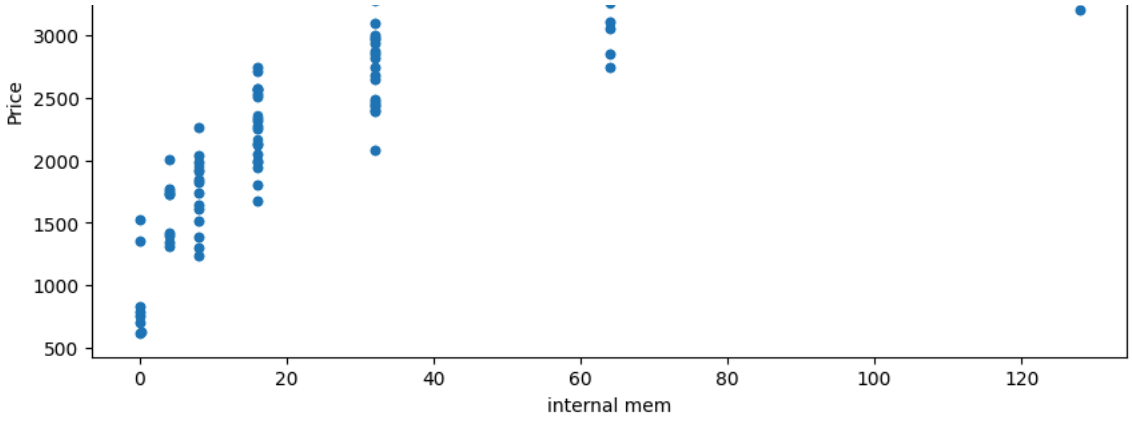
```
ContinuousCols=["weight","Price","cpu freq","internal mem","ram","battery","thickness","ppi"]

# Plotting scatter chart for each predictor vs the target variable
for predictor in ContinuousCols:
    CellData.plot.scatter(x=predictor, y='Price', figsize=(10,5), title=predictor+" VS "+ 'Price')
```

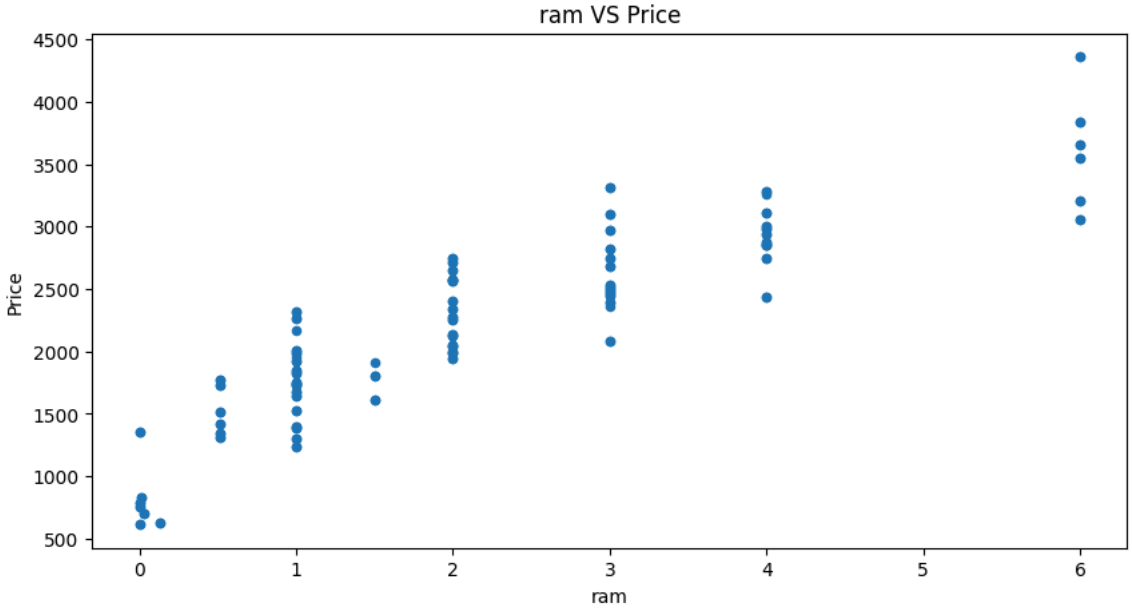

weight VS Price
Price
weight
4500
4000
3500
3000
2500
2000
1500
1000
500
100
200
300
400
500
600
700

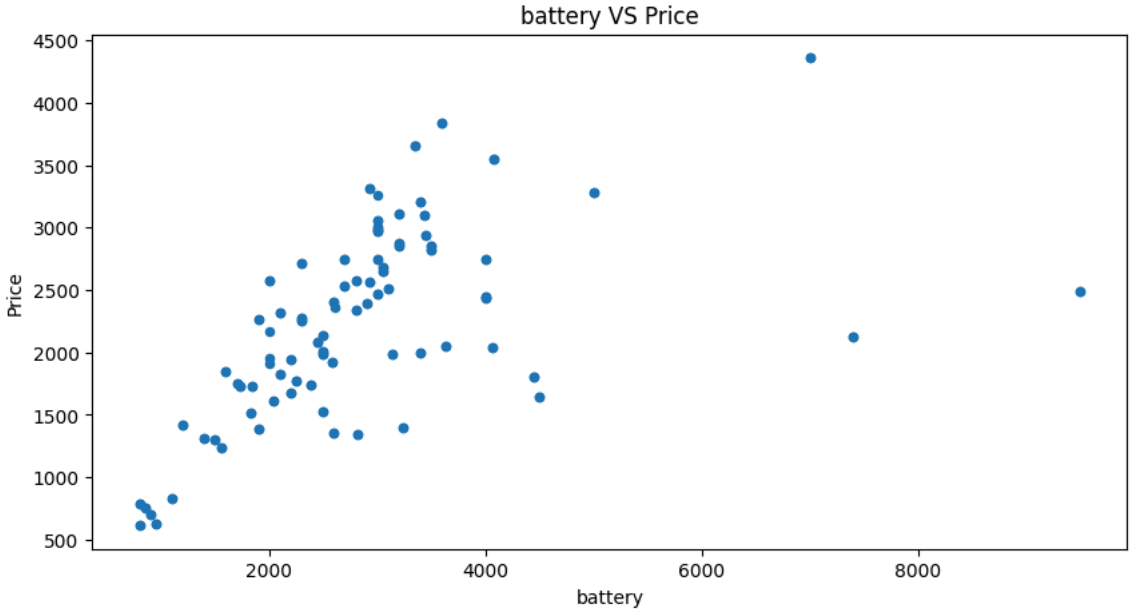

Price VS Price
Price
Price
4500
4000
3500
3000
2500
2000
1500
1000
500
500
1000
1500
2000
2500
3000
3500
4000
4500

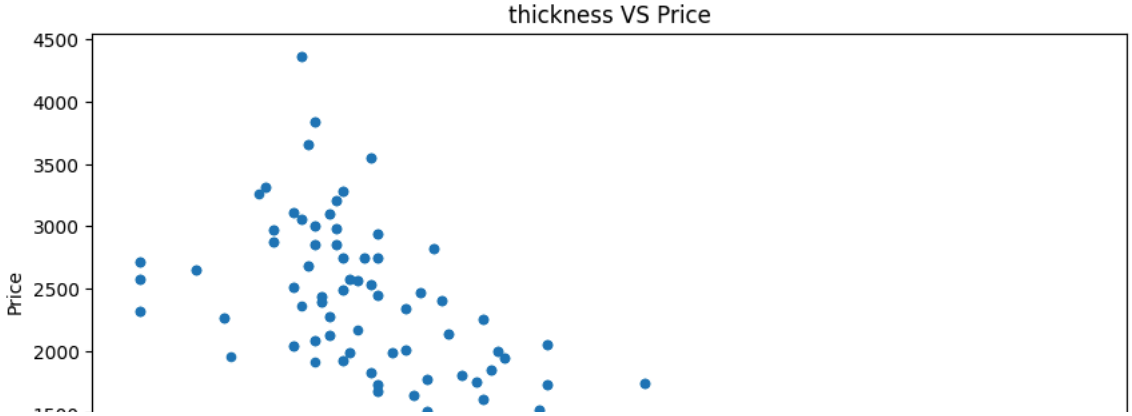

cpu freq VS Price
Price
cpu freq
4500
4000
3500
3000
2500
2000
1500
1000
500
0.0
0.5
1.0
1.5
2.0
2.5

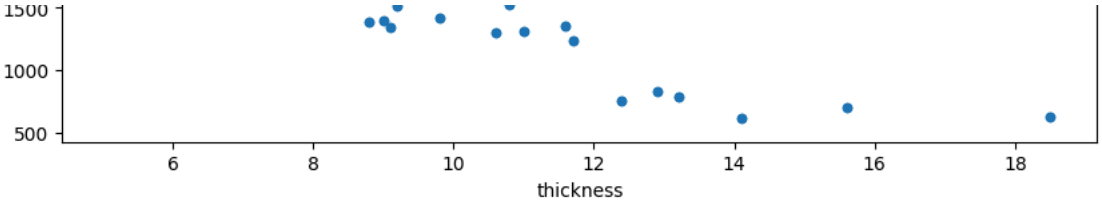

internal mem VS Price
4500
4000
3500

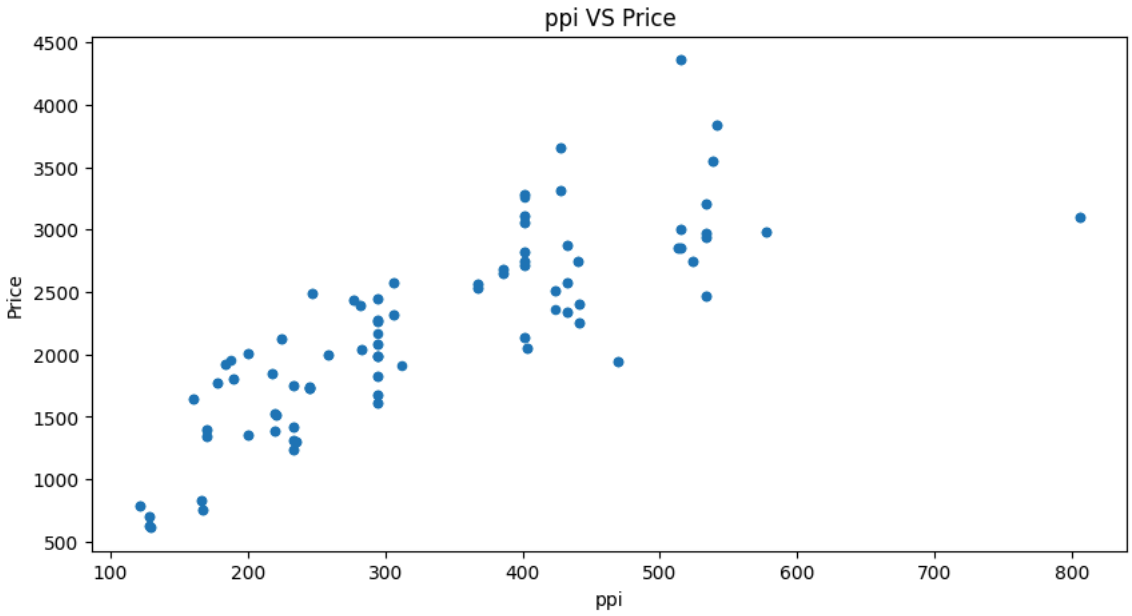

Price
3000
2500
2000
1500
1000
500
0
20
40
60
80
100
120
internal mem


ram VS Price
Price
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6
ram


battery VS Price
Price
4500
4000
3500
3000
2500
2000
1500
1000
500
2000
4000
6000
8000
battery


thickness VS Price
Price
4500
4000
3500
3000
2500
2000





Outlier

identifying the outliers

to wright the following code understand how means and interquartile range (measure of statistical dispersion) works.

Sort the Data: Arrange the data in ascending order.

Find the Quartiles:

- Q1 (First Quartile): The median of the lower half of the dataset (not including the median if the number of data points is odd).
- Q3 (Third Quartile): The median of the upper half of the dataset. Calculate

IQR: Subtract Q1 from Q3:

IQR =$Q3-Q1$

https://www.calculator.io/quartile-calculator/

this is the formula that has been worked into the code and allows for the removal of outliers that may impact the AI/Mechine learning model

```
def detect_outlier():
  for i in CellData:
    print('=======================',i,'==============================')
    q1,q2=np.percentile(CellData[i],[25,75])
    iqr=q2-q1
    print('25%, 75%:   ',q1,', ',q2)
    lower_bound=q1-(1.5*iqr)
    upper_bound=q2+(1.5*iqr)
    print('lower_bound, upper_bound:   ',lower_bound,', ',upper_bound)
    out=CellData[(CellData[i]<lower_bound) | (CellData[i]>upper_bound)]
    print('outliers_val: ',out[i].to_list())
    print('Total Outlier: ',round((len(out)/len(CellData))*100,2),'%')


detect_outlier()
```

```
outliers_val:  [489.0, 489.0, 260.0, 260.0, 310.0, 310.0, 279.0, 66.0, 404.0, 279.0, 66.0, 404.0, 393.0, 393.0, 78.4, 78.4, 77.9,
Total Outlier:  14.91 %
======================== resoloution ================================
25%, 75%:    4.8 ,  5.5
lower_bound, upper_bound:    3.7499999999999996 ,  6.550000000000001
outliers_val:  [10.1, 10.1, 7.0, 7.0, 8.0, 8.0, 7.0, 1.5, 8.0, 7.0, 1.5, 8.0, 8.0, 8.0, 2.4, 2.4, 2.4, 2.4, 2.2, 2.2, 12.2, 12.2,
Total Outlier:  16.15 %
======================== ppi ================================
25%, 75%:    233.0 ,  428.0
lower_bound, upper_bound:    -59.5 ,  720.5
outliers_val:  [806, 806]
Total Outlier:  1.24 %
======================== cpu core ================================
25%, 75%:    4.0 ,  8.0
lower_bound, upper_bound:    -2.0 ,  14.0
outliers_val:  []
Total Outlier:  0.0 %
======================== cpu freq ================================
25%, 75%:    1.2 ,  1.875
lower_bound, upper_bound:    0.18749999999999978 ,  2.8875
outliers_val:  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Total Outlier:  6.21 %
======================== internal mem ================================
25%, 75%:    8.0 ,  32.0
lower_bound, upper_bound:    -28.0 ,  68.0
outliers_val:  [128.0, 128.0, 128.0, 128.0, 128.0, 128.0, 128.0, 128.0]
Total Outlier:  4.97 %
======================== ram ================================
25%, 75%:    1.0 ,  3.0
lower_bound, upper_bound:    -2.0 ,  6.0
outliers_val:  []
Total Outlier:  0.0 %
======================== RearCam ================================
25%, 75%:    5.0 ,  16.0
lower_bound, upper_bound:    -11.5 ,  32.5
outliers_val:  []
Total Outlier:  0.0 %
======================== Front_Cam ================================
25%, 75%:    0.0 ,  8.0
lower_bound, upper_bound:    -12.0 ,  20.0
outliers_val:  []
Total Outlier:  0.0 %
======================== battery ================================
25%, 75%:    2040.0 ,  3240.0
lower_bound, upper_bound:    240.0 ,  5040.0
outliers_val:  [7400, 7400, 9500, 9500, 7000, 7000]
Total Outlier:  3.73 %
======================== thickness ================================
```

This function above calculates the Interquartile Range (IQR) for each column in the dataset, defines bounds for what would be considered an outlier (1.5 times the IQR below the first quartile or above the third quartile) pulled from the formula above, and then identifies and prints out the outliers & percentage of total outliers.

the function `np.percentile` is used to compute the nth percentile of the given data (array elements) along the specified axis.

https://www.geeksforgeeks.org/numpy-percentile-in-python/

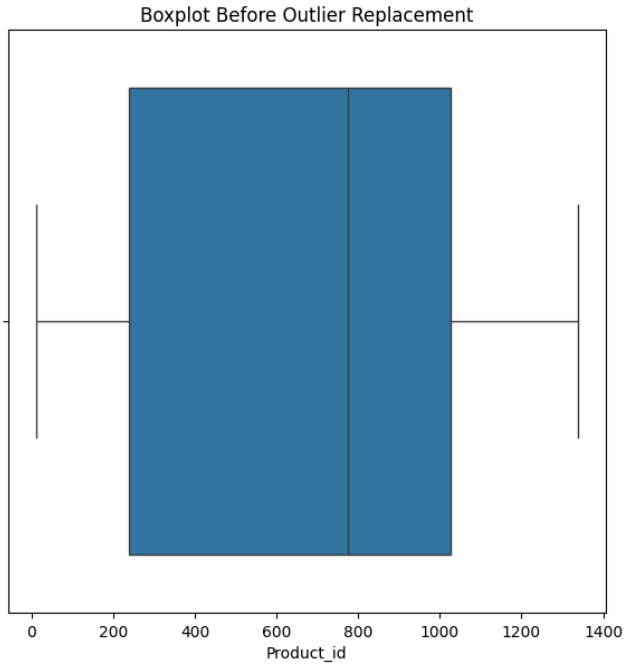
Removal and Comparison of Outlires

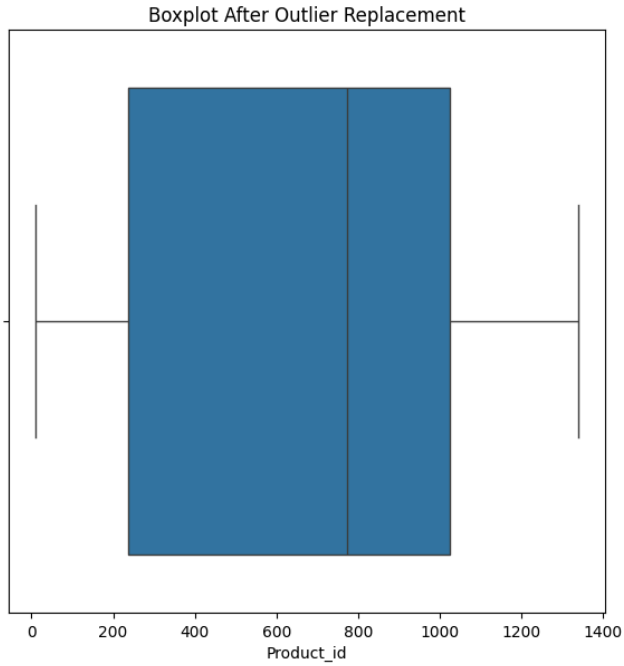
the follow is calling on the above and create veriables to call on to compare the orgian and capped data. then creates graphs to show the comparison

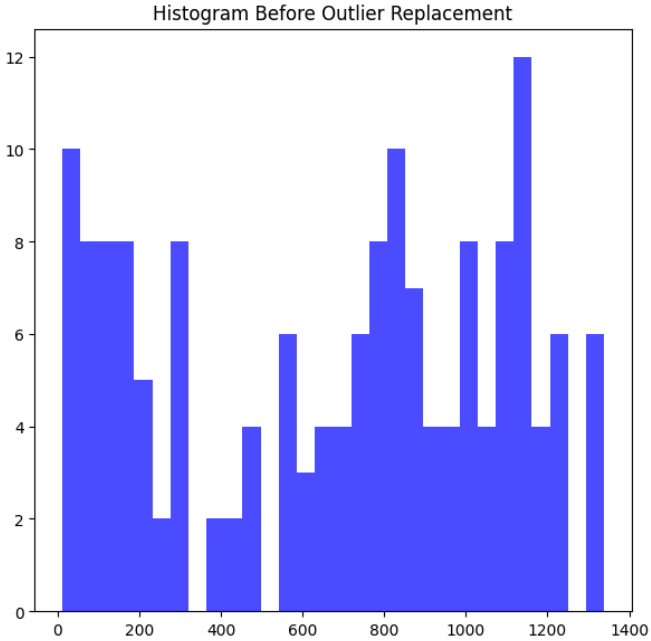
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

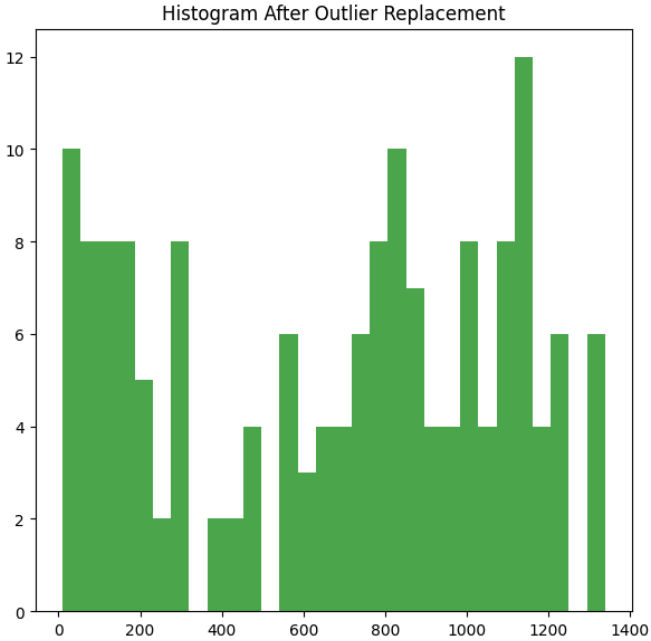
```
def cap_outliers(series, width=1.5):
    """Caps outliers in a pandas series based on IQR and returns the modified series."""
    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
    iqr = q3 - q1
    upper_bound = q3 + width * iqr
    return series.clip(upper=upper_bound)
```

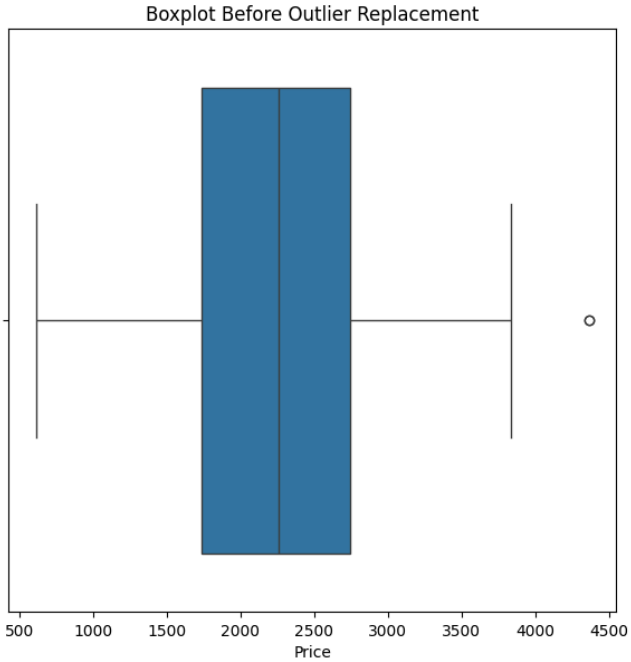
```
def plot_boxplots_and_histograms(data, column):
    """Plots boxplots and histograms for original and outlier-capped data of a given column."""
    data_modified = data.copy()
    data_modified[column] = cap_outliers(data_modified[column])

    plt.figure(figsize=(12, 12))

    # Plot original data boxplot
    plt.subplot(2, 2, 1)
    sns.boxplot(x=data[column])
    plt.title('Boxplot Before Outlier Replacement')

    # Plot modified data boxplot
    plt.subplot(2, 2, 2)
    sns.boxplot(x=data_modified[column])
    plt.title('Boxplot After Outlier Replacement')

    # Plot original data histogram
    plt.subplot(2, 2, 3)
    plt.hist(data[column], bins=30, alpha=0.7, color='blue')
    plt.title('Histogram Before Outlier Replacement')

    # Plot modified data histogram
    plt.subplot(2, 2, 4)
    plt.hist(data_modified[column], bins=30, alpha=0.7, color='green')
    plt.title('Histogram After Outlier Replacement')

    plt.tight_layout()
    plt.show()



# Assuming 'CellData' is a DataFrame already loaded in your workspace
if __name__ == "__main__":
    for column in CellData.columns:
        plot_boxplots_and_histograms(CellData, column)
```

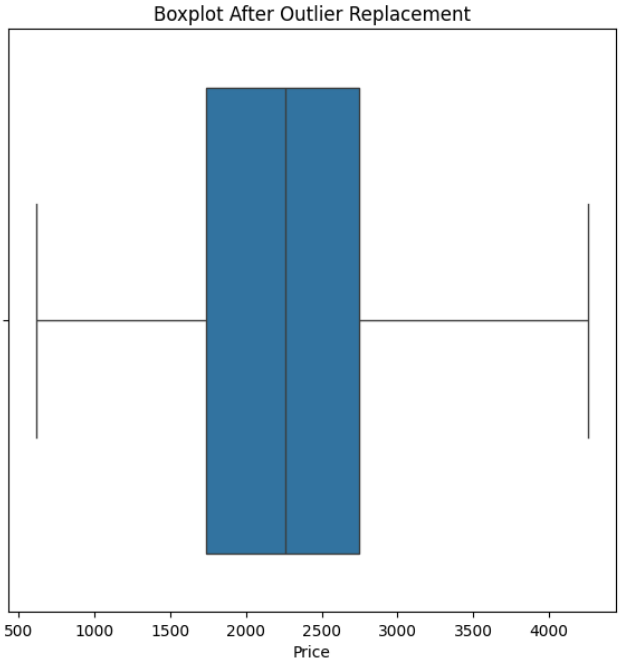

Boxplot Before Outlier Replacement
0
200
400
600
800
1000
1200
1400
Product_id

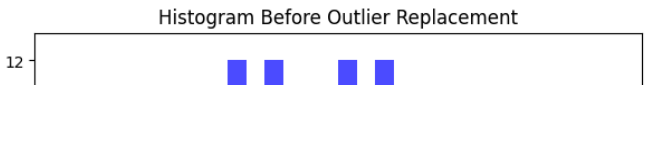

Boxplot After Outlier Replacement
0
200
400
600
800
1000
1200
1400
Product_id

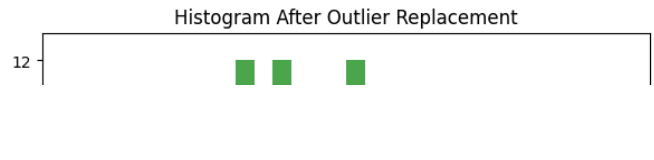

Histogram Before Outlier Replacement
0
2
4
6
8
10
12
0
200
400
600
800
1000
1200
1400

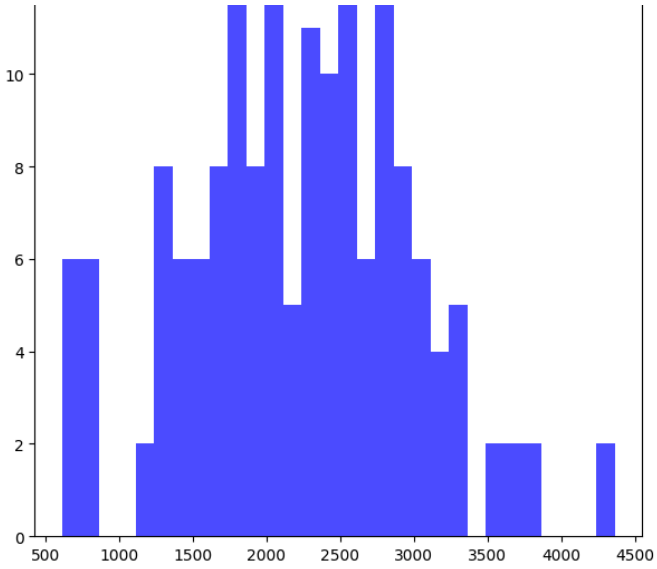

Histogram After Outlier Replacement
0
2
4
6
8
10
12
0
200
400
600
800
1000
1200
1400

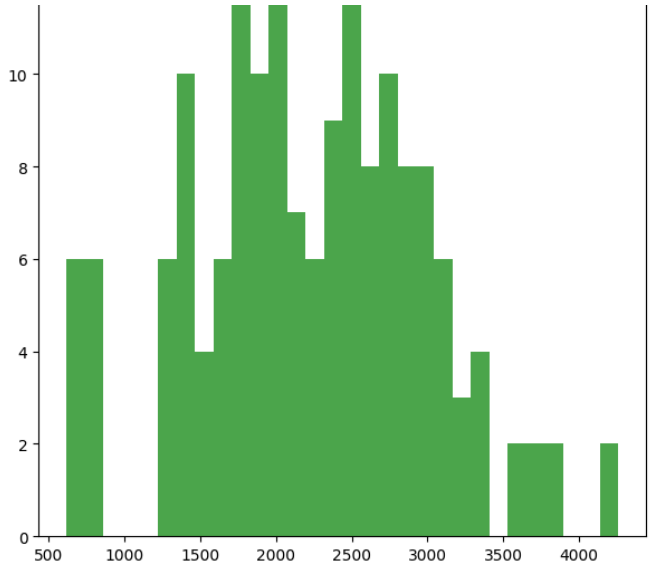

Boxplot Before Outlier Replacement
500
1000
1500
2000
2500
3000
3500
4000
4500
Price

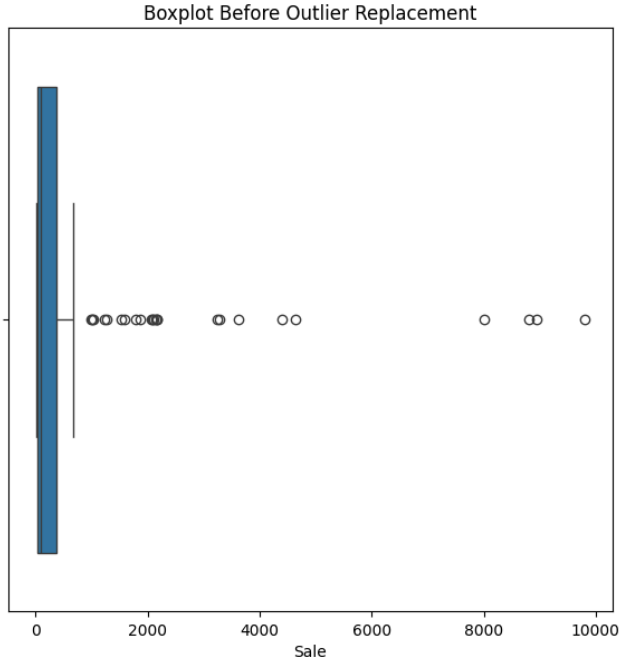

Boxplot After Outlier Replacement
500
1000
1500
2000
2500
3000
3500
4000
Price

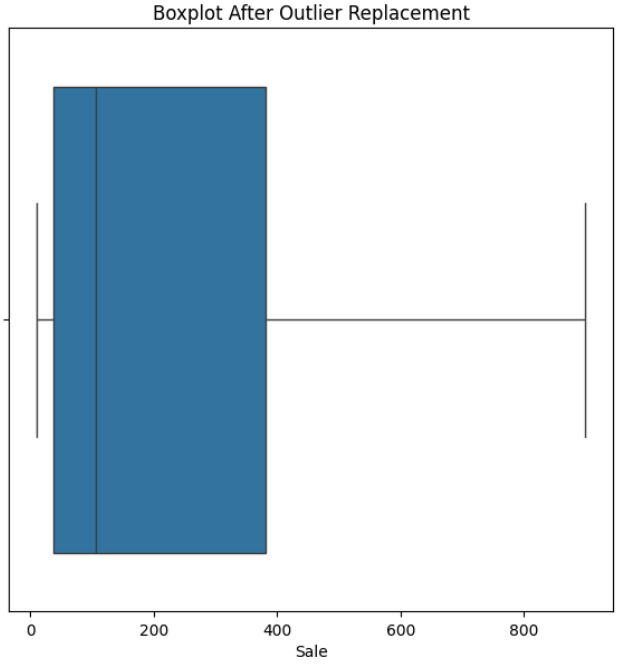

Histogram Before Outlier Replacement
12

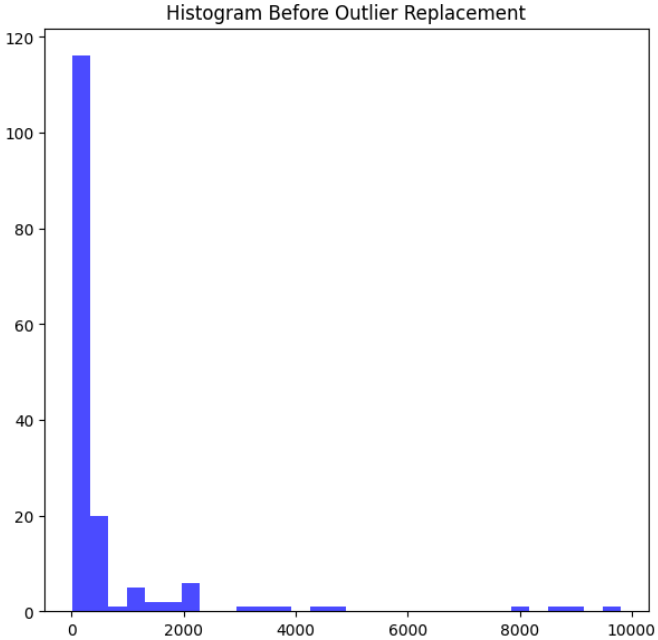

Histogram After Outlier Replacement
12

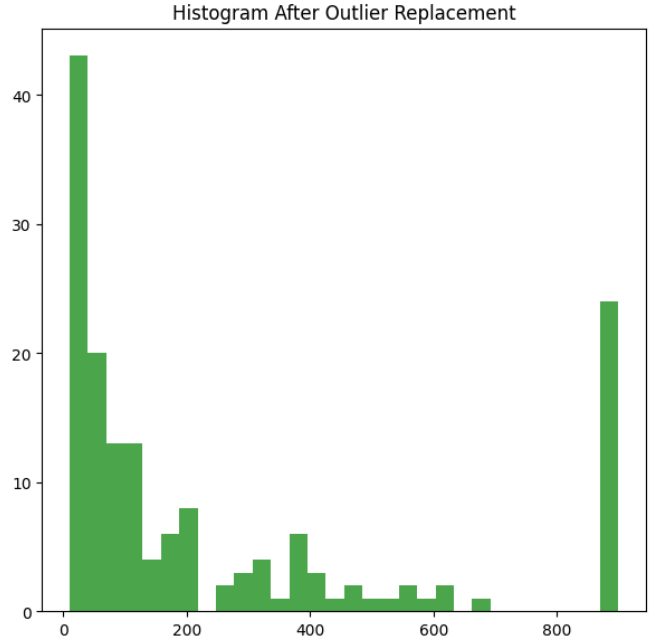


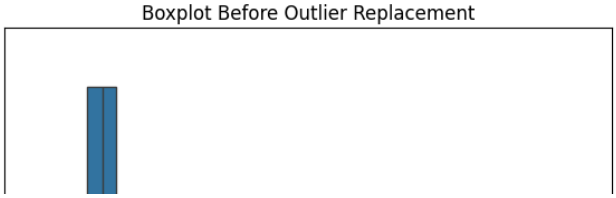


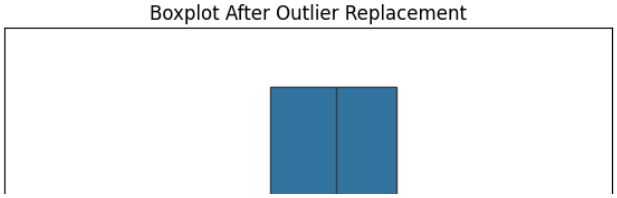

Boxplot Before Outlier Replacement
Sale

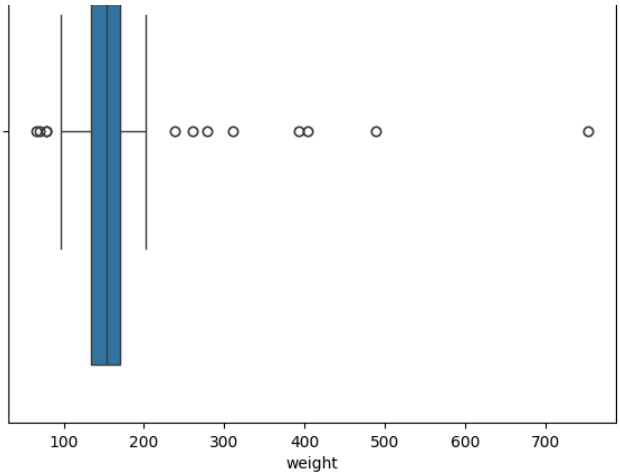

Boxplot After Outlier Replacement
Sale

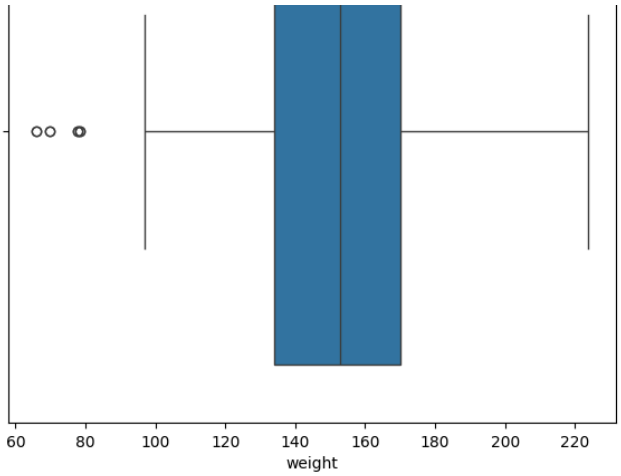

Histogram Before Outlier Replacement

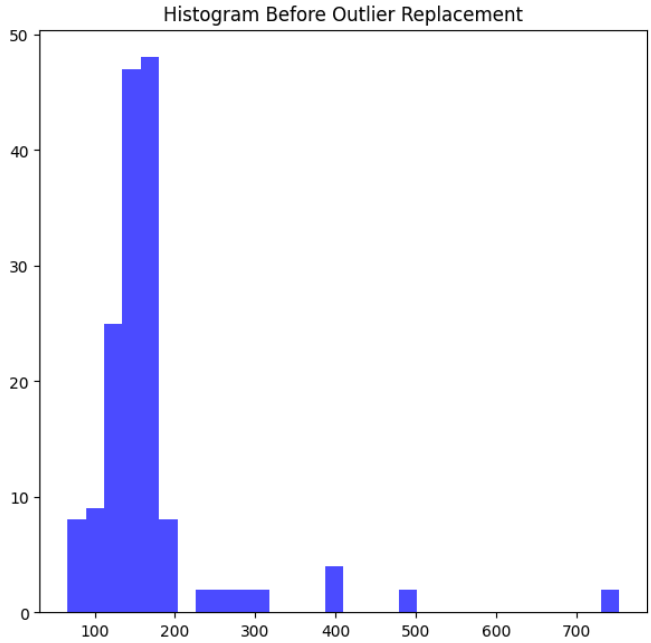

Histogram After Outlier Replacement

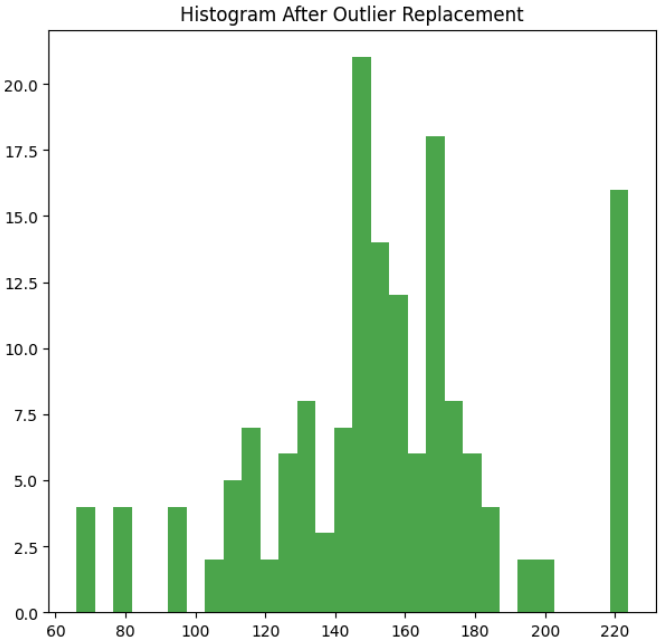

Boxplot Before Outlier Replacement

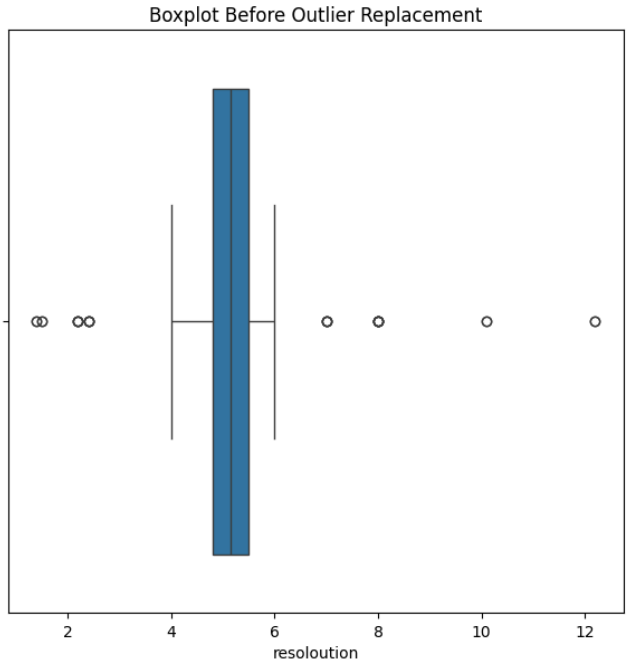

Boxplot After Outlier Replacement

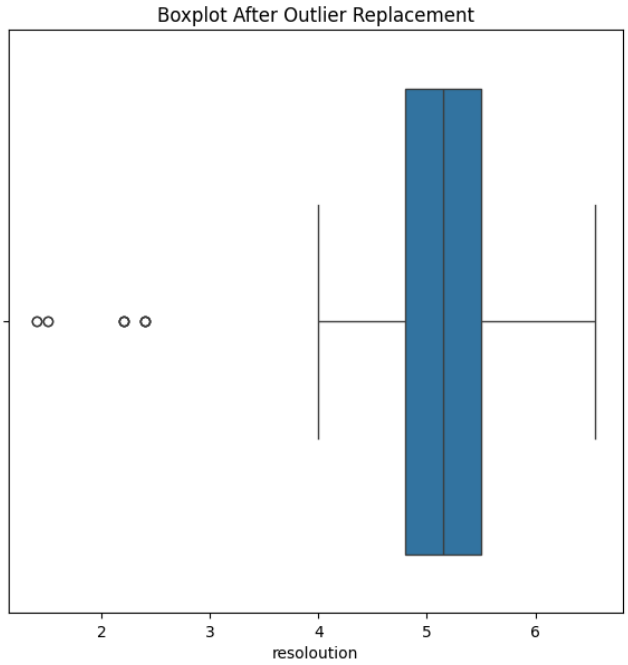

100
200
300
400
500
600
700
weight

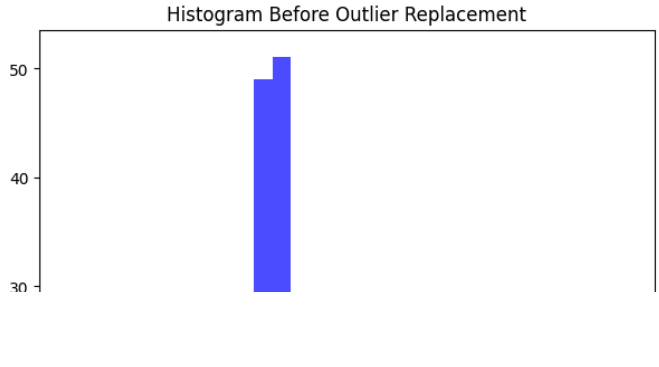

60
80
100
120
140
160
180
200
220
weight

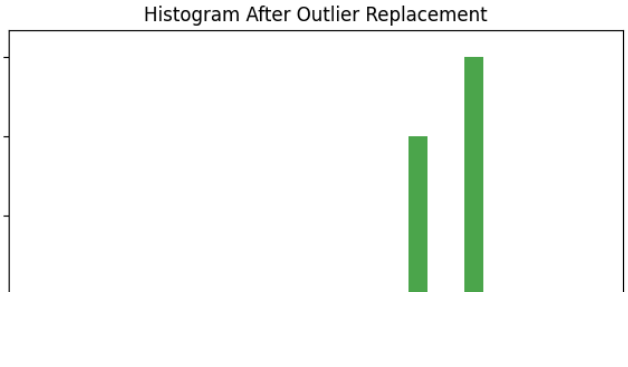

Histogram Before Outlier Replacement
0
10
20
30
40
50
100
200
300
400
500
600
700

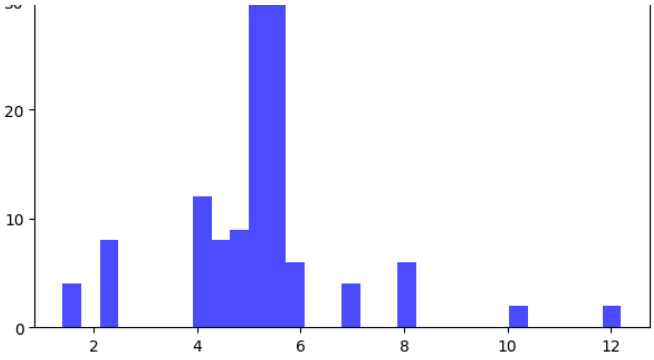

Histogram After Outlier Replacement
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
60
80
100
120
140
160
180
200
220

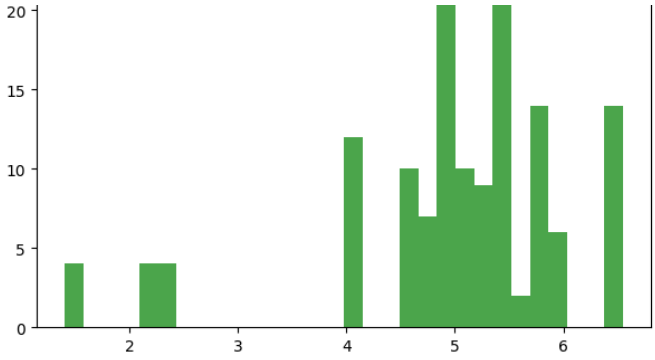

Boxplot Before Outlier Replacement
2
4
6
8
10
12
resoloution


Boxplot After Outlier Replacement
2
3
4
5
6
resoloution

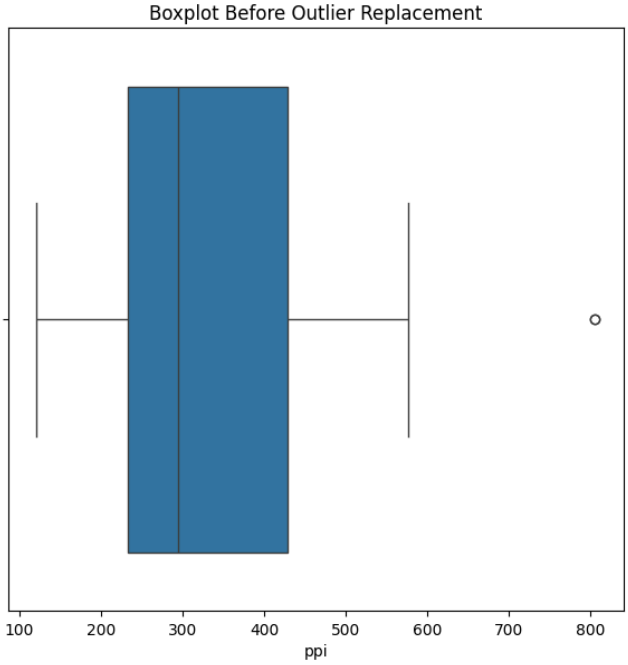

Histogram Before Outlier Replacement
30
40
50


Histogram After Outlier Replacement
25
30
35

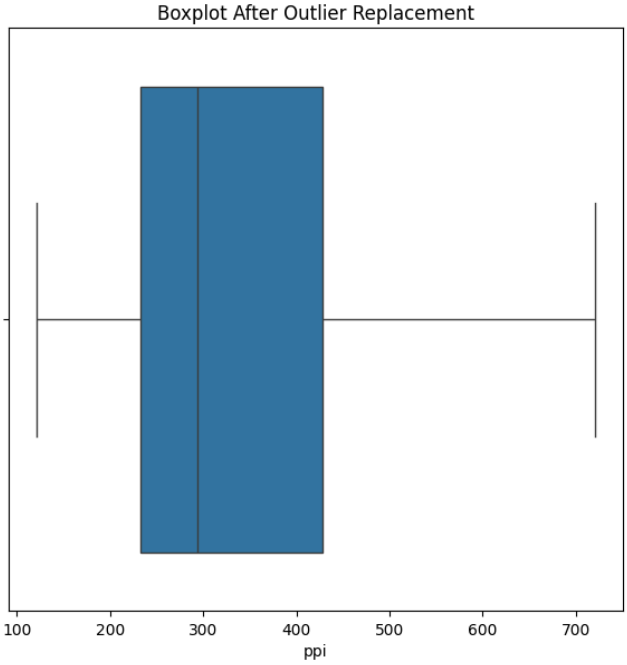




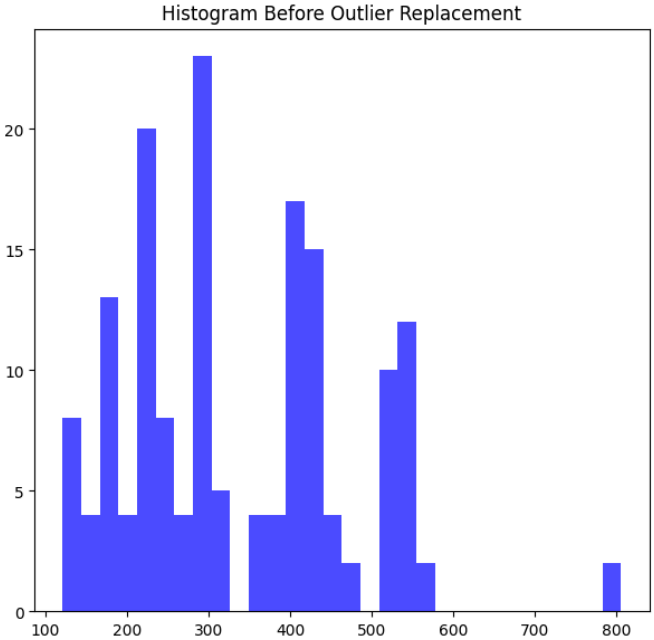
Boxplot Before Outlier Replacement



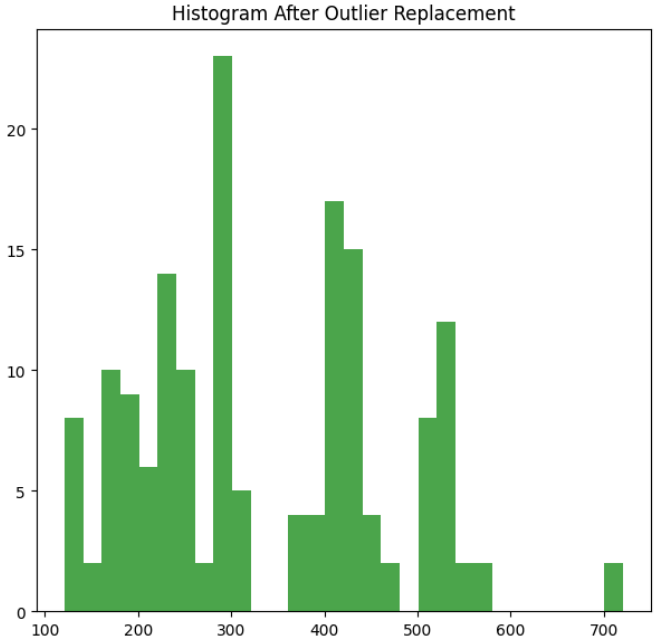
Boxplot After Outlier Replacement



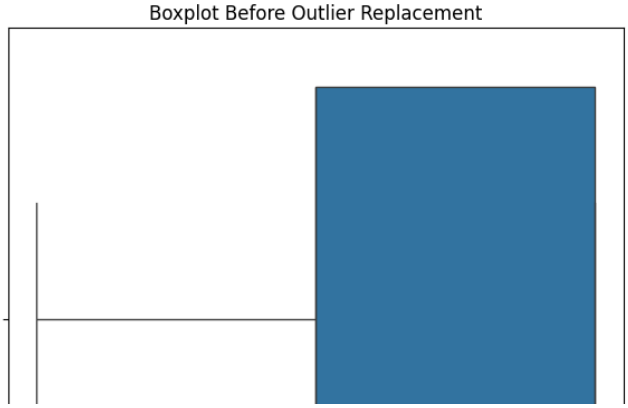
Histogram Before Outlier Replacement



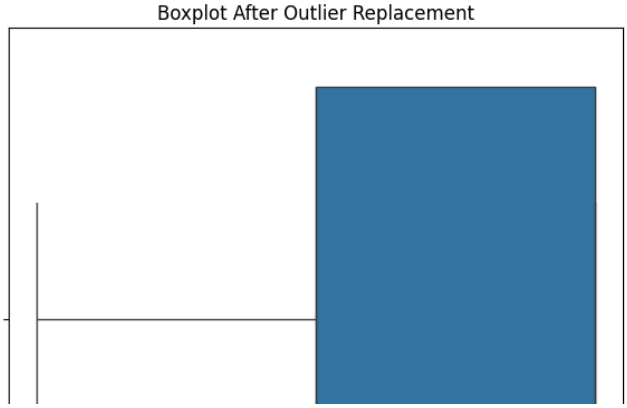
Histogram After Outlier Replacement

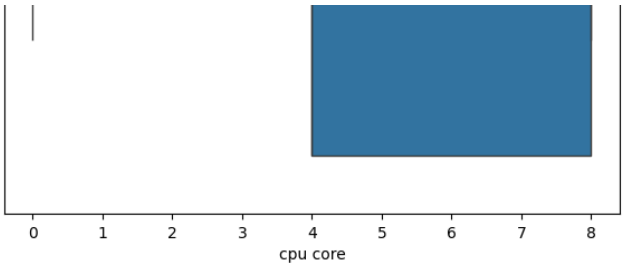


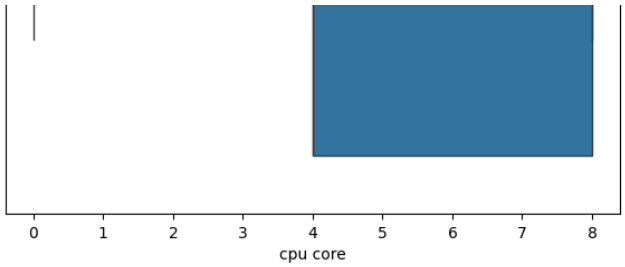
Boxplot Before Outlier Replacement

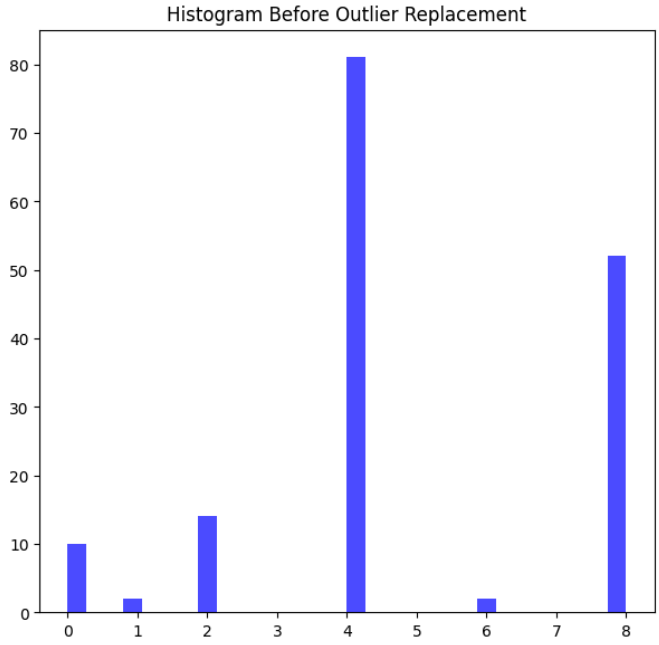


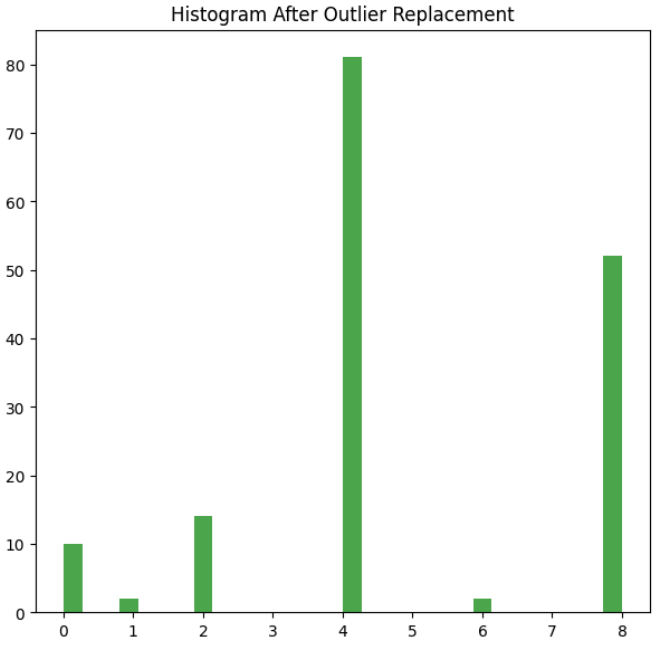
Boxplot After Outlier Replacement

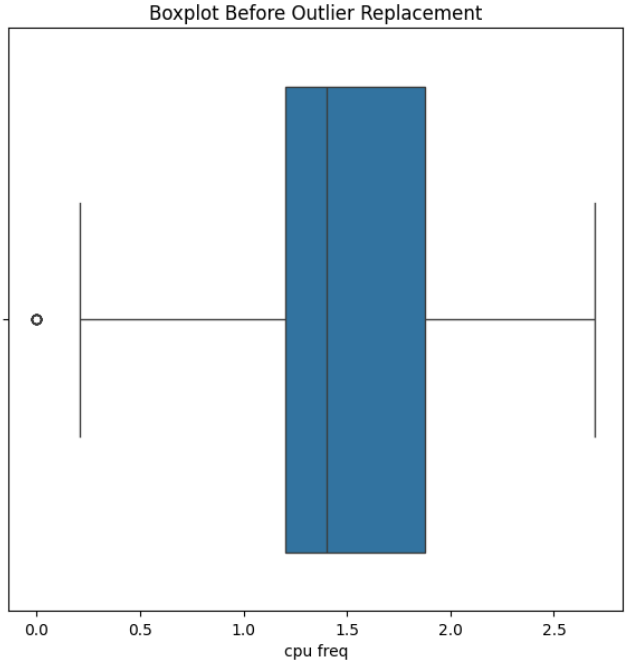


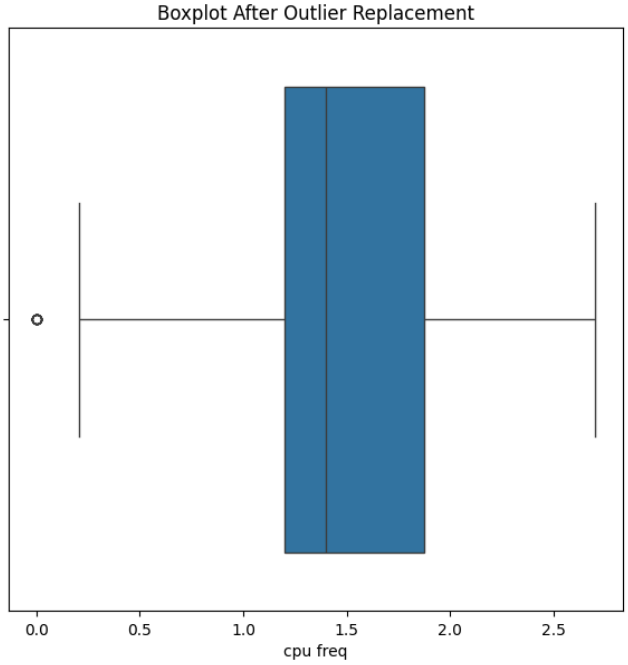

0 1 2 3 4 5 6 7 8
cpu core

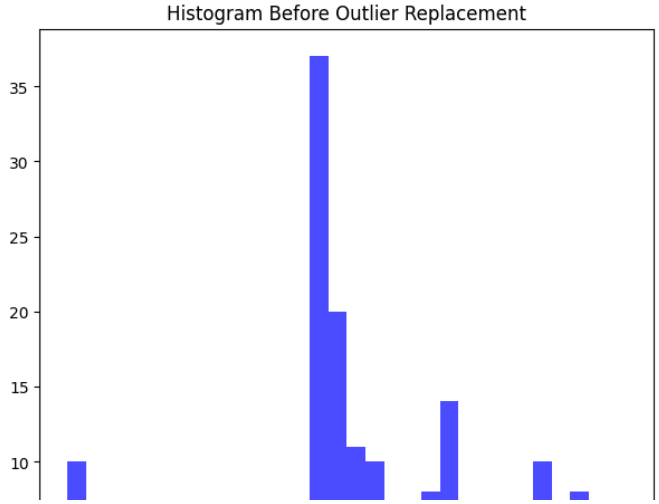

0 1 2 3 4 5 6 7 8
cpu core

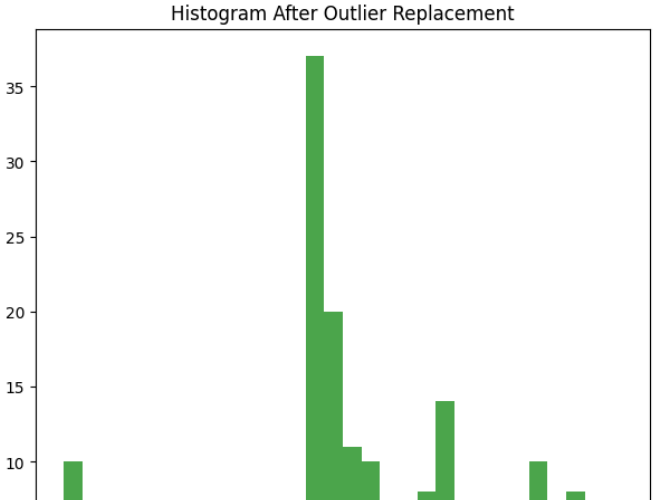

Histogram Before Outlier Replacement
0 10 20 30 40 50 60 70 80
0 1 2 3 4 5 6 7 8

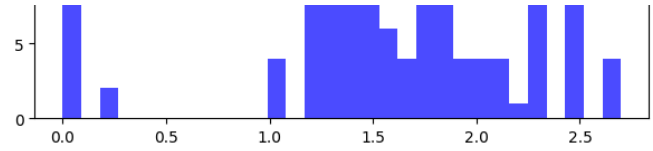

Histogram After Outlier Replacement
0 10 20 30 40 50 60 70 80
0 1 2 3 4 5 6 7 8

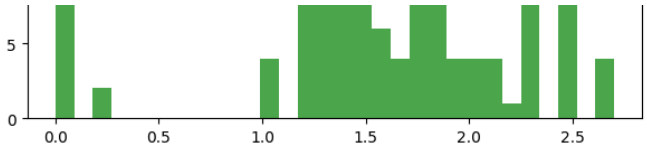

Boxplot Before Outlier Replacement
0.0 0.5 1.0 1.5 2.0 2.5
cpu freq

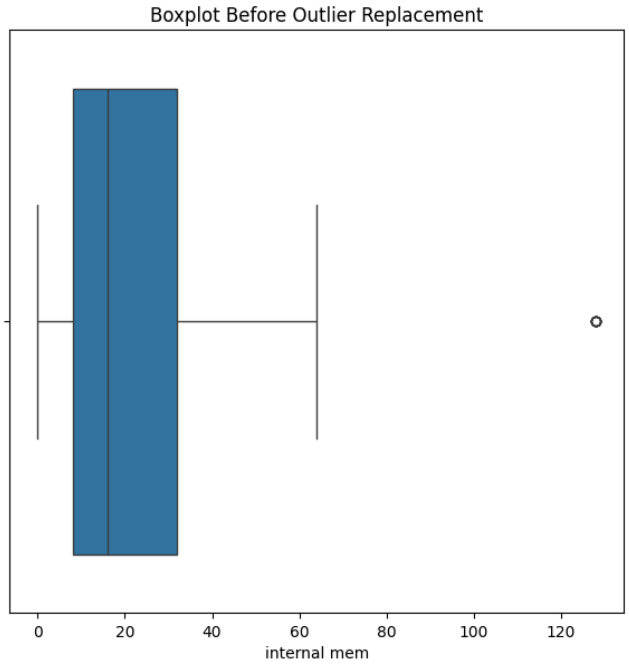

Boxplot After Outlier Replacement
0.0 0.5 1.0 1.5 2.0 2.5
cpu freq

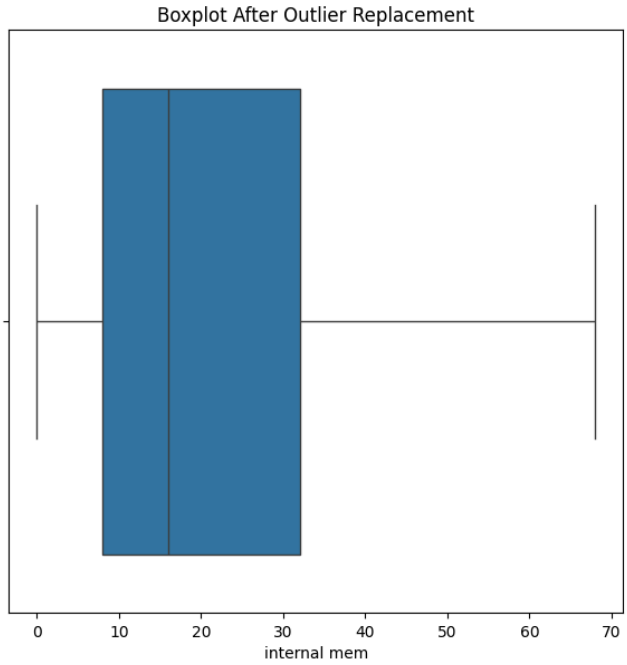

Histogram Before Outlier Replacement
10 15 20 25 30 35

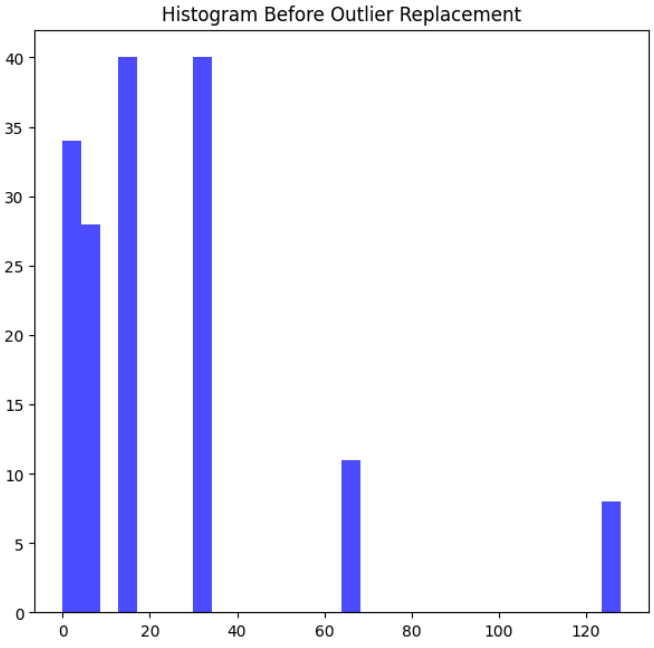

Histogram After Outlier Replacement
10 15 20 25 30 35

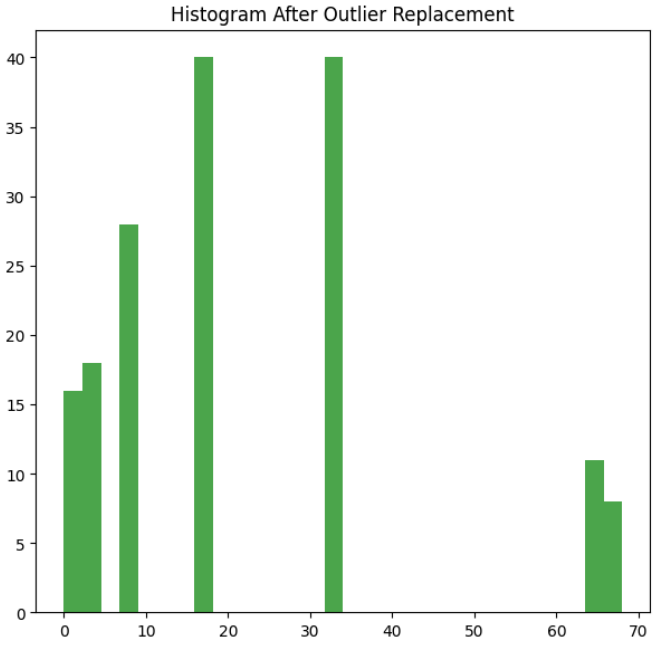

0.0
0.5
1.0
1.5
2.0
2.5
0
5

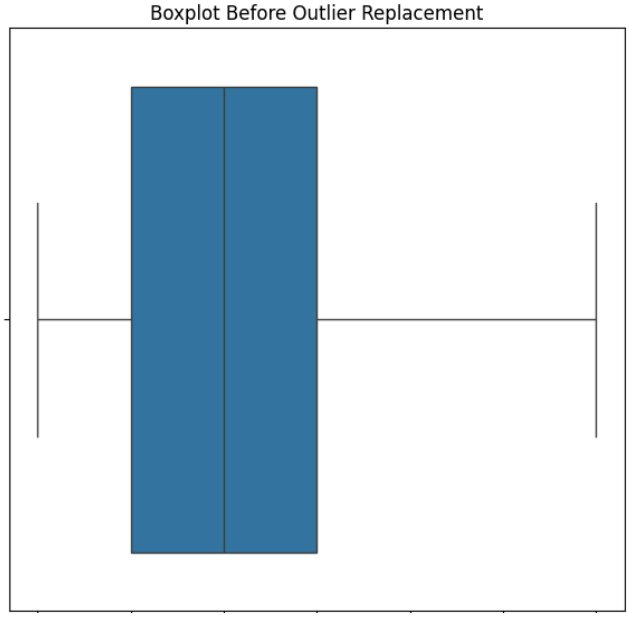

0.0
0.5
1.0
1.5
2.0
2.5
0
5

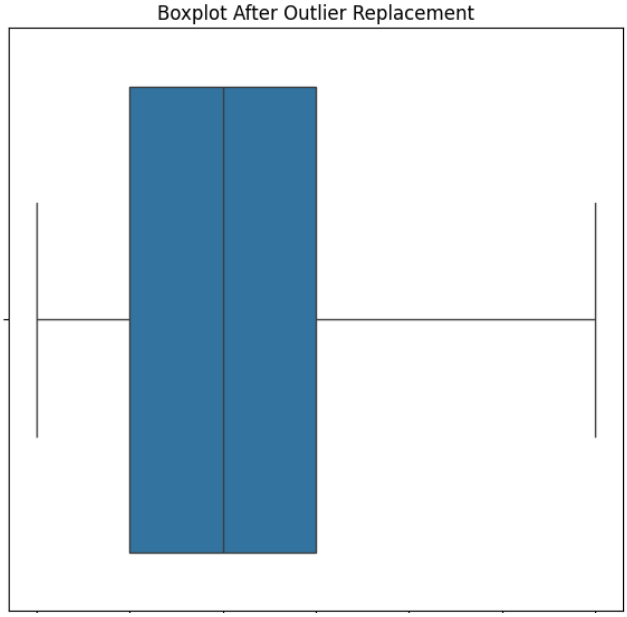

Boxplot Before Outlier Replacement
0
20
40
60
80
100
120
internal mem


Boxplot After Outlier Replacement
0
10
20
30
40
50
60
70
internal mem

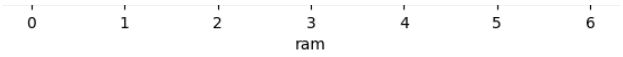

Histogram Before Outlier Replacement
0
5
10
15
20
25
30
35
40
0
20
40
60
80
100
120

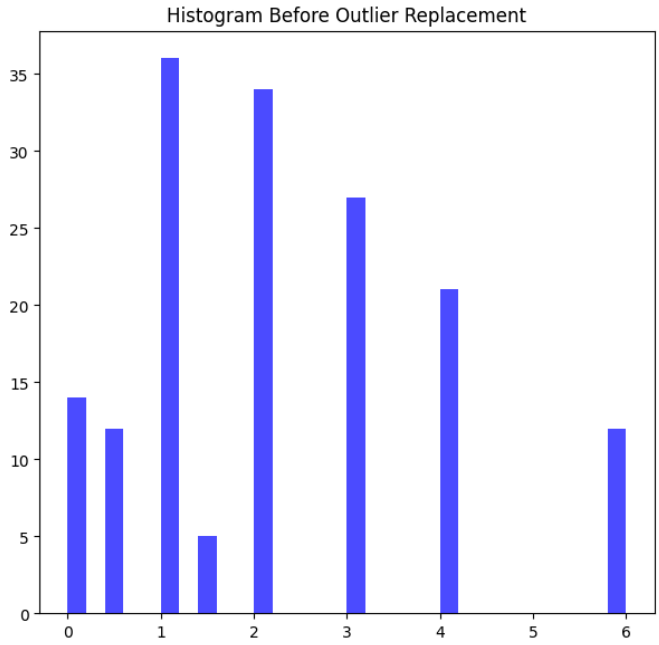

Histogram After Outlier Replacement
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50
60
70

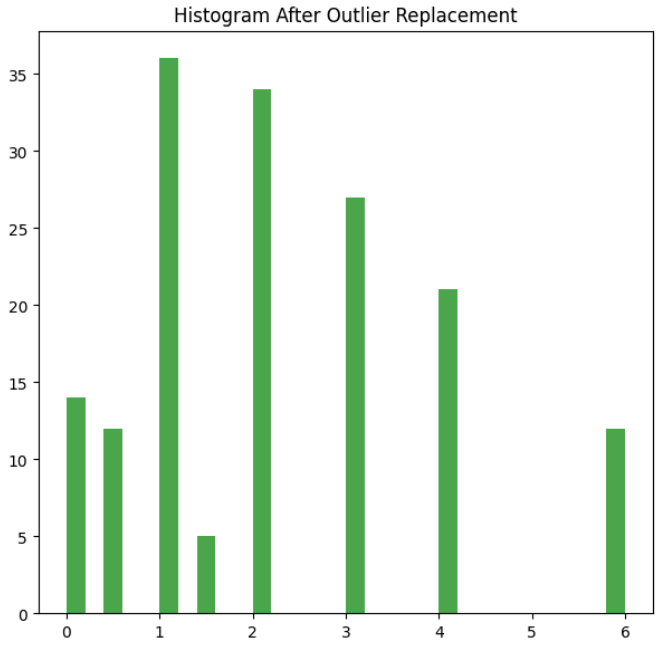

Boxplot Before Outlier Replacement

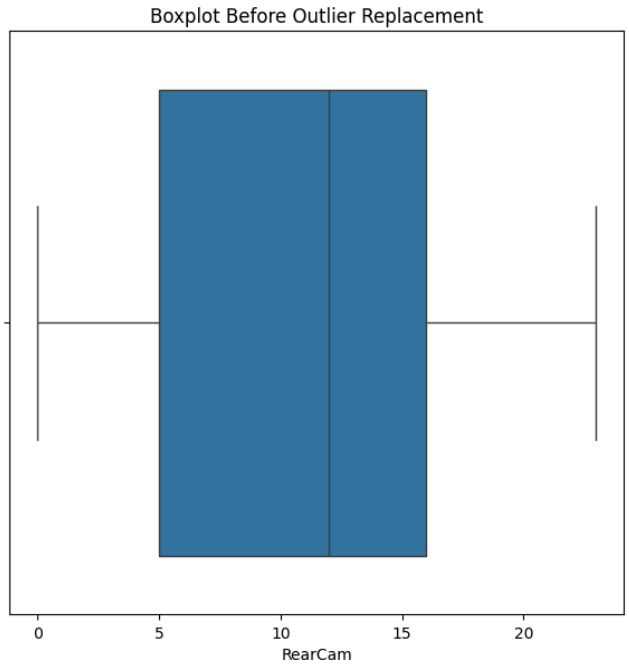

Boxplot After Outlier Replacement

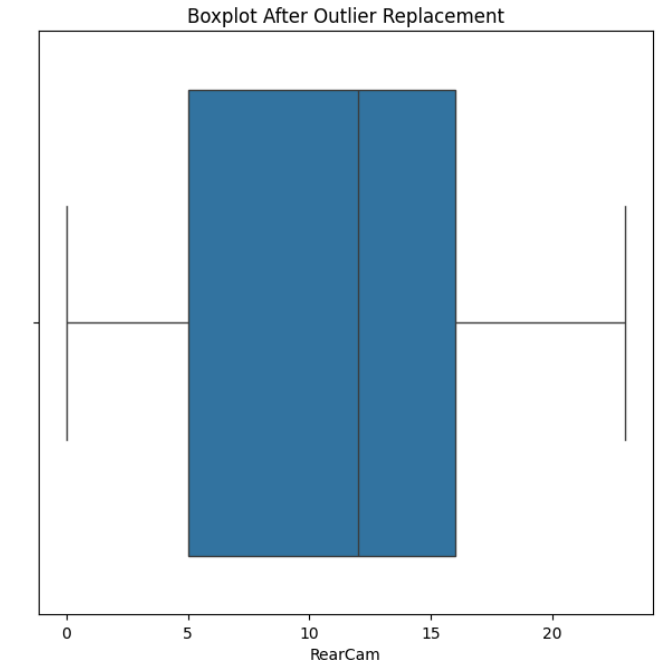

0
1
2
3
4
5
6
ram

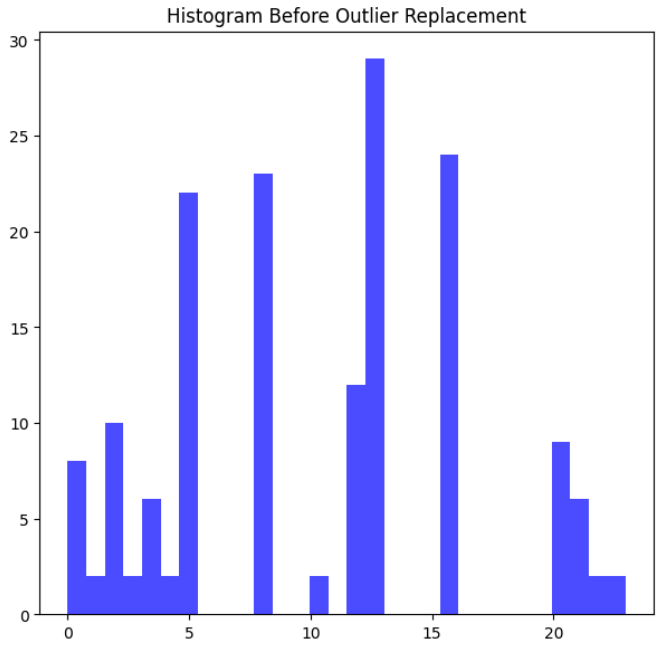

0
1
2
3
4
5
6
ram

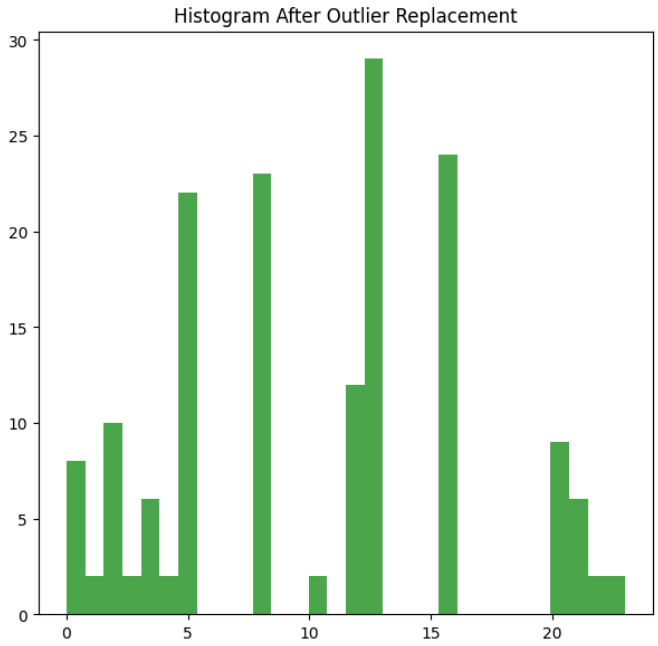

Histogram Before Outlier Replacement
0
5
10
15
20
25
30
35
0
1
2
3
4
5
6

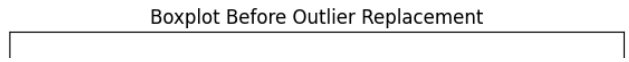

Histogram After Outlier Replacement
0
5
10
15
20
25
30
35
0
1
2
3
4
5
6

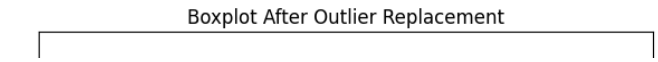

Boxplot Before Outlier Replacement
0
5
10
15
20
RearCam

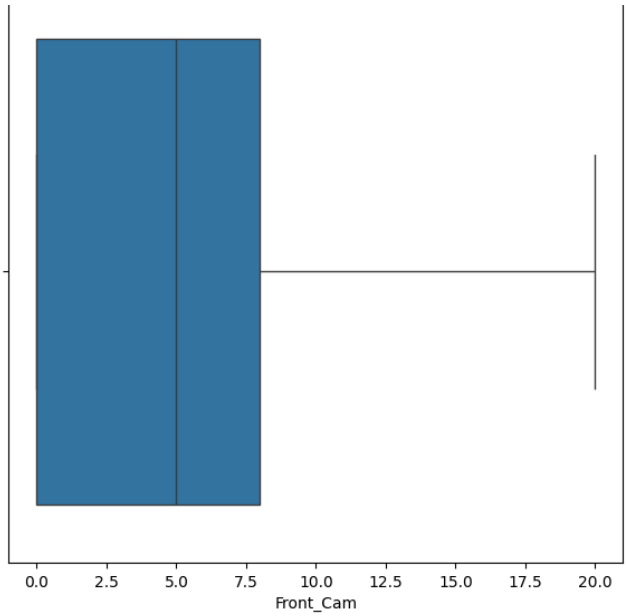

Boxplot After Outlier Replacement
0
5
10
15
20
RearCam

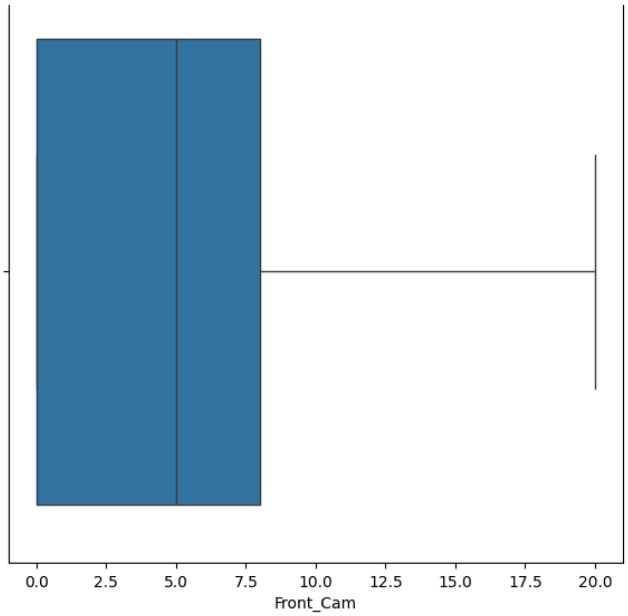

Histogram Before Outlier Replacement
0
5
10
15
20
25
30
0
5
10
15
20

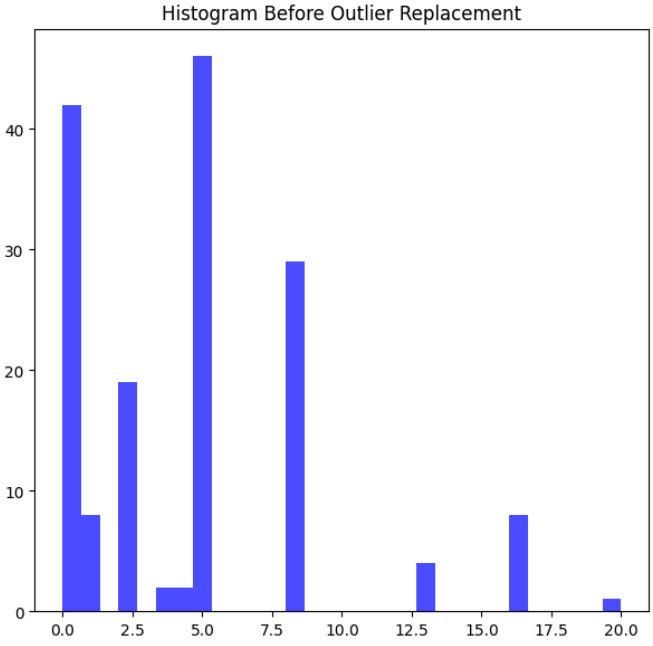

Histogram After Outlier Replacement
0
5
10
15
20
25
30
0
5
10
15
20

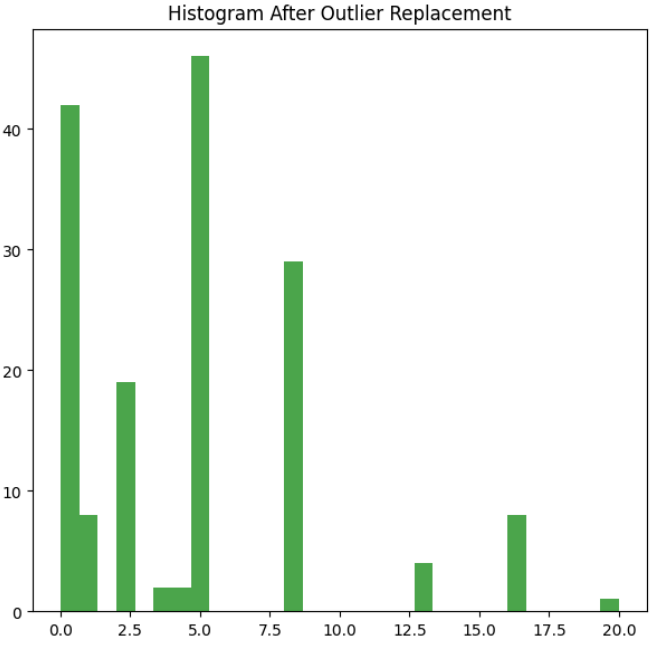

Boxplot Before Outlier Replacement

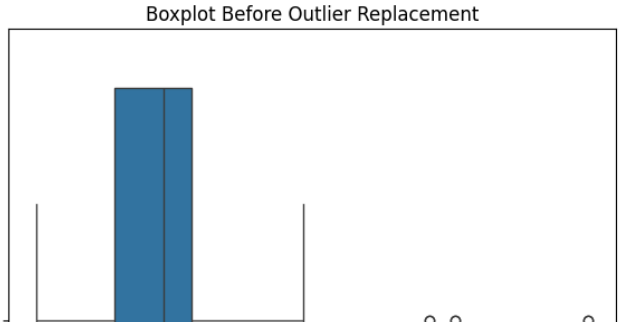

Boxplot After Outlier Replacement

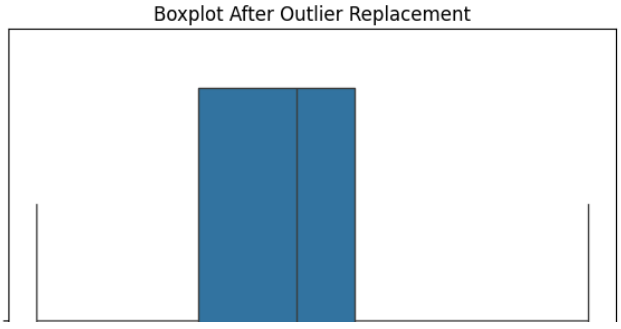

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Front_Cam


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Front_Cam


Histogram Before Outlier Replacement
0
10
20
30
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Histogram After Outlier Replacement
0
10
20
30
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Boxplot Before Outlier Replacement


Boxplot After Outlier Replacement

the above shows.... note the data may need to be sanited before running so add in a function above to do that :)

ANOVA Test

the code tests the categorical predictors against a target variable in the dataset. The function checks if there is a statistically significant difference in means for each categorical predictor with respect to the target variable (the Price).

```
def FunctionAnova(inpData, TargetVar, CatPredList):
    SelectedPredictors=[]
    print('============== ANOVA ==============\n')
    from scipy.stats import f_oneway
    for predictor in CatPredList:
        CategoryGroupLists=inpData.groupby(predictor)[TargetVar].apply(list)
        AnovaResults = f_oneway(*CategoryGroupLists)

        if (AnovaResults[1] < 0.05):
            print(predictor, 'Correlation \n', TargetVar, '| Value:', AnovaResults[1])
            SelectedPredictors.append(predictor)
        else:
            print(predictor, 'NO Correlation \n', TargetVar, '| Value:', AnovaResults[1])

    return(SelectedPredictors)
```

this is done by:

1. importing stament `f_oneway` (The test is called "one-way" because there is only one independent variable or factor under consideration, and it classifies subjects into different groups based on this single classification factor.)

2. then it is initializised through `SelectedPredictors=[]` which stores the names of the pedictors
3. the title is printed `print('============== ANOVA ==============\n')`
4. the loop is created

```
for predictor in CatPredList:
    CategoryGroupLists=inpData.groupby(predictor)[TargetVar].apply(list)
    AnovaResults = f_oneway(*CategoryGroupLists)
```

- Groups the Data: The function groups the data in inpData by each category of the given predictor.
- Collects Values: For each group, it collects the values of TargetVar into a list.
- Creates List of Lists: This collection process results in a list of lists, where each inner list contains all the values of TargetVar that correspond to a specific category of the predictor.
- perform ANOVA test: `f_oneway(*CategoryGroupLists)` performs the one-way ANOVA test on the grouped data. Noting the * is used to unpack the list.

5. Evaluating and printing Results

- Checking relevents: if it is less then 0.05 them there is a stistically important differance from the mean of the Price.

6. Print & Store:

- Depending on whether the p-value indicates significance, it either:
    - prints that there is a correlation and adds the predictor to SelectedPredictors
    - prints that there is no correlation.

7. finally Returns the significant predictors

```
CatPredList=["cpu core","RearCam","Front_Cam","resoloution"]
FunctionAnova(inpData=CellData,TargetVar='Price', CatPredList=CatPredList)
```

```
    ============== ANOVA ==============

    cpu core Correlation
     Price | Value: 3.721669566887769e-25
    RearCam Correlation
     Price | Value: 4.810966235170514e-43
    Front_Cam Correlation
     Price | Value: 2.370649577763112e-28
    resoloution Correlation
     Price | Value: 3.8580541035958067e-35
    ['cpu core', 'RearCam', 'Front_Cam', 'resoloution']
```

the above code is defining the CatPredList function by stating the catagorical columns the TsrgetVar function as price the inpData as CellData our main data set

this will run through the `FunctionAnova` and output the data

Data Preperation For Machine Learning

The Selected Columns is used to take the relvent columns and creating a secondary verable which will be used to run through the Machine Learning code.

the non relvent columns are as follows

- Price
    - with this being our target veriable by having price within the dat it create a sqew
- the sales
    - this the output of each product this does not determain the price. this varries from product to product
    - in the above data it also contains a high amount of outliers due to the fact that they very depending on product
- product id
    - this is just how each product is randomly definied and has no impact of the price

then the new defind data is called

```
SelectedColumns=["weight","cpu freq","internal mem","ram","battery","thickness","ppi","cpu core","RearCam","Front_Cam","resoloution"]
MLData=CellData[SelectedColumns]
MLData.head()
```

	weight	cpu freq	internal mem	ram	battery	thickness	ppi	cpu core	RearCam	Front_Cam	resoloution
0	135.0	1.35	16.0	3.000	2610	7.4	424	8	13.00	8.0	5.2
1	125.0	1.30	4.0	1.000	1700	9.9	233	2	3.15	0.0	4.0
2	110.0	1.20	8.0	1.500	2000	7.6	312	4	13.00	5.0	4.7
3	118.5	1.30	4.0	0.512	1400	11.0	233	2	3.15	0.0	4.0
4	125.0	1.30	4.0	1.000	1700	9.9	233	2	3.15	0.0	4.0

Next steps: Generate code with MLData View recommended plots

The to_pickle method serializes the MLData DataFrame. Serialization is the process of converting an object into a format that can be saved to a file or transmitted over a network. In this case, the object is the DataFrame and the format is pickle.

```
# Ensure you include the full path in the file name
MLData.to_pickle(r'C:\CSP_CellData\archive\MLData.pkl')
```

```
MLData.to_pickle('MLData.pkl')
```

the bellow code:

1. Converts from categorical to numeric
    - The pd.get_dummies(MLData) function from the pandas library converts categorical variables in the MLData
    - DataFrame into binary dummy/indicator columns for each unique category value
    - creating a new DataFrame MLData_Numeric suitable for numerical input in machine learning models

```
MLData_Numeric=pd.get_dummies(MLData)
MLData_Numeric['Price']=CellData['Price']
MLData_Numeric.head()
```

	weight	cpu freq	internal mem	ram	battery	thickness	ppi	cpu core	RearCam	Front_Cam	resoloution	Price
0	135.0	1.35	16.0	3.000	2610	7.4	424	8	13.00	8.0	5.2	2357
1	125.0	1.30	4.0	1.000	1700	9.9	233	2	3.15	0.0	4.0	1749
2	110.0	1.20	8.0	1.500	2000	7.6	312	4	13.00	5.0	4.7	1916
3	118.5	1.30	4.0	0.512	1400	11.0	233	2	3.15	0.0	4.0	1315
4	125.0	1.30	4.0	1.000	1700	9.9	233	2	3.15	0.0	4.0	1749

Next steps: Generate code with MLData_Numeric View recommended plots

This code is preparing the dataset for training a machine learning model by defining which features to use, selecting the target variable, and splitting the data into training and testing sets to both train the model and evaluate its performance on unseen data. The use of training and testing sets helps validate the generalization ability of the model.

```
TargetVar='Price'
Predictors=['weight', 'cpu freq', 'internal mem', 'ram', 'battery', 'thickness', 'ppi', 'cpu core', 'RearCam', 'Front_Cam', 'resoloutior

X=MLData_Numeric[Predictors].values
y=MLData_Numeric[TargetVar].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=428)
```

This code prepares the dataset for machine learning by scaling the features to ensure they are on the same scale, which is important for many algorithms that are sensitive to the scale of input data (like gradient descent-based algorithms). It then splits the data into training and testing sets to allow for effective model training and evaluation.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
PredictorScaler=MinMaxScaler()
PredictorScalerFit=PredictorScaler.fit(X)
X=PredictorScalerFit.transform(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

The code you provided consists of print statements that output the shapes of the arrays

```
  X_train, y_train, X_test, and y_test
```

These arrays represent the feature and target data split into training and testing sets.

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(112, 11)
(112,)
(49, 11)
(49,)
```

Section 3

---

within this section:

- Multiple Linear regression alogorithm
- Dession Tree Regression
    - Single dessiob Tree
    - Plotting/visualising the decsion tree
- Random Forest Regressor
    - Plotting on of the Decision Tree in Random Forest Regresson
- Adaboot Alogrithm Building
- XGBoost Regressor
- K_Nearesr Neighbor(KNN)
- SMV Refressor
- Cross Vlaidation

Multiple Linear Regression Algorithm

resgresson analysis: is used to infer or predit another variavle based on the basis of one or more veriables.

Multiple linear regression uses several inderpendant veriables to prodict the dependant veriable that needs to be metric (in this case it is price). the formula bellow is:

simple linear regressionz: one verable podicting another and is the simplifyed formula of the the MLR

y = b.x+a multiple linear regression forumla

y = b1 . x1 + b2 . x2 +...+ bn . xn + a Y is the dependant veriable x1,x2,x3,ect is the inderpendant veriable b1,b2,b3,ect the impact of the independant veriable on the dependant veriable a the intercept or constant veriable

Overall regression analysis, including simple linear regression and multiple linear regression, serves to infer or predict a dependent variable based on one or more independent variables. Multiple linear regression extends this concept by utilizing several independent variables to predict a dependent variable, such as price, through a formula that incorporates coefficients and an intercept term.

```
from sklearn.linear_model import LinearRegression
RegModel = LinearRegression()

print(RegModel)

LREG=RegModel.fit(X_train,y_train)
prediction=LREG.predict(X_test)

from sklearn import metrics
print('R2 Value:',metrics.r2_score(y_train, LREG.predict(X_train)))

print('\n##### Model Validation and Accuracy Calculations ##########')

TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVar]=y_test
TestingDataResults[('Predicted'+TargetVar)]=np.round(prediction)

print(TestingDataResults.head())


TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['Price'])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy)
print('Median Accuracy on test data:', MedianAccuracy)


def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    return(100-MAPE)

from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

from sklearn.model_selection import cross_val_score

Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
LinearRegression()
R2 Value: 0.9425118630708915

##### Model Validation and Accuracy Calculations ##########
   weight  cpu freq  internal mem  ram  battery  thickness    ppi  cpu core  \
0   133.5     1.300           8.0  1.0   1600.0       10.1  218.0       4.0
1    97.0     1.700          16.0  1.0   2100.0        5.1  306.0       8.0
2   112.0     1.400           8.0  1.0   1900.0        6.3  294.0       8.0
3   158.0     1.875          64.0  6.0   3000.0        7.4  401.0       4.0
4   180.0     1.750          32.0  3.0   3430.0        7.8  806.0       8.0

   RearCam  Front_Cam  resoloution  Price  PredictedPrice
0      5.0        0.0          4.5   1843          1626.0
1      8.0        5.0          4.8   2323          2502.0
2      8.0        8.0          5.0   2266          2289.0
3     16.0        8.0          5.5   3055          3061.0
4     23.0        5.1          5.5   3102          3204.0
Mean Accuracy on test data: 91.21568919554772
Median Accuracy on test data: 92.29444683598794

Accuracy values for 10-fold Cross Validation:
 [92.0637325  90.10099884 91.38501617 94.02519619 92.98905199 90.80833026
 93.0783774  90.83855461 90.51159746 91.24017577]

Final Average Accuracy of the model: 91.7
```

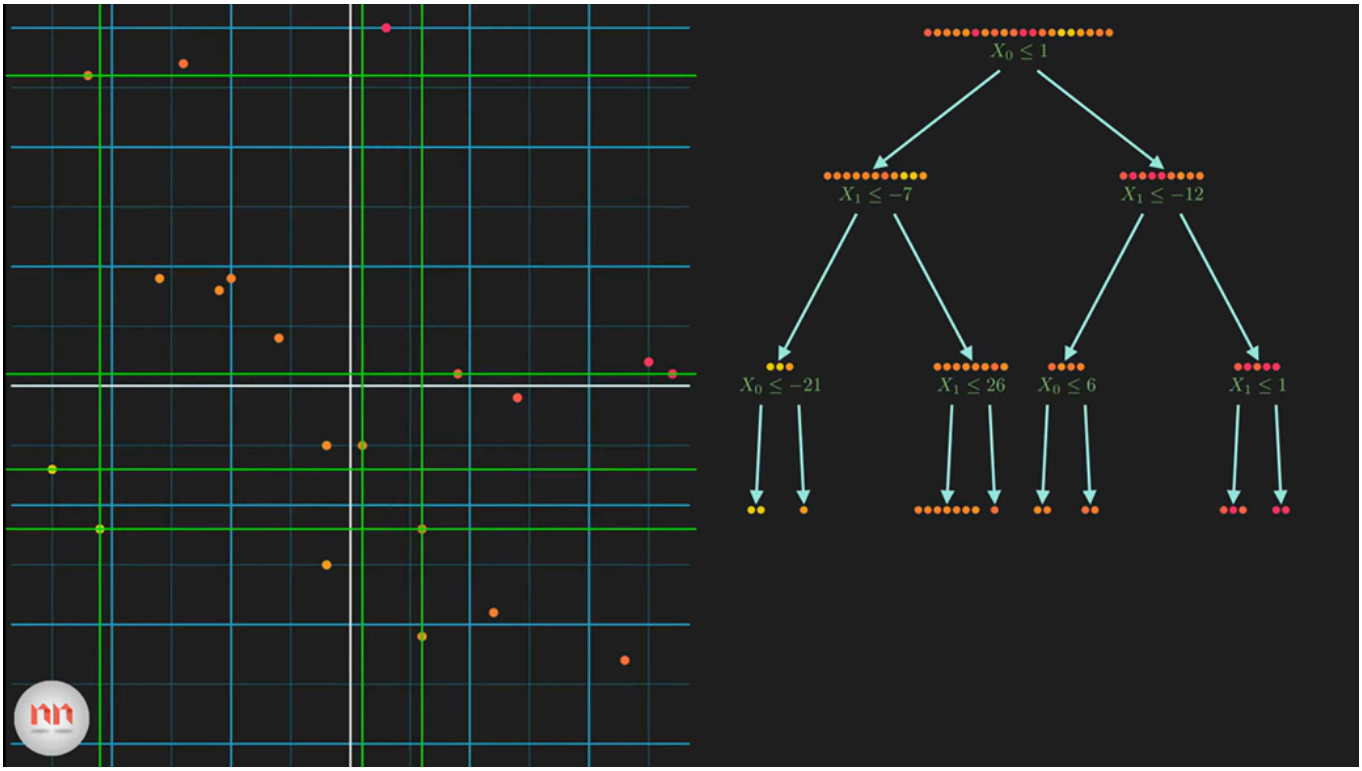
Explaining the MLR algorithm code:

1. sets yp the LRM
2. Fitting the Model & making Prodictions
3. evaluating the model
4. model validation & Accuracy Calculator
5. Custom Acuracy function & cross Validation

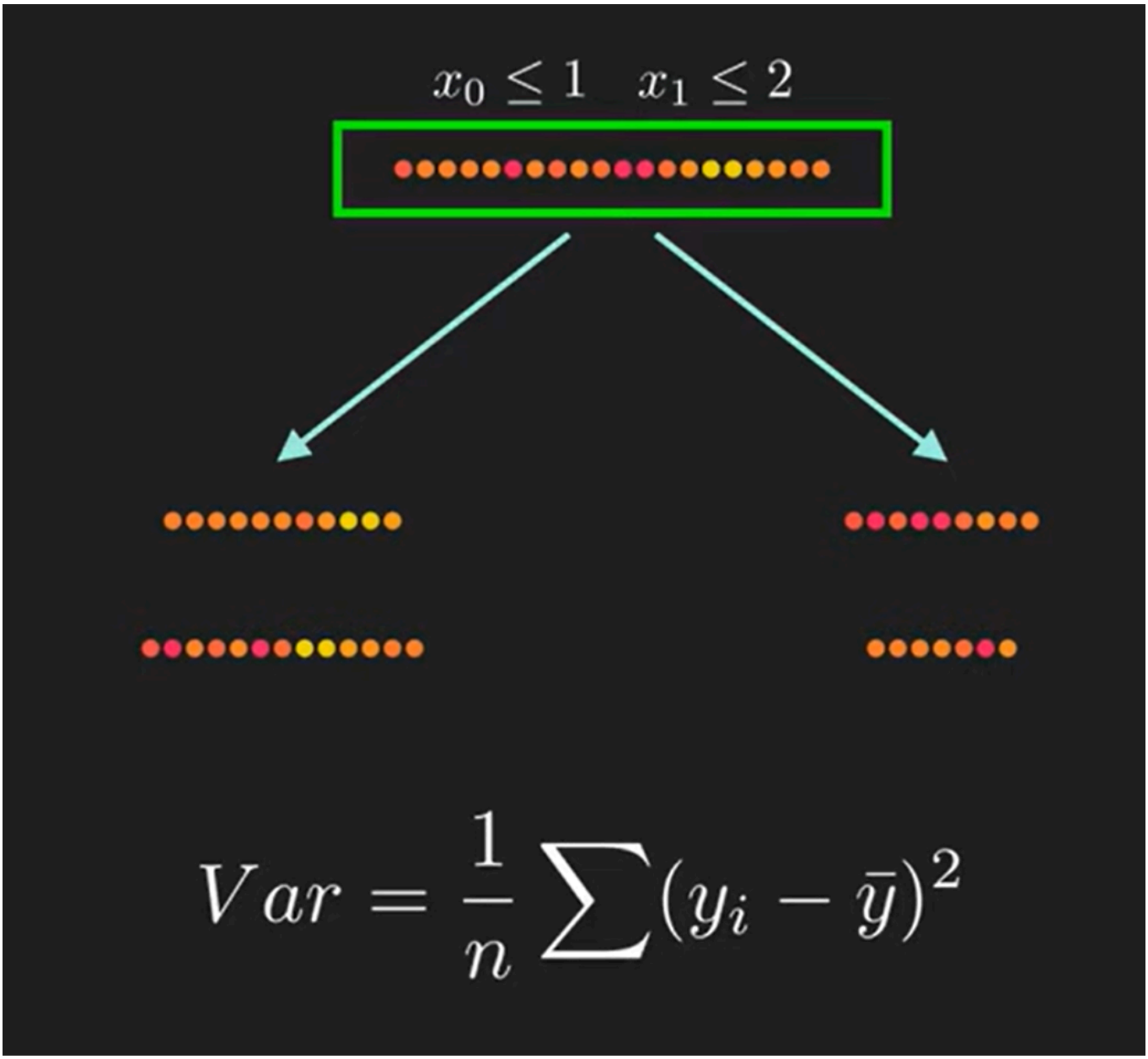
Decision Tree Regressor

A Decision Tree Regressor is a type of decision tree model specifically used for regression tasks, where the goal is to predict a continuous outcome variable (dependent variable) based on one or more predictor variables (independent variables).

the following image shows a 2d represenation of a 3d regregssoin with the yellow being a lower numbers then the red being the highest allowing for some observation of the 3rd axi. within the graph there is several green lines these are the splitting points for the desion tree breaking up the data. the visual representation of the dession tree shows the root nod splitting into secondary nodes and at what point they split up.



the way you calculate a dession tree is through the use of veriance reduction which help us to determain the inpurity. the following formula looks at finding the variants for the whole formula:



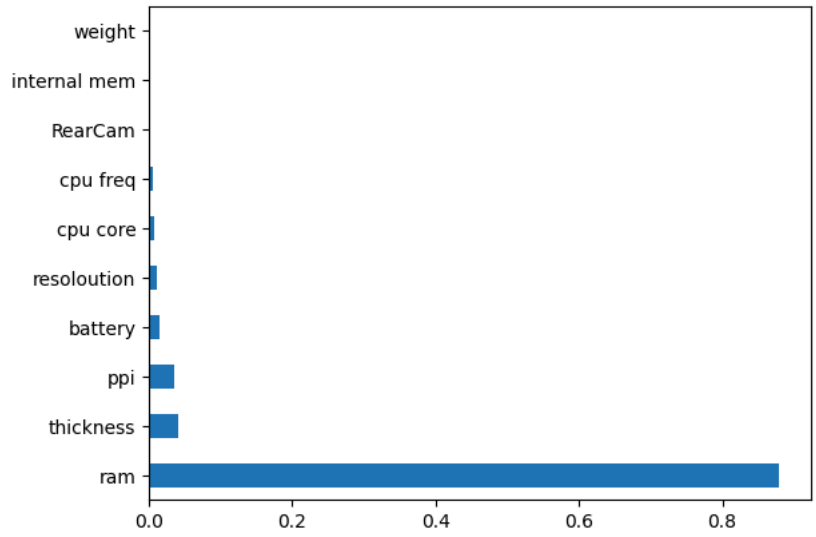
the bellow formula is the prossess of taking the secondary data sets and comparing to the root to find the best fitting models (this is done continuosely and automticaly when coding till the ideal model is found)

$$\text{Var Red} = Var(parent) - \sum w_i \, Var(child_i)$$

fore more detatis: https://www.youtube.com/watch?v=UhY5vPfQIrA&t=54s

```
# Decision Trees (Multiple if-else statements!)
from sklearn.tree import DecisionTreeRegressor
RegModel = DecisionTreeRegressor(max_depth=5,criterion='friedman_mse')
# Good Range of Max_depth = 2 to 20

# Printing all the parameters of Decision Tree
print(RegModel)

# Creating the model on Training Data
DT=RegModel.fit(X_train,y_train)
prediction=DT.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, DT.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(DT.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVar]=y_test
TestingDataResults[('Predicted'+TargetVar)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['Price'])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

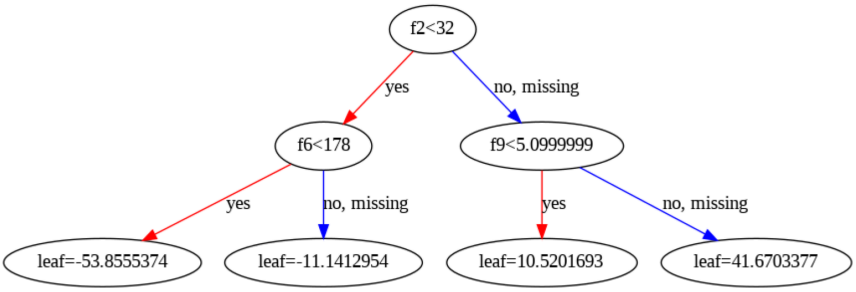
```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=5)
R2 Value: 0.9664023076213232

##### Model Validation and Accuracy Calculations ##########
   weight  cpu freq  internal mem  ram  battery  thickness    ppi  cpu core  \
0   133.5     1.300           8.0  1.0   1600.0       10.1  218.0       4.0
1    97.0     1.700          16.0  1.0   2100.0        5.1  306.0       8.0
2   112.0     1.400           8.0  1.0   1900.0        6.3  294.0       8.0
3   158.0     1.875          64.0  6.0   3000.0        7.4  401.0       4.0
4   180.0     1.750          32.0  3.0   3430.0        7.8  806.0       8.0

   RearCam  Front_Cam  resoloution  Price  PredictedPrice
0      5.0        0.0          4.5   1843          1638.0
1      8.0        5.0          4.8   2323          2266.0
2      8.0        8.0          5.0   2266          2266.0
3     16.0        8.0          5.5   3055          3055.0
4     23.0        5.1          5.5   3102          2732.0
Mean Accuracy on test data: 90.1489790518639
Median Accuracy on test data: 94.75696340797379

Accuracy values for 10-fold Cross Validation:
 [87.72532766 91.53036646 88.38378255 88.54716437 92.13588358 90.792989
 94.41769215 94.56258904 86.06221969 86.45753372]

Final Average Accuracy of the model: 90.06
```



Single Desision Tree

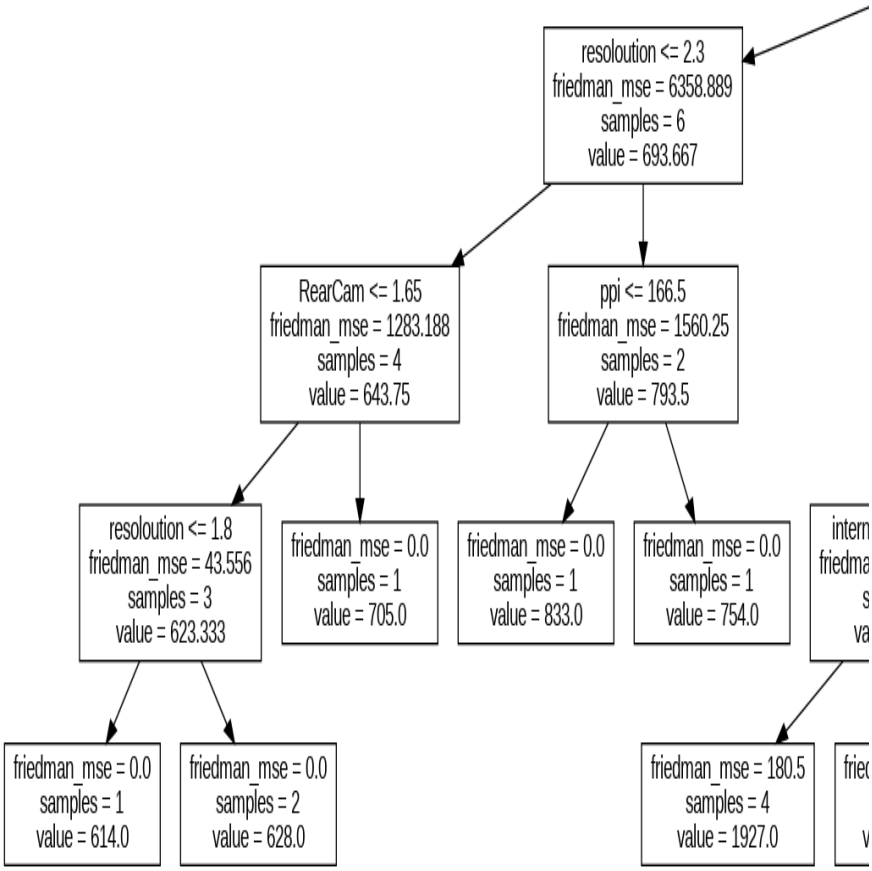
```
#Plotting a single Decision tree out of XGBoost
from xgboost import plot_tree
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20, 8))
plot_tree(XGB, num_trees=10, ax=ax)
```

<Axes: >



Plotting/Visualizing the Decision Tree

```
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data
dot_data = tree.export_graphviz(RegModel, out_file=None,feature_names=Predictors, class_names=TargetVar)

# printing the rules
#print(dot_data)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=3700,height=1000)
# Double click on the graph to zoom in
```

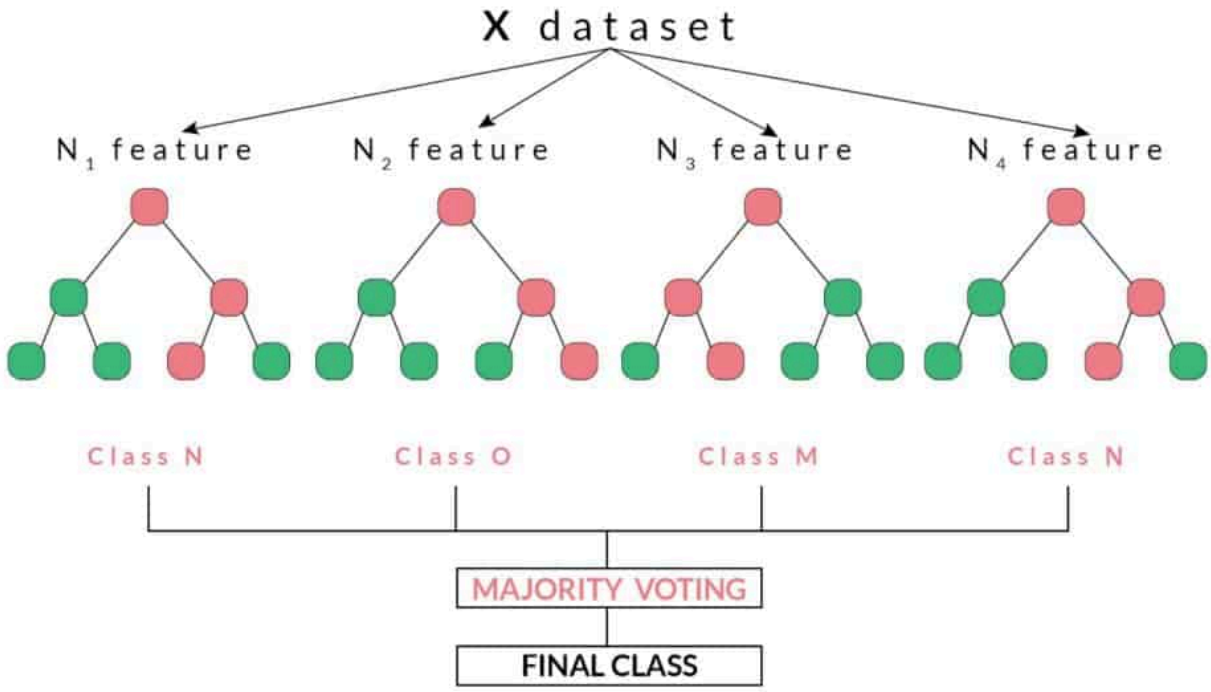


the above show the visual reprsentation of cellphone price prodiction desion tree regression. looking at how the data has been split helps us to understand where different values go.

Random Foresr Regressor

Defintion: random forest regression it is a tequnice that uses both classification and regressiob tasks. "The algorithm fits a number of decision tree regressors on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

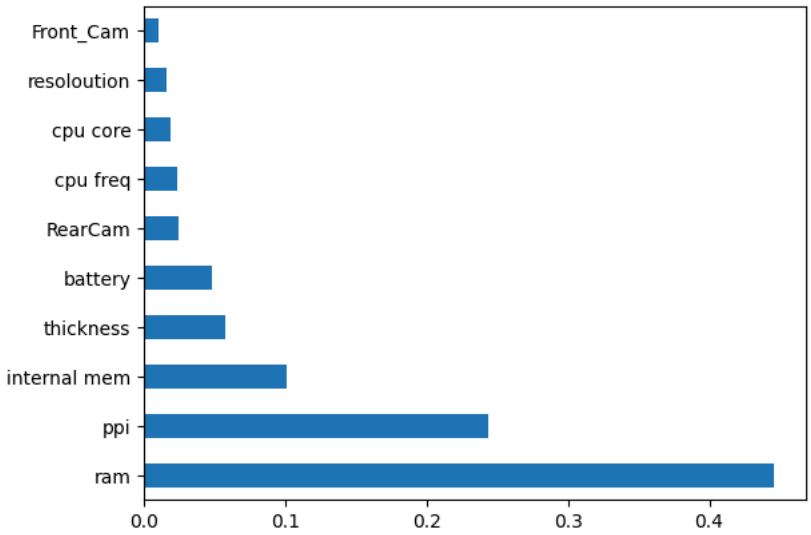
Equations:




```
# Random Forest (Bagging of multiple Decision Trees)
from sklearn.ensemble import RandomForestRegressor
RegModel = RandomForestRegressor(max_depth=4, n_estimators=400,criterion='friedman_mse')
# Good range for max_depth: 2-10 and n_estimators: 100-1000

# Printing all the parameters of Random Forest
print(RegModel)

# Creating the model on Training Data
RF=RegModel.fit(X_train,y_train)
prediction=RF.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, RF.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVar]=y_test
TestingDataResults[('Predicted'+TargetVar)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['Price'])
MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)
```

```
# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

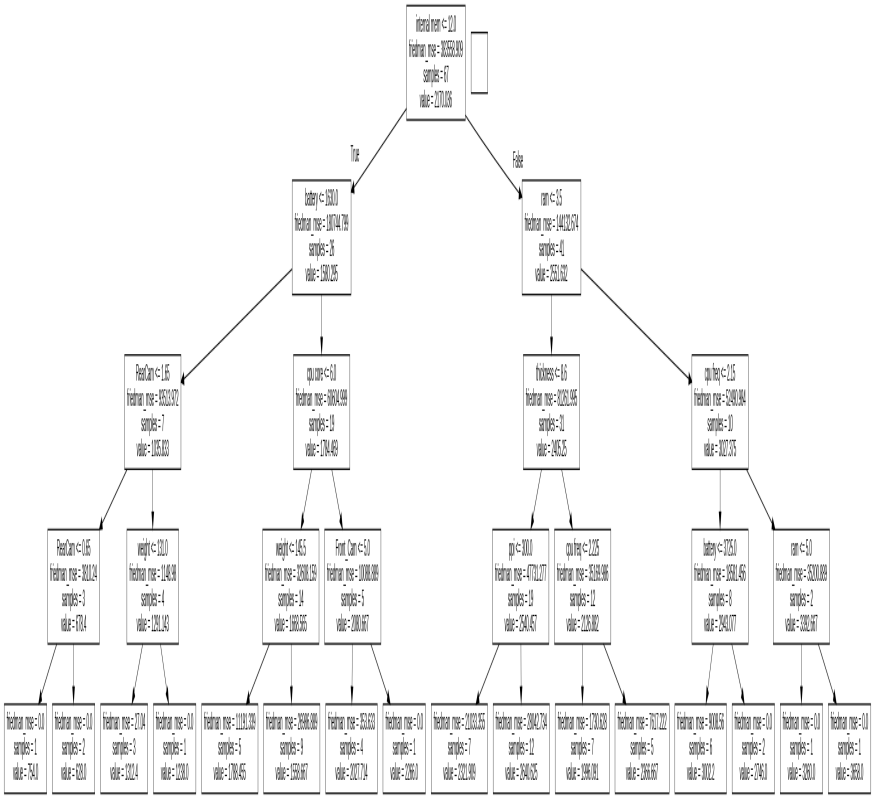
```
RandomForestRegressor(criterion='friedman_mse', max_depth=4, n_estimators=400)
R2 Value: 0.9702246327916387

##### Model Validation and Accuracy Calculations ##########
   weight  cpu freq  internal mem  ram  battery  thickness    ppi  cpu core  \
0   133.5     1.300           8.0  1.0   1600.0       10.1  218.0       4.0
1    97.0     1.700          16.0  1.0   2100.0        5.1  306.0       8.0
2   112.0     1.400           8.0  1.0   1900.0        6.3  294.0       8.0
3   158.0     1.875          64.0  6.0   3000.0        7.4  401.0       4.0
4   180.0     1.750          32.0  3.0   3430.0        7.8  806.0       8.0

   RearCam  Front_Cam  resoloution  Price  PredictedPrice
0      5.0        0.0          4.5   1843          1689.0
1      8.0        5.0          4.8   2323          2199.0
2      8.0        8.0          5.0   2266          2057.0
3     16.0        8.0          5.5   3055          3232.0
4     23.0        5.1          5.5   3102          2786.0
Mean Accuracy on test data: 92.06217711108663
Median Accuracy on test data: 93.82391590013141

Accuracy values for 10-fold Cross Validation:
 [89.54817478 91.91673299 89.71986851 89.71086215 92.49874673 92.93949083
 93.14192064 92.03690448 88.82566041 88.07353217]

Final Average Accuracy of the model: 90.84
```



Plotting on of the Decision Tree in Random Forest Regressor

the bellow is the code that generates the trees within the forst aka the equations for the forest to help prodict the price.

```
# Plotting a single Decision Tree from Random Forest
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Random Forest
dot_data = tree.export_graphviz(RegModel.estimators_[5] , out_file=None, feature_names=Predictors, class_names=TargetVar)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=2500,height=600)
# Double click on the graph to zoom in
```



Adaboot Alogrithm Building

Defintion: this is one of the first booster algorithms, it is a clasification based leating it is maily a one level decision tree and often comes with mainy waited errrors.

https://www.machinelearningplus.com/machine-learning/introduction-to-adaboost/#:~:text=AdaBoost%20is%20one%20of%20the%20first%20boosting%20algorithms,strong%20classifier%20from%20a%20series%20of%20weak%20classifiers.

the following is the formula https://math.stackexchange.com/questions/3778238/understanding-adaboost-algorithm

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.