Exploratory Data Analysis for Machine Learning

IBM Skills Network

Project Report

By:

Fahd Seddik

Table of Contents

Brief Description

Mobile price depends on various factors such as resolution, brand, size, weight, imaging quality, RAM, battery and CPU power. In this dataset, we want to estimate the price of mobile phones using the above features.

Columns:

```
data.head()
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	203	2357	10	135.0	5.2	424	8	1.35	16.0	3.000	13.00	8.0	2610	7.4
1	880	1749	10	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9
2	40	1916	10	110.0	4.7	312	4	1.20	8.0	1.500	13.00	5.0	2000	7.6
3	99	1315	11	118.5	4.0	233	2	1.30	4.0	0.512	3.15	0.0	1400	11.0
4	880	1749	11	125.0	4.0	233	2	1.30	4.0	1.000	3.15	0.0	1700	9.9

Initial plan

The plan would go as follows:

- Check for duplicates and deal with any
- Check for missing values and deal with any
- Calculate correlation values
- Check for skewness of data
- Visualize through boxplots to check for outliers
- Apply feature engineering to formulate possible useful features
- Use seaborn pair plots to see underlying patterns
- Construct hypothesis about data set

Data cleaning & Feature engineering

We will first sort by Product_id to make data more readable

```
d = data.copy()
d.sort_values(by='Product_id',inplace=True)
d.reset_index(inplace=True)
d.drop('index',axis=1,inplace=True)
d
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	10	1950	26	118.0	5.0	187	4	1.300	8.0	1.000	8.0	2.0	2000	6.4
1	14	2276	91	116.0	5.0	294	8	1.500	16.0	2.000	13.0	5.0	2300	7.8
2	14	2276	98	116.0	5.0	294	8	1.500	16.0	2.000	13.0	5.0	2300	7.8
3	30	2975	307	149.0	5.5	534	8	1.600	32.0	3.000	16.0	8.0	3000	7.0
4	30	2975	302	149.0	5.5	534	8	1.600	32.0	3.000	16.0	8.0	3000	7.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
156	1296	3211	8946	170.0	5.5	534	4	1.975	128.0	6.000	20.0	8.0	3400	7.9
157	1327	2001	393	194.8	5.7	258	4	1.200	16.0	2.000	8.0	1.0	3400	10.2
158	1327	2001	399	194.8	5.7	258	4	1.200	16.0	2.000	8.0	1.0	3400	10.2
159	1339	1421	40	120.0	4.0	233	2	1.000	4.0	0.512	2.0	0.0	1200	9.8
160	1339	1421	31	120.0	4.0	233	2	1.000	4.0	0.512	2.0	0.0	1200	9.8

Check for any duplicates

```
d.Product_id.is_unique
```

```
False
```

We will drop rows with duplicate ids

```
d.drop_duplicates(subset='Product_id',inplace=True)
d
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
0	10	1950	26	118.0	5.00	187	4	1.300	8.0	1.000	8.0	2.0	2000	6.4
1	14	2276	91	116.0	5.00	294	8	1.500	16.0	2.000	13.0	5.0	2300	7.8
3	30	2975	307	149.0	5.50	534	8	1.600	32.0	3.000	16.0	8.0	3000	7.0
5	32	1921	1781	179.0	6.00	184	4	1.300	8.0	1.000	13.0	8.0	2580	8.0
7	40	1916	10	110.0	4.70	312	4	1.200	8.0	1.500	13.0	5.0	2000	7.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
151	1221	2714	106	156.0	5.50	401	8	1.350	16.0	2.000	13.0	5.0	2300	5.1
153	1248	3658	52	168.0	5.15	428	8	2.450	64.0	6.000	12.0	8.0	3350	7.5
155	1296	3211	8016	170.0	5.50	534	4	1.975	128.0	6.000	20.0	8.0	3400	7.9
157	1327	2001	393	194.8	5.70	258	4	1.200	16.0	2.000	8.0	1.0	3400	10.2
159	1339	1421	40	120.0	4.00	233	2	1.000	4.0	0.512	2.0	0.0	1200	9.8

83 rows × 14 columns

Check for missing values

```
d.info()
```

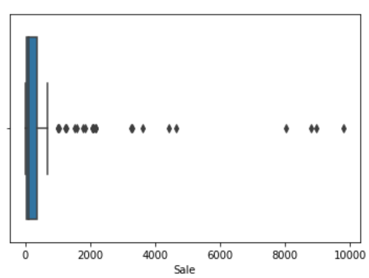
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 83 entries, 0 to 159
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Product_id    83 non-null     int64
 1   Price         83 non-null     int64
 2   Sale          83 non-null     int64
 3   weight        83 non-null     float64
 4   resoloution   83 non-null     float64
 5   ppi           83 non-null     int64
 6   cpu core      83 non-null     int64
 7   cpu freq      83 non-null     float64
 8   internal mem  83 non-null     float64
 9   ram           83 non-null     float64
 10  RearCam       83 non-null     float64
 11  Front_Cam     83 non-null     float64
 12  battery       83 non-null     int64
 13  thickness     83 non-null     float64
dtypes: float64(8), int64(6)
memory usage: 9.7 KB
```

Since number of entries is 83 and all columns contain 83 non-null entries then we do not have any missing values.

Check for outliers

```
sns.boxplot(x=data['Sale'])
```

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




```
len(data[data['Sale']>1700])
```

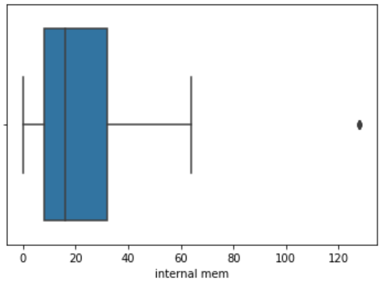
```
17
```

Will drop outliers in Sale column. On the next page, there will be box plots for all features.

```
d.drop(d[d['Sale']>1700].index,inplace=True)
```

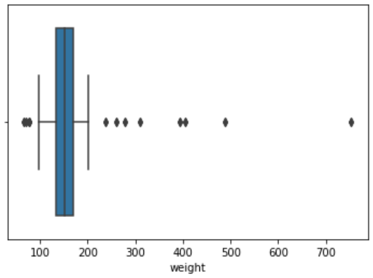
```python
sns.boxplot(x=data['internal mem'])
```

```
<AxesSubplot:xlabel='internal mem'>
```



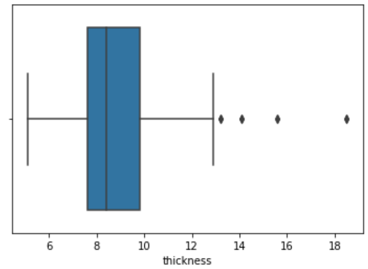

```python
sns.boxplot(x=data['weight'])
```

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



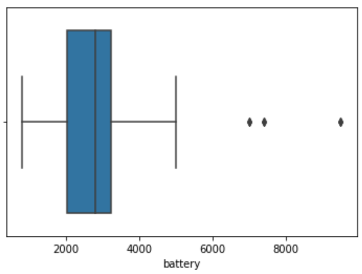

```python
sns.boxplot(x=data['thickness'])
```

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



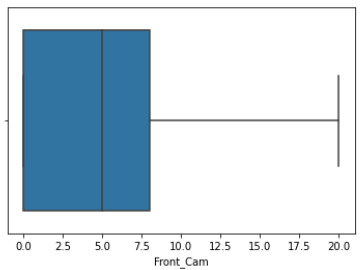

```python
sns.boxplot(x=data['battery'])
```

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



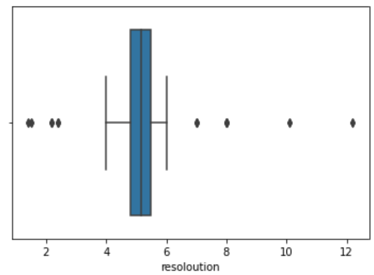

```python
sns.boxplot(x=data['Front_Cam'])
```

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



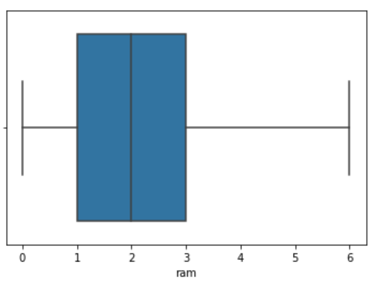

```python
sns.boxplot(x=data['resoloution'])
```

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



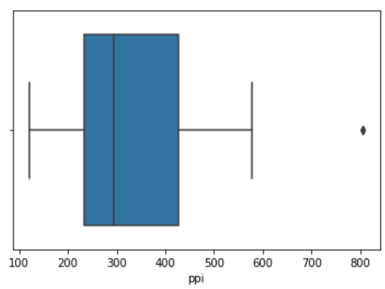

```python
sns.boxplot(x=data['ram'])
```

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



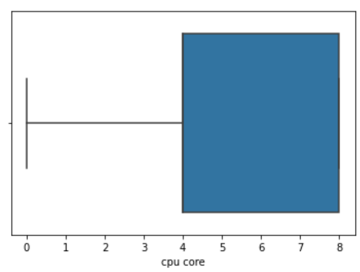

```python
sns.boxplot(x=data['ppi'])
```

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



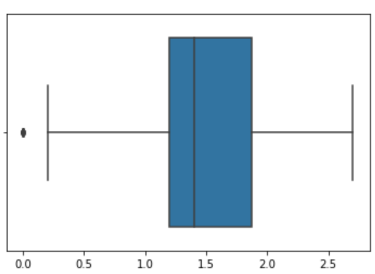

```python
sns.boxplot(x=data['cpu core'])
```

```
<AxesSubplot:xlabel='cpu core'>
```



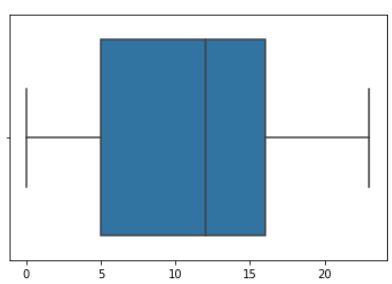

```python
sns.boxplot(x=data['cpu freq'])
```

```
<AxesSubplot:xlabel='cpu freq'>
```




```python
sns.boxplot(x=data['RearCam'])
```

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



Calculate each skew value

```python
float_cols = d.columns
skew_limit = 0.75
skew_vals = d[float_cols[1:]].skew()
skew_cols = (skew_vals.sort_values(ascending=False)
             .to_frame().rename(columns={0:'Skew'}).query('abs(Skew) > {}'.format(skew_limit)))
skew_cols
```

	Skew
weight	4.077420
Sale	4.074989
internal mem	2.407779
battery	2.151937
thickness	1.630595
Front_Cam	1.250122
resoloution	1.210662
ram	0.801043

We will apply **log transformation** to all skewed columns.

```python
to_skew = ['weight','Sale','internal mem','battery','thickness','Front_Cam','resoloution','ram']
for i in to_skew:
    d[i] = np.log1p(d[i]);
```

```python
float_cols = d.columns
skew_limit = 0.75
skew_vals = d[float_cols[1:]].skew()
skew_cols = (skew_vals.sort_values(ascending=False)
             .to_frame().rename(columns={0:'Skew'}).query('abs(Skew) > {}'.format(skew_limit)))
skew_cols
```

	Skew
weight	1.382336
resoloution	-1.021314

After applying log transformation to our columns, skewness values are mostly corrected. We end up with only 2 columns with skewed values instead of the initial 8 columns.

Feature Engineering

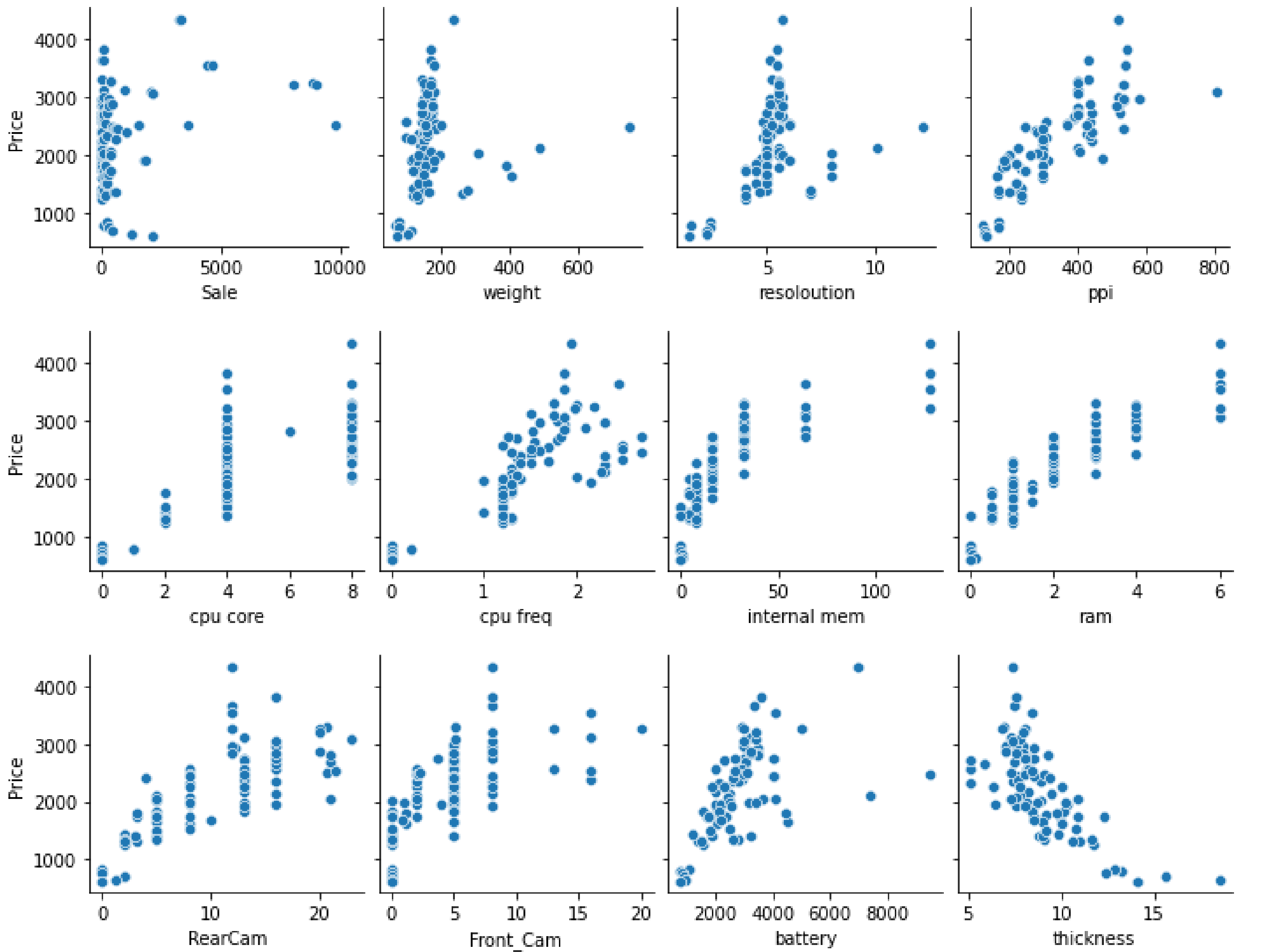
Since thickness is more likely to have an inverse relation to price (would be shown in next section), then we can add a 1/thickness feature.

There are some other notable feature interactions that should be mentioned:

- Weight_/thickness
- Internal mem_*_ram
- RearCam_*_Front_cam

Key Findings and Insights

```
for i in range(2,14,4):
    sns.pairplot(data=data_num,x_vars=data_num.columns[i:i+4],y_vars=['Price']);
```



Calculate statistics (before log transformation)

```
stats_df = d.describe()
stats_df.loc['range'] = stats_df.loc['max'] - stats_df.loc['min']

out_fields = ['mean','25%','50%','75%', 'range']
stats_df = stats_df.loc[out_fields]
stats_df.rename({'50%': 'median'}, inplace=True)
stats_df
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
mean	665.638554	2219.084337	201.066667	169.226506	5.206024	334.710843	4.879518	1.501843	24.43841	2.201831	10.469277	4.608434	2826.445783	8.872289
25%	227.500000	1737.500000	33.500000	134.050000	4.900000	233.000000	4.000000	1.200000	8.00000	1.000000	5.000000	0.450000	2020.000000	7.600000
median	763.000000	2258.000000	90.000000	152.000000	5.150000	294.000000	4.000000	1.400000	16.00000	2.000000	12.000000	5.000000	2700.000000	8.400000
75%	1023.000000	2744.000000	235.500000	170.000000	5.500000	428.000000	8.000000	1.875000	32.00000	3.000000	16.000000	8.000000	3220.000000	9.750000
range	1329.000000	3747.000000	1574.000000	687.000000	10.800000	685.000000	8.000000	2.700000	128.00000	6.000000	23.000000	20.000000	8700.000000	13.400000

Calculate statistics (after log transformation)

```
stats_df = d.describe();
stats_df.loc['range'] = stats_df.loc['max'] - stats_df.loc['min'];

out_fields = ['mean','25%','50%','75%', 'range'];
stats_df = stats_df.loc[out_fields];
stats_df.rename({'50%': 'median'}, inplace=True);
stats_df
```

	Product_id	Price	Sale	weight	resoloution	ppi	cpu core	cpu freq	internal mem	ram	RearCam	Front_Cam	battery	thickness
mean	665.638554	2219.084337	4.499836	5.037165	1.781928	334.710843	4.879518	1.501843	2.621478	1.014924	10.469277	1.317973	7.827422	2.274214
25%	227.500000	1737.500000	3.510542	4.905275	1.749162	233.000000	4.000000	1.200000	2.197225	0.693147	5.000000	0.000000	7.601402	2.151762
median	763.000000	2258.000000	4.499810	5.030438	1.808289	294.000000	4.000000	1.400000	2.833213	1.098612	12.000000	1.791759	7.855932	2.251292
75%	1023.000000	2744.000000	5.457526	5.138731	1.871802	428.000000	8.000000	1.875000	3.496508	1.386294	16.000000	1.808289	8.071219	2.393329
range	1329.000000	3747.000000	4.970444	2.420700	1.704748	685.000000	8.000000	2.700000	4.859812	1.945910	23.000000	2.833213	2.473291	1.162126

Insights

We can tell from the pair plots there are many features how have a positive correlation with price of phone. Let's calculate the correlation values.

+ Code + Markdown

2]:

```
# Calculate correlation values
data_num = d.select_dtypes(include = ['float64','int64'])
corr = data_num.corr()['Price'][2:]
top_features = corr[abs(corr) > 0.5].sort_values(ascending=False)
print("{} Strongly correlated values : \n{}".format(len(top_features),top_features))
```

```
4 Strongly correlated values :
ppi         0.814855
RearCam     0.740738
cpu freq    0.729808
cpu core    0.688402
Name: Price, dtype: float64
```

Hypothesis

We can hypothesize about the data set in several ways. Here are some of the hypotheses we can have about our data set.

1. H_o: A weight of range 140 to 160 represent 50% of examples

   H_a: weight range 140 to 160 does not represent 50% of examples

2. H_o: 90% of phones with 8 cores have same range of 4 core phones

   H_a: they do not have the same range

3. H_o: all phones around the world have thickness with mu = 10.98 (sample mu)

   H_a: mu !=10.98

We will be conducting a formal significance test for the third hypothesis. Since our sample mu = 10.98 we will calculate t_value, z_value and having a significance level of alpha=0.05. This would give us a t_value of 2.00 and z_value of 0.4798.

Suggestions

Of course, the analysis we did on the data set is merely scraping the surface of all possible analysis methods we can apply to this data set. We can try to formulate more features by applying feature engineering. In addition to that, we can visualize correlation values using heatmaps. Feature scaling could be one of the methods we would use if we are intending on using the data set in models that are prone to not scaled features. Calculation of z-score could be one of the ways to determine more statistics about our data set.

Summary

Predicting the price of phones could help companies compete by just choosing the phone's specifications and estimating how much the phone would sell for and calculate their budget.

In conclusion, I believe that there is much potential in this data set. Although further EDA could be done on this data set and fine-tune it better, but we managed to stick to the initial plan.