Mobile Class Classification

Table of contents

About tha Data:

Data of Mobile Phones of Various Companies

Collected sales data of mobile phones of various companies in order to find out some relation between features of a mobile phone(e.g: RAM, Internal Memory, ...etc.) and its selling price.

Data Source Link: Kaggle (Mobile Class Classification)

Download link: Mobile Class Classification Download

The source site page contains two datasets (`test.csv` and `train.csv`). We will deal only with train data for this time.

The dataset contains 2000 row and each row consist of 21 columns as follows:

- **battery_power:** Total energy a battery can store in one time measured in mAh
- **blue:** Has bluetooth or not
- **clock_speed:** speed at which microprocessor executes instructions
- **dual_sim:** Has dual sim support or not
- **fc:** Front Camera mega pixels
- **four_g:** Has 4G or not
- **int_memory:** Internal Memory in Gigabytes
- **m_dep:** Mobile Depth in cm
- **mobile_wt:** Weight of mobile phone
- **n_cores:** Number of cores of processor
- **pc:** Primary Camera mega pixels
- **px_height:** Pixel Resolution Height
- **px_width:** Pixel Resolution Width
- **ram:** Random Access Memory in Megabytes
- **sc_h:** Screen Height of mobile in cm
- **sc_w:** Screen Width of mobile in cm
- **talk_time:** Longest time that a single battery charge will last when used for talking
- **three_g:** Has 3G or not
- **touch_screen:** Has touch screen or not
- **wifi:** Has wifi or not
- **price_range:** Which is our target feature

First, We read the csv file into dataframe using `pandas` 'read_csv' function.Then, We make a copy of the dataset. After that, we have a quick look over the first 10 rows in the dataframe using head() method.

```
import pandas as pd

# reading dataset of file 'train.csv' and store it in a dataframe 'df'
df = pd.read_csv('train.csv')

data = df   # making a copy

data.head(10)   # showing first 10 rows in dataframe
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212
5	1859	0	0.5	1	3	0	22	0.7	164	1	...	1004	1654
6	1821	0	1.7	0	4	1	10	0.8	139	8	...	381	1018
7	1954	0	0.5	1	0	0	24	0.8	187	4	...	512	1149
8	1445	1	0.5	0	0	0	53	0.7	174	7	...	386	836
9	509	1	0.6	1	2	1	9	0.1	93	5	...	1137	1224

10 rows × 21 columns

> ***Back to the top***

Initial Data Exploration:

Here we use 'info()' method to get a short summary of data features values. We notice that we haven't any null value, and all dataframe columns (features) we have are numeric. In fact, some of the columns represents categorical variables(e.g.: three_g, four_g, wifi, ...etc), but they are already encoded for us from the data source.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
```

As we can see we have a clean (cleaned) dataset that contains no columns with `NaN` value to be dropped or replaced with suitable value, But columns names need to be replaced with more human-readable names. So, let's do that:

```
new_names = ['Battery Power', 'has Bluetooth', 'Clock Speed', 'Dual Sim', 'Front Camera Mpx', 'Supports 4G', 'Internal Memory', 'Mobile Depth in cm',
             'Mobile Weight', 'Cores Number', 'Primary Camera Mpx', 'Pixel Resolution Height', 'Pixel Resolution Width', 'Ram', 'Screen Height in cm',
             'Screen Width in cm', 'Talk Time', 'Supports 3G', 'Touch Screen', 'Wifi',
             'Price Range']
names = {}
for i, col in enumerate(data):
    names[col] = new_names[i]
data.rename(columns=names, inplace=True)
data.head()
```

	Battery Power	has Bluetooth	Clock Speed	Dual Sim	Front Camera Mpx	Supports 4G	Internal Memory	Mobile Depth in cm	Mobile Weight	Cores Number	...	Pixel Resolution Height	Pixel Resolution Width	Ram
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411

5 rows × 21 columns

Now, Let's take a look at summary statistics of data using 'describe()' method from `pandas` library:

```
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Battery Power	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
has Bluetooth	2000.0	0.49500	0.500100	0.0	0.00	0.0	1.00	1.0
Clock Speed	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
Dual Sim	2000.0	0.50950	0.500035	0.0	0.00	1.0	1.00	1.0
Front Camera Mpx	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
Supports 4G	2000.0	0.52150	0.499662	0.0	0.00	1.0	1.00	1.0
Internal Memory	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
Mobile Depth in cm	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
Mobile Weight	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
Cores Number	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
Primary Camera Mpx	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
Pixel Resolution Height	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
Pixel Resolution Width	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
Ram	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
Screen Height in cm	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
Screen Width in cm	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
Talk Time	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
Supports 3G	2000.0	0.76150	0.426273	0.0	1.00	1.0	1.00	1.0
Touch Screen	2000.0	0.50300	0.500116	0.0	0.00	1.0	1.00	1.0
Wifi	2000.0	0.50700	0.500076	0.0	0.00	1.0	1.00	1.0
Price Range	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

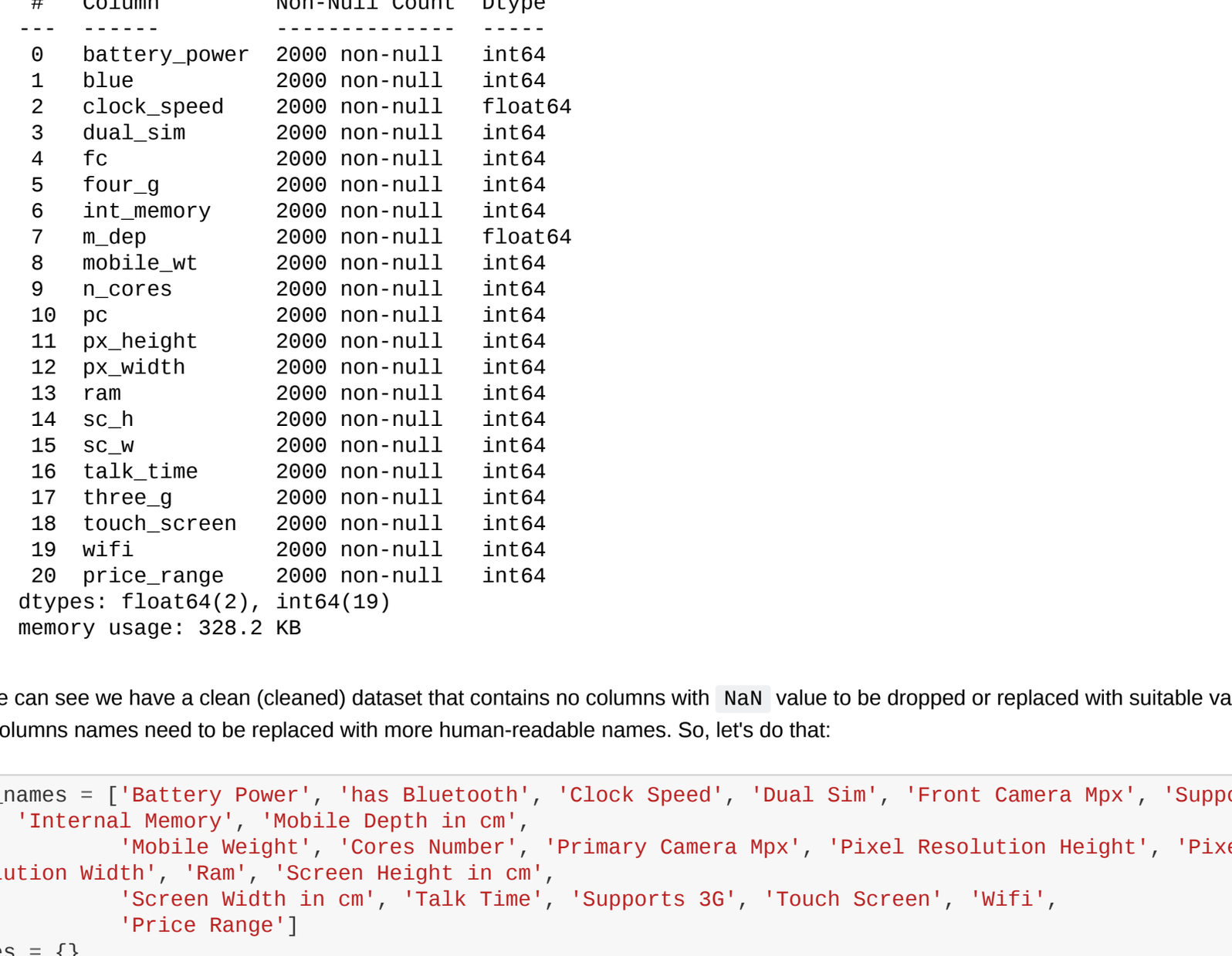
Instead, We can see a visual representation of how data distribution looks like. but first, let's import needed library:

```
# importing required python packages
# for plotting
import seaborn as sns
import matplotlib.pyplot as plt
# for mathmatical operations on data
import numpy as np
# for data normalization
from sklearn.preprocessing import StandardScaler as stds
%matplotlib inline
```

```
target_feature = 'Price Range'
plot_data = (data
             .set_index(target_feature)
             .stack()
             .to_frame()
             .reset_index()
             .rename(columns={0: 'value', 'level_1': 'features'})
             )

f = plt.figure(figsize=(30, 20))
sns.boxplot(x='features', y='value',
            hue=target_feature, data=plot_data)
```

```
<AxesSubplot:xlabel='features', ylabel='value'>
```

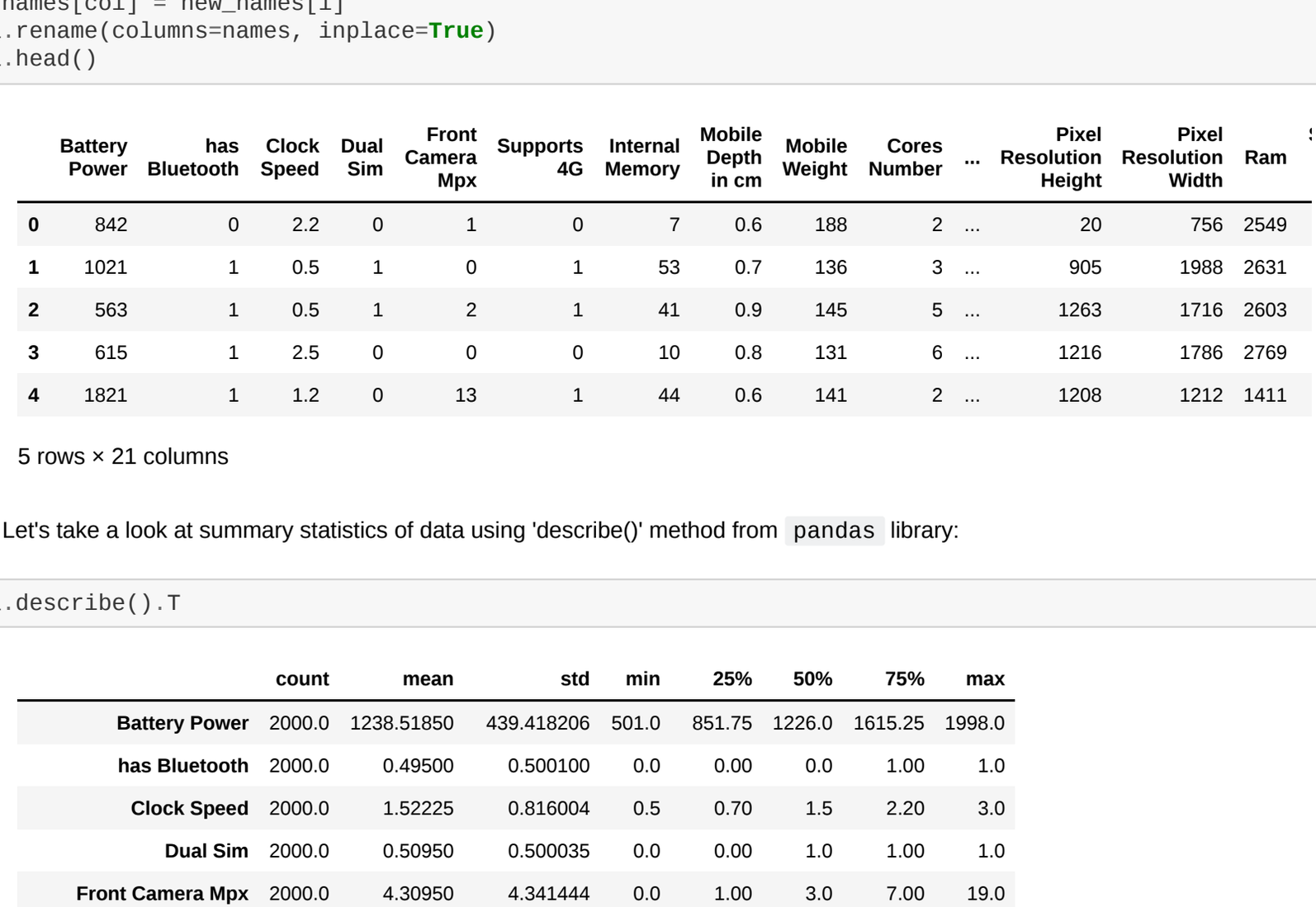


As we see above in the box plots, only some features (**Battery Power**, **Pixel Resolution Height**, **Pixel Resolution Width** and **Ram**) seems to have in some relation with **Price Range**, and the most obvious one is in the **Ram** memory's box plot.To make it clearer we can draw a scatter plot (or better a bar plot since we have categorical variable) for these 4 features , but first, let's verify other features don't affect selling price by comparing their normalized values. For this purpose we use 'StandardScaler' function ('stds' as we called before) from `seaborn` library.

```
# scaling data
data_scaled = pd.DataFrame(stds().fit_transform(data), columns=data.columns)
```

```
target_feature = 'Price Range'
plot_data = (data_scaled
             .set_index(target_feature)
             .stack()
             .to_frame()
             .reset_index()
             .rename(columns={0: 'value', 'level_1': 'features'})
             )
f = plt.figure(figsize=(30, 20))
sns.boxplot(x='features', y='value',
            hue=target_feature, data=plot_data)
```

```
<AxesSubplot:xlabel='features', ylabel='value'>
```



The scaled data box plots shows that other features aren't important factors in determining **Price Range** with the exception of **Mobile Weight** (which slightly affect the price in inverse way) and mobile **Support 3G** (here we may have biased data and/or because 3G is becoming outdated feature resulting in relation with low **Price Range**).

> ***Back to the top***

Data Cleaning and Feature Engineering:

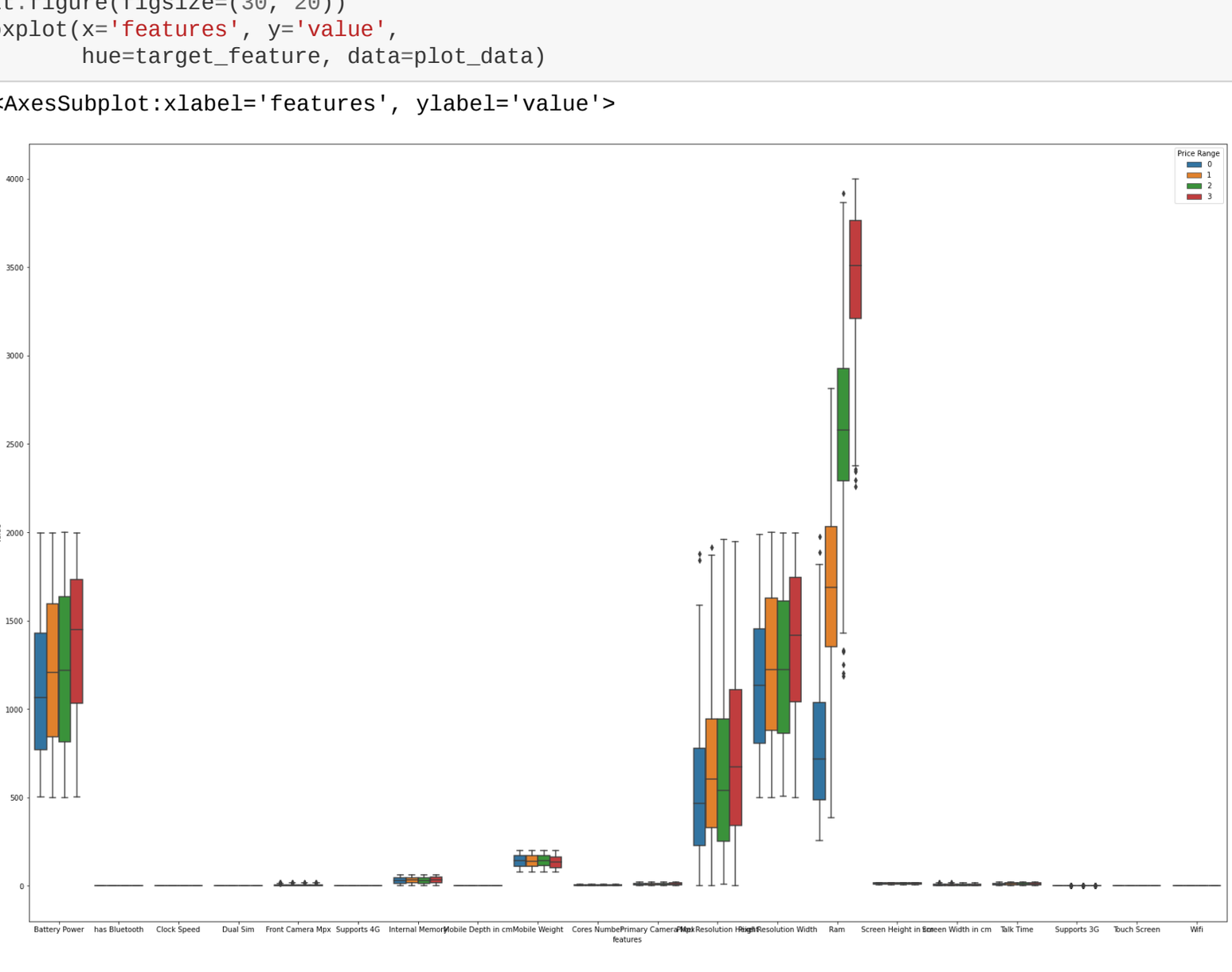
So, now let's narrow our study focusing on features have more impact on selling price: **Battery Power**, **Pixel Resolution Height**, **Pixel Resolution Width**, **Ram** and our target feature **Price Range**. We start with some visual representation of each feature to see how it related to **Price Range**.

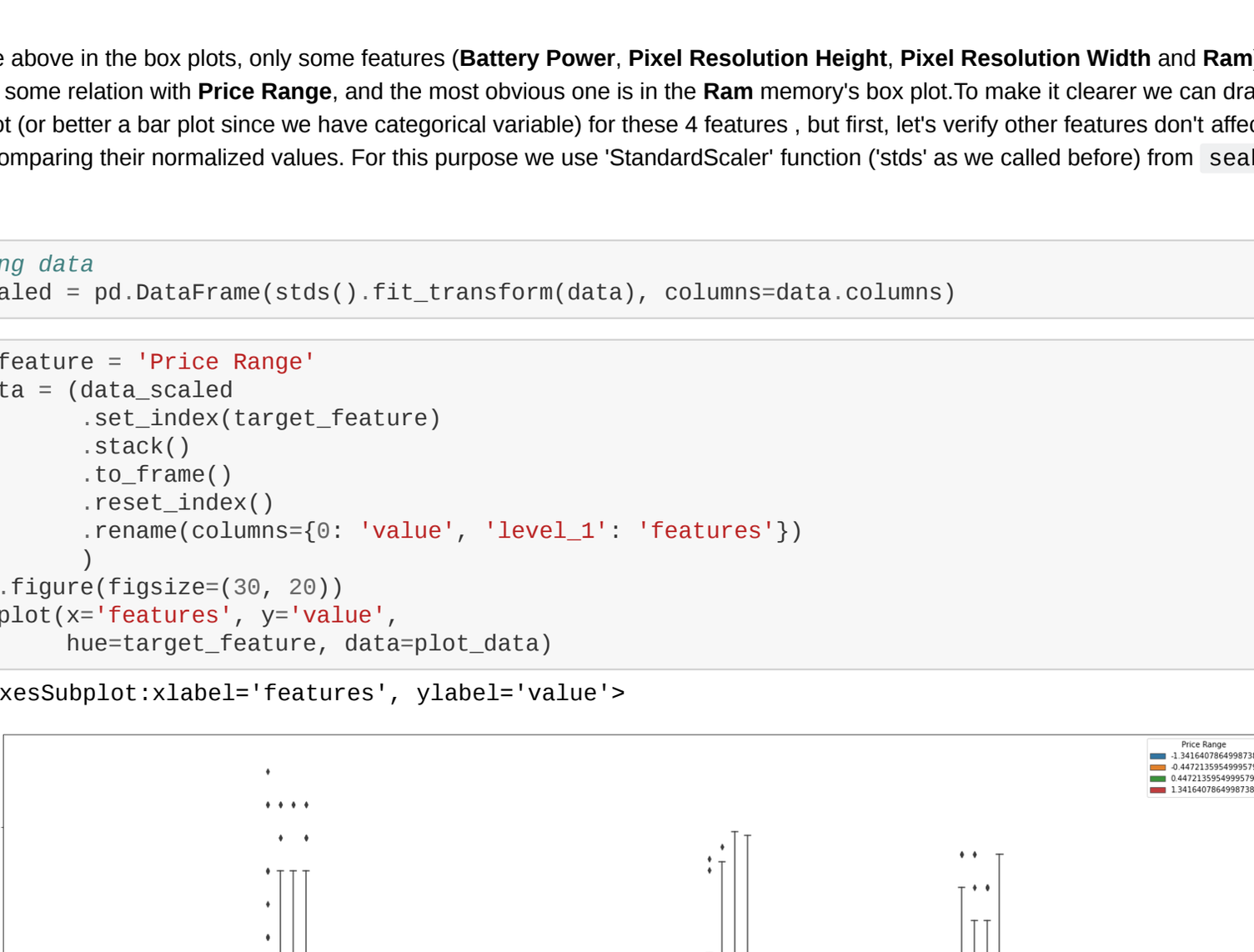
```
selected_features = ['Battery Power', 'Pixel Resolution Height',
                     'Pixel Resolution Width', 'Ram', 'Price Range']
df_1 = data[selected_features]
df_1.head()
```

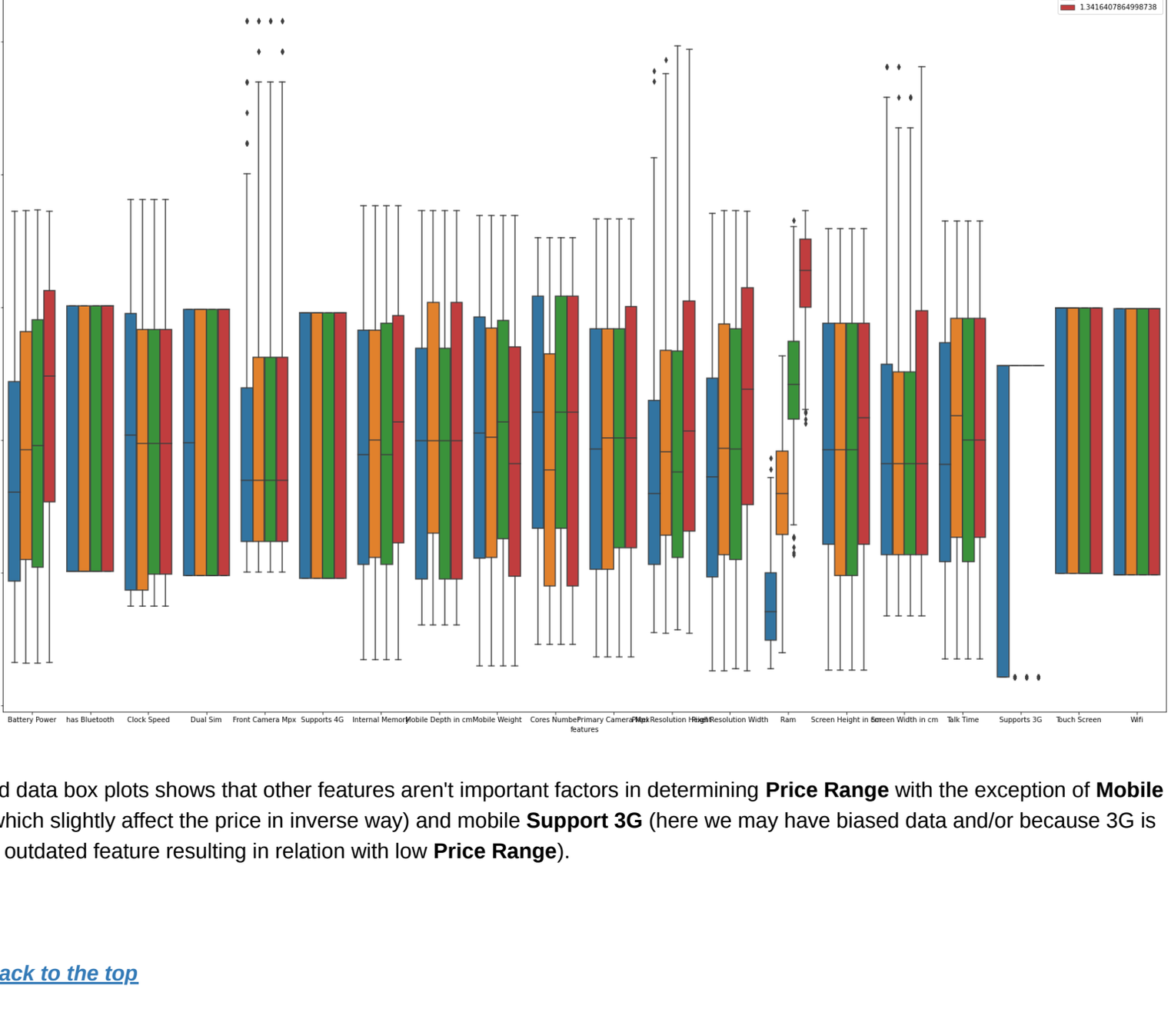
	Battery Power	Pixel Resolution Height	Pixel Resolution Width	Ram	Price Range
0	842	20	756	2549	1
1	1021	905	1988	2631	2
2	563	1263	1716	2603	2
3	615	1216	1786	2769	2
4	1821	1208	1212	1411	1

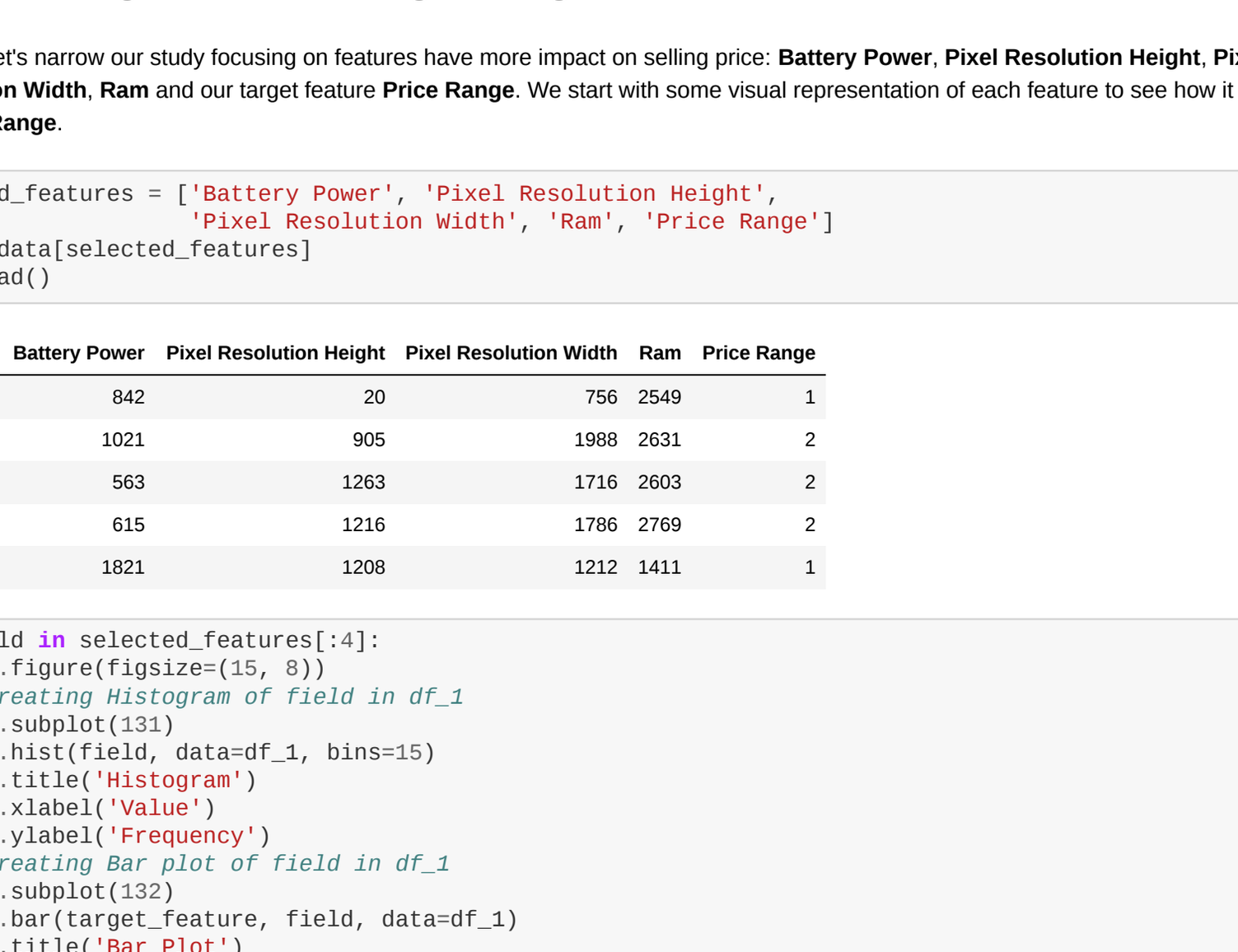
```
for field in selected_features[:4]:
    plt.figure(figsize=(15, 8))
    # Creating Histogram of field in df_1
    plt.subplot(131)
    plt.hist(field, data=df_1, bins=15)
    plt.title('Histogram')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    # Creating Bar plot of field in df_1
    plt.subplot(132)
    plt.bar(target_feature, field, data=df_1)
    plt.title('Bar Plot')
    plt.xlabel('Price Range')
    plt.ylabel('Frequency')
    # Creating scatter plot of field in df_1
    plt.subplot(133)
    plt.scatter(target_feature, field, data=df_1)
    plt.title('Scatter Plot')
    plt.xlabel('Price Range')
    plt.ylabel(field)

    plt.suptitle(field)
    plt.show()
```









The figure above shows that only **Ram** has a clear linear relation with price. But, Since we have found before form Box Plots that **Pixel Resolution Height** and **Pixel Resolution Width** are both possible factors, Let's add their product as a new feature and see what we might find.

```
new_col = pd.DataFrame(df_1['Pixel Resolution Height']*df_1['Pixel Resolution Width'],
                       columns=["Pixel Resolution Product"], index=df_1.index)
new_col.head()
```

	Pixel Resolution Product
0	15120
1	1799140
2	2167308
3	2171776
4	1464096

The new DataFrame:

```
df_1_new = df_1.merge(new_col, right_index=True, left_index=True)
df_1_new.head()
```

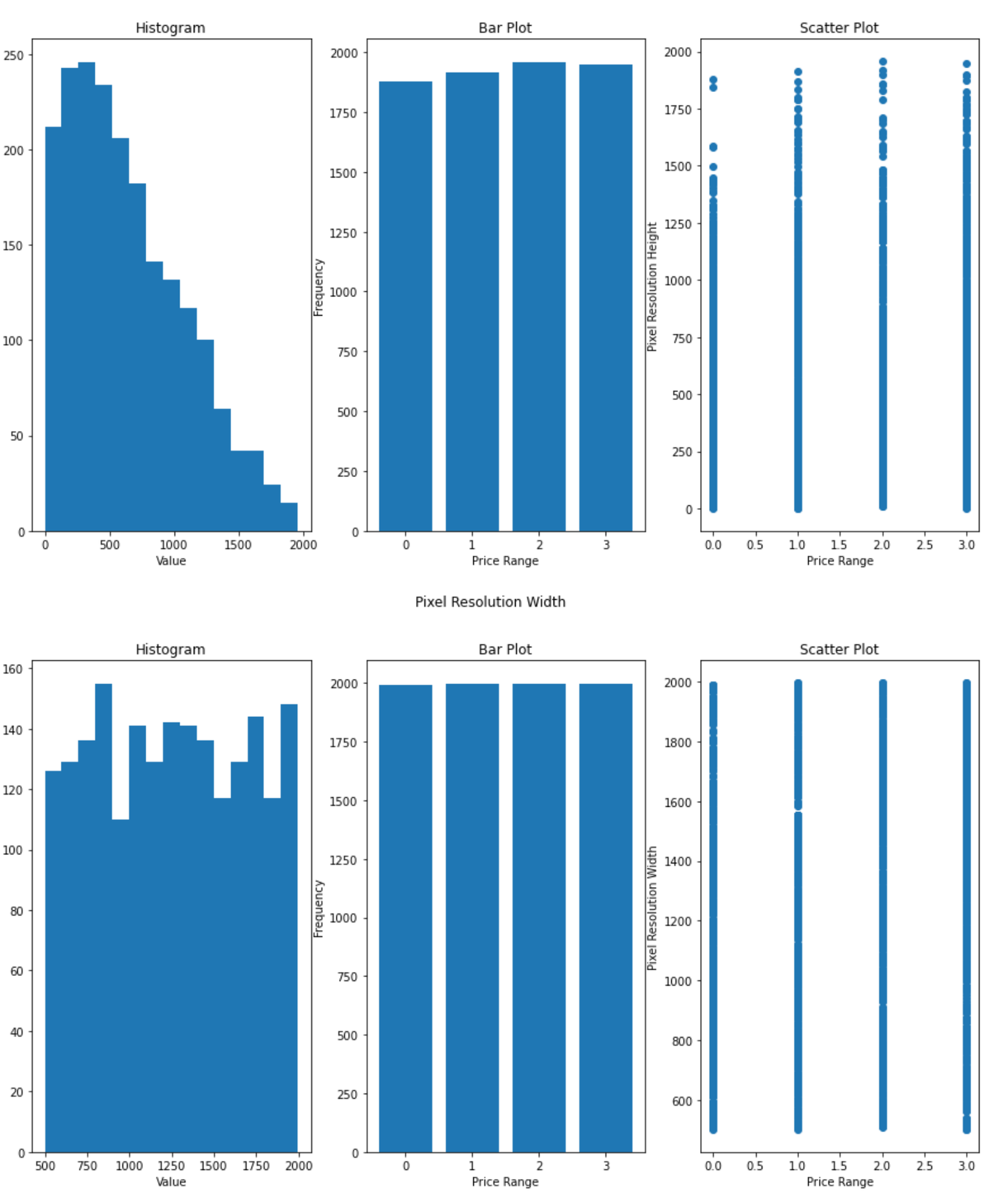
	Battery Power	Pixel Resolution Height	Pixel Resolution Width	Ram	Price Range	Pixel Resolution Product
0	842	20	756	2549	1	15120
1	1021	905	1988	2631	2	1799140
2	563	1263	1716	2603	2	2167308
3	615	1216	1786	2769	2	2171776
4	1821	1208	1212	1411	1	1464096

Pair plot for normalized data:

After we added the new column **Pixel Resolution Product**, let's draw a pair plot of all features of the new subset.

```
selected_features.append("Pixel Resolution Product")
# to avoid having a duplicate value of added column when running cell again
df1_new_t = pd.DataFrame(stds().fit_transform(
    df_1_new), columns=selected_features)
print("Fit transform done")
sns.pairplot(df1_new_t, plot_kws=dict(alpha=.3, edgecolor='none'))
```

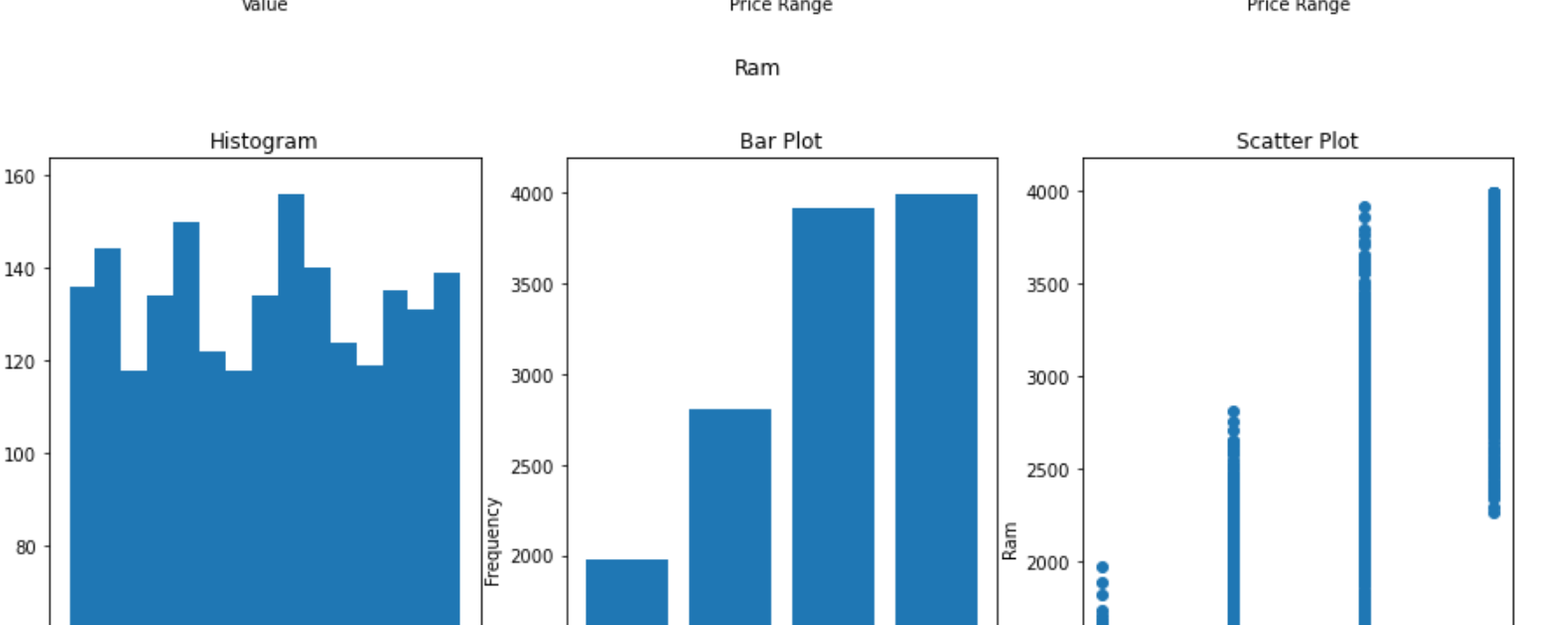
```
Fit transform done
<seaborn.axisgrid.PairGrid at 0x7fc6336870d0>
```



The next figure shows that the added new column hasn't much effect on **Price Range**:

```
fig, (ram_ax, res_ax) = plt.subplots(1, 2, figsize=(15, 8))
ram_ax.scatter(y='Price Range', x='Ram', data=df_1_new, marker='+')
ram_ax.set(xlabel='Ram', ylabel='Price Range')
res_ax.scatter(y='Price Range', x='Pixel Resolution Product',
               data=df_1_new, marker='o')
res_ax.set(xlabel='Pixel Resolution Product', ylabel='Price Range')
```

```
[Text(0.5, 0, 'Pixel Resolution Product'), Text(0, 0.5, 'Price Range')]
```



> ***Back to the top***

Key Finding and Insights:

We find that ram memory size is the most effective feature that could be used to predict the price of a mobile phone.

> ***Back to the top***

Hypothesis About the Data:

1. Ram size can predict the range of mobile.
2. Mobile ram memory size can always predict the right price range.
3. Battery hasn't mush effect on mobile price.

> ***Back to the top***

Significance Test for 1st Hypothesis:

Null Hypothesis:

Ram true guess rate is 0.25.

Alternative Hypothesis:

Ram true guess rate doesn't equal 0.25.

If the null is correct, the test statistic is uniformly distributed. So, The first 25% of ram size values should be in range 0, The second 25% in range 1 and so on...

Let's calculate the probability.

```
from scipy import stats
ram_df = data[["Ram", "Price Range"]]
ram_df_1=ram_df.sort_values(by=['Ram'],ascending=True).set_index('Ram').reset_index()
```

```
# Dividing the sorted dataset into 4 subsets with each contains 500 row
ram_groups = [ram_df_1.iloc[i:i+500] for i in range(0, 2000, 250)]
ram_group_p = []
# calculating the probability of each range in each subset
for group in ram_groups:
    ram_group_p.append(group.groupby('Price Range').count()/500)
```

```
p = {0: 0, 1: 0, 2: 0, 3: 0}
for i in range(4):
    g_temp = pd.DataFrame(ram_group_p[i])
    p[i] = g_temp.iloc[i]['Ram']

p = pd.DataFrame(p, index=['P'])
p
```

	0	1	2	3
P	0.846	0.456	0.172	0.036

So, the probability to get mobile of price range 0 in the first 25% from dataset is 84.6%, and to get range 1 mobile in second 25% is 45.6%, 17.2% for the range 2 in third quarter and 3.6% range 3 in the last 500 sample. So, the average guess rate is: p = 0.38

```
p.mean(axis=1)
```

```
P    0.3775
dtype: float64
```

As we see the probability of getting a correct prediction is 38%, Which is not equal to 25% as the `null` hypothesis suggests. So, we can reject the `null` hypothesis.

> ***Back to the top***

Next Steps

Finding a linear model that can predict Price Range from Ram size or multilinear (or non-linear) model that can predict range using several features in addition to Ram like Battery Power, Pixel Resolution...

> ***Back to the top***

Summery of the Quality of the Dataset:

Since we couldn't find much relationship between our target variable and other variables unlike what expected, It's fair to say that our dataset isn't sufficient for its purpose, and it needs to be supported (or replaced) with more reliable data.

> ***Back to the top***