# Build Linear Regression Model in Python

In this Jupyter notebook, I will be showing you how to build a linear regression model in Python using the scikit-learn package on a Mobile Price Prediction Dataset from Kaggle

In [1]:
```
#Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
```

In [2]:
```
#Load Dataset
data=pd.read_csv("C:/Users/vadla/Downloads/Cellphone.csv")
```

In [3]:
```
#Quick View about Dataset
data
```

Out[3]:

| | 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 | 5.20 | 424 | 8 | 1.350 | 16.0 | 3.000 | 13.00 | 8.0 | 2610 | 7.4 |
| **1** | 880 | 1749 | 10 | 125.0 | 4.00 | 233 | 2 | 1.300 | 4.0 | 1.000 | 3.15 | 0.0 | 1700 | 9.9 |
| **2** | 40 | 1916 | 10 | 110.0 | 4.70 | 312 | 4 | 1.200 | 8.0 | 1.500 | 13.00 | 5.0 | 2000 | 7.6 |
| **3** | 99 | 1315 | 11 | 118.5 | 4.00 | 233 | 2 | 1.300 | 4.0 | 0.512 | 3.15 | 0.0 | 1400 | 11.0 |
| **4** | 880 | 1749 | 11 | 125.0 | 4.00 | 233 | 2 | 1.300 | 4.0 | 1.000 | 3.15 | 0.0 | 1700 | 9.9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **156** | 1206 | 3551 | 4638 | 178.0 | 5.46 | 538 | 4 | 1.875 | 128.0 | 6.000 | 12.00 | 16.0 | 4080 | 8.4 |
| **157** | 1296 | 3211 | 8016 | 170.0 | 5.50 | 534 | 4 | 1.975 | 128.0 | 6.000 | 20.00 | 8.0 | 3400 | 7.9 |
| **158** | 856 | 3260 | 8809 | 150.0 | 5.50 | 401 | 8 | 2.200 | 64.0 | 4.000 | 20.00 | 20.0 | 3000 | 6.8 |
| **159** | 1296 | 3211 | 8946 | 170.0 | 5.50 | 534 | 4 | 1.975 | 128.0 | 6.000 | 20.00 | 8.0 | 3400 | 7.9 |
| **160** | 1131 | 2536 | 9807 | 202.0 | 6.00 | 367 | 8 | 1.500 | 16.0 | 3.000 | 21.50 | 16.0 | 2700 | 8.4 |

161 rows × 14 columns

In [4]:
```
data.shape # Number of rows and columns
```

Out[4]:
```
(161, 14)
```

In [5]:
```
data.describe() #statistical Analysis
```

Out[5]:

| | Product_id | Price | Sale | weight | resoloution | ppi | cpu core | cpu freq | internal mem | ram | RearCam | Front_C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000000 | 161.000( |
| **mean** | 675.559006 | 2215.596273 | 621.465839 | 170.426087 | 5.209938 | 335.055901 | 4.857143 | 1.502832 | 24.501714 | 2.204994 | 10.378261 | 4.5031 |
| **std** | 410.851583 | 768.187171 | 1546.618517 | 92.888612 | 1.509953 | 134.826659 | 2.444016 | 0.599783 | 28.804773 | 1.609831 | 6.181585 | 4.342( |
| **min** | 10.000000 | 614.000000 | 10.000000 | 66.000000 | 1.400000 | 121.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000( |
| **25%** | 237.000000 | 1734.000000 | 37.000000 | 134.100000 | 4.800000 | 233.000000 | 4.000000 | 1.200000 | 8.000000 | 1.000000 | 5.000000 | 0.000( |
| **50%** | 774.000000 | 2258.000000 | 106.000000 | 153.000000 | 5.150000 | 294.000000 | 4.000000 | 1.400000 | 16.000000 | 2.000000 | 12.000000 | 5.000( |
| **75%** | 1026.000000 | 2744.000000 | 382.000000 | 170.000000 | 5.500000 | 428.000000 | 8.000000 | 1.875000 | 32.000000 | 3.000000 | 16.000000 | 8.000( |
| **max** | 1339.000000 | 4361.000000 | 9807.000000 | 753.000000 | 12.200000 | 806.000000 | 8.000000 | 2.700000 | 128.000000 | 6.000000 | 23.000000 | 20.000( |

In [6]:
```
#Data Cleansing:
data.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
```

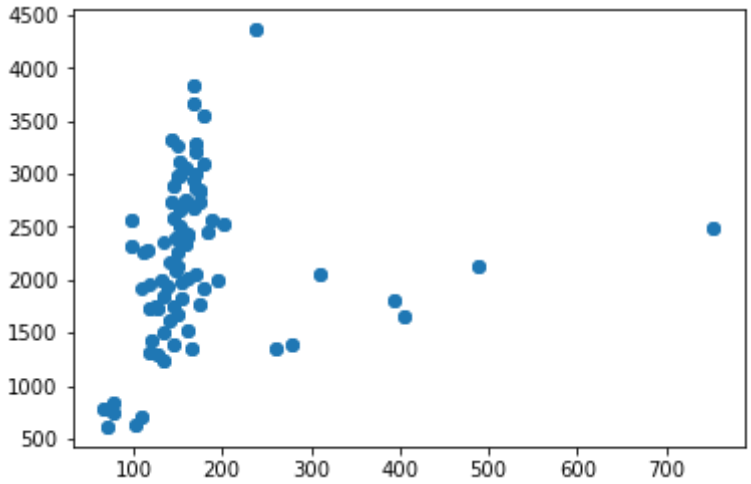
All of the features of this dataset belongs to either "Int" or "Float" Type. There are No Null values in this dataset.

## Data Visualization

We will be using Scatter plots. They will observe the relationship between variables and uses dots to represent the connection between them.

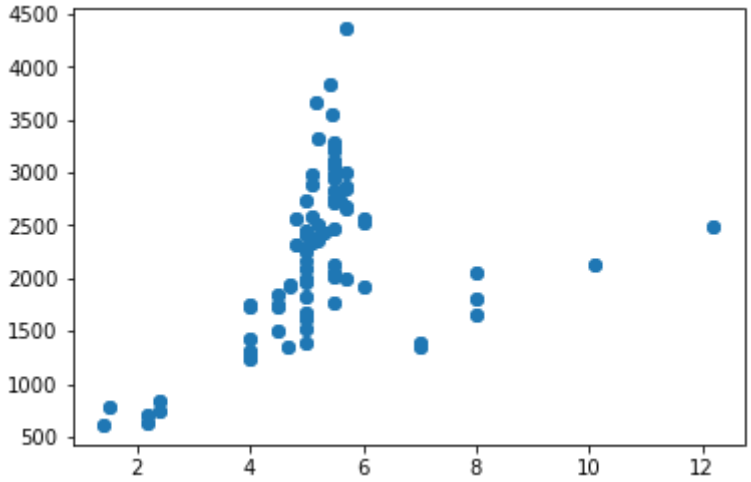
In [7]:
```
plt.scatter(data["weight"], data["Price"])
```

Out[7]:
```
<matplotlib.collections.PathCollection at 0x1ebe885f370>
```



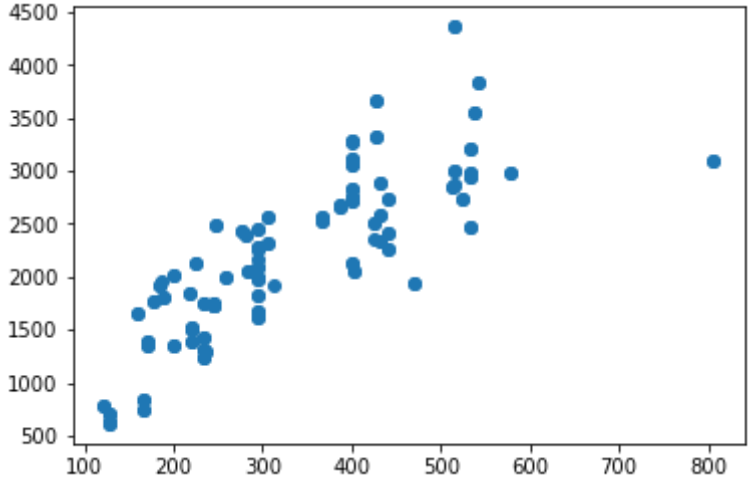

In [8]:
```
plt.scatter(data["resoloution"], data["Price"])
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x1ebe8963be0>
```



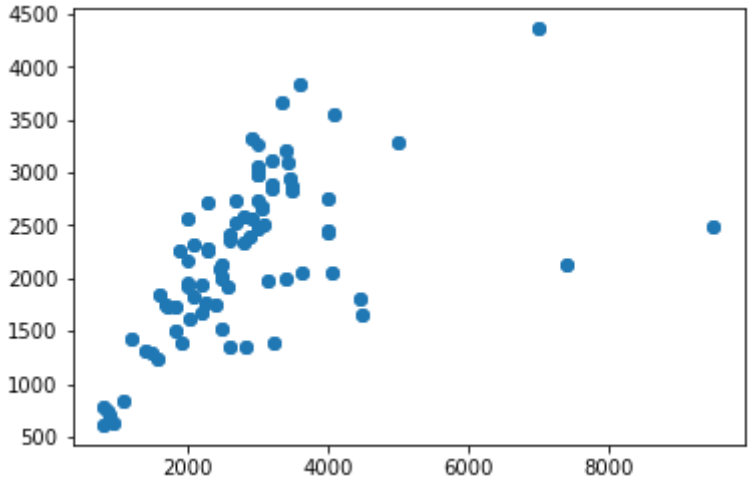

In [9]:
```
plt.scatter(data["ppi"], data["Price"])
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x1ebe89de400>
```



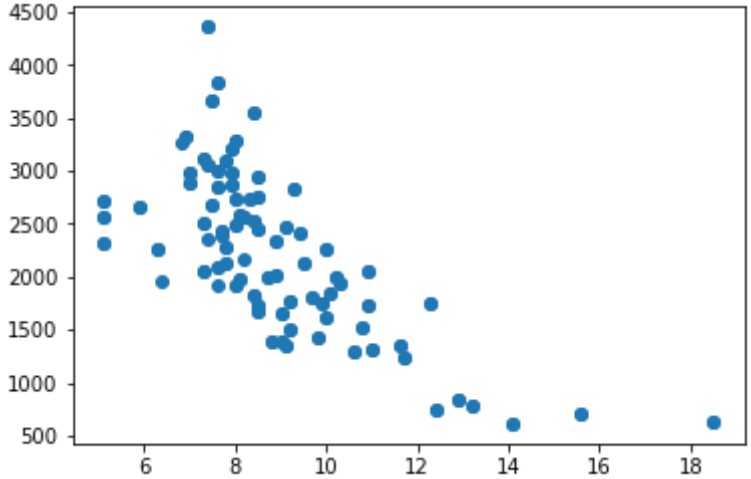

In [10]:
```
plt.scatter(data["battery"], data["Price"])
```

Out[10]:
```
<matplotlib.collections.PathCollection at 0x1ebe8a553d0>
```



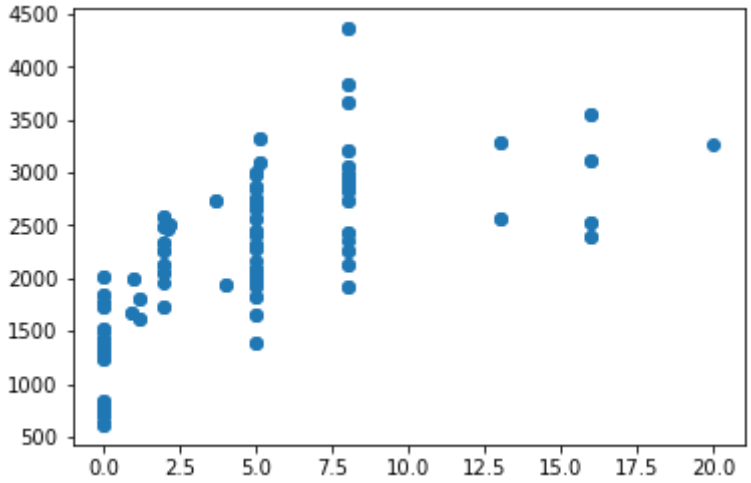

In [11]:
```
plt.scatter(data["thickness"], data["Price"])
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x1ebe8ab2d60>
```



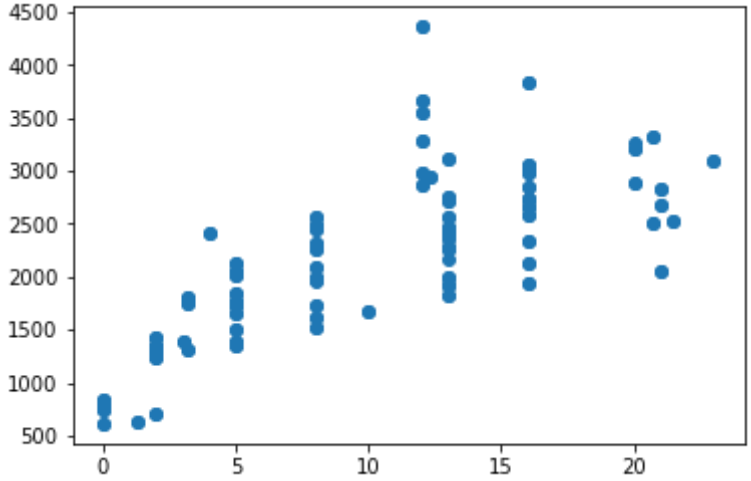

In [12]:
```
plt.scatter(data["Front_Cam"], data["Price"])
```

Out[12]:
```
<matplotlib.collections.PathCollection at 0x1ebe8b24a60>
```



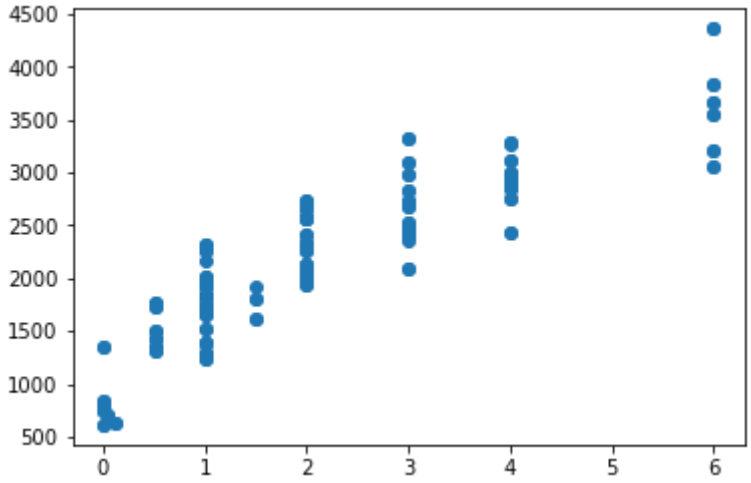

In [13]:
```
plt.scatter(data["RearCam"], data["Price"])
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x1ebe8b90f40>
```



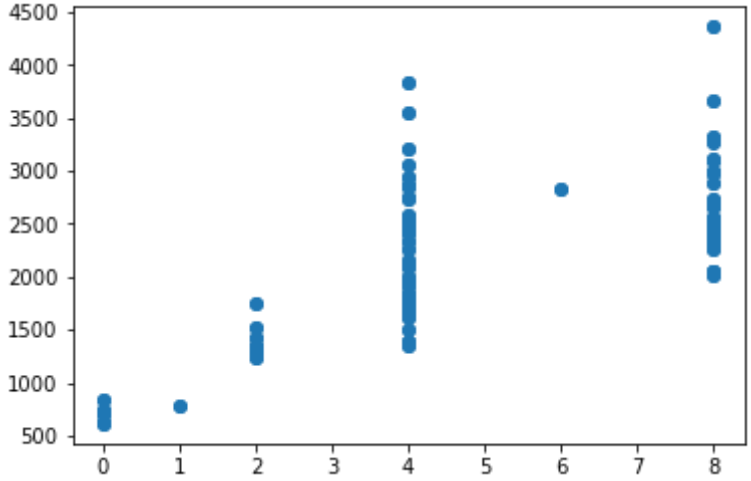

In [14]:
```
plt.scatter(data["ram"], data["Price"])
```

Out[14]:
```
<matplotlib.collections.PathCollection at 0x1ebe8bf6f10>
```



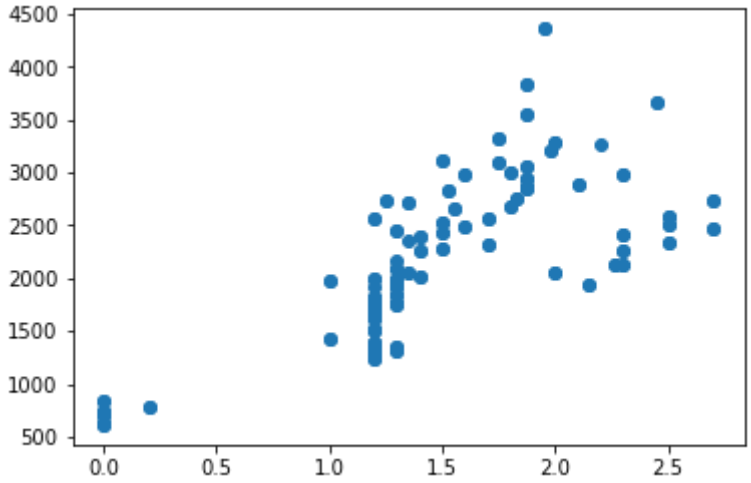

In [15]:
```
plt.scatter(data["cpu core"], data["Price"])
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x1ebe9c3a7c0>
```



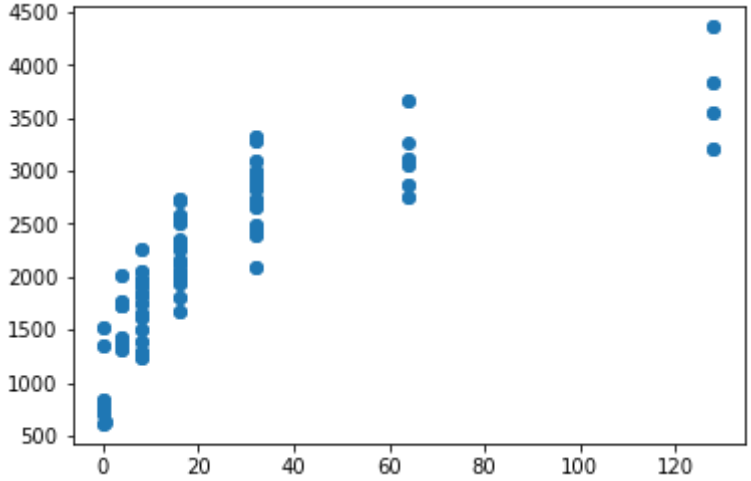

In [16]:
```
plt.scatter(data["cpu freq"], data["Price"])
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x1ebe9cb7160>
```




In [17]:
```
plt.scatter(data["internal mem"], data["Price"])
```

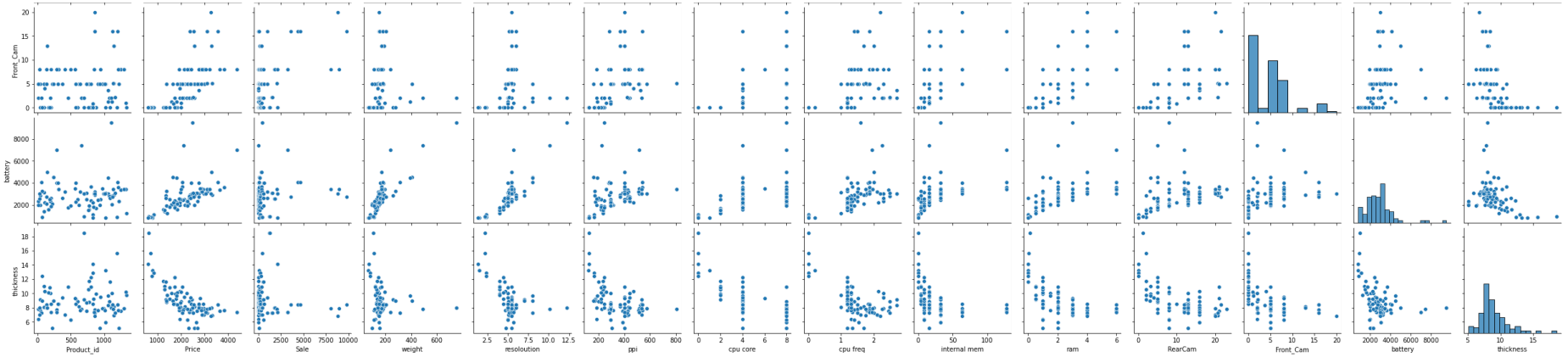
Out[17]:
```
<matplotlib.collections.PathCollection at 0x1ebe9d1f790>
```




In [18]:
```
sns.pairplot(data)
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x1ebe9c68c40>
```


Product_id
Price
Sale
weight
resolution
ppi
cpu core
cpu freq
internal mem
ram
Rear Cam

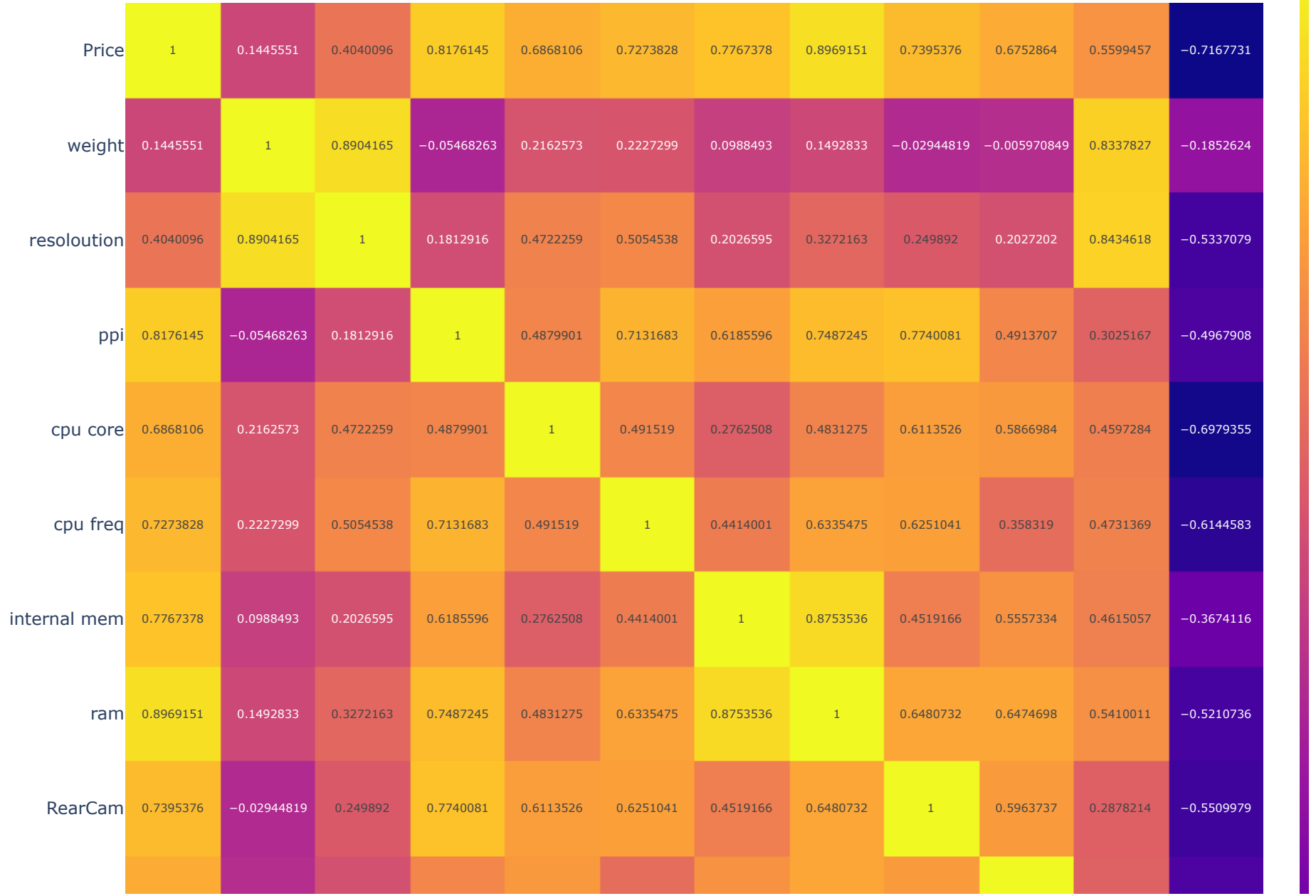


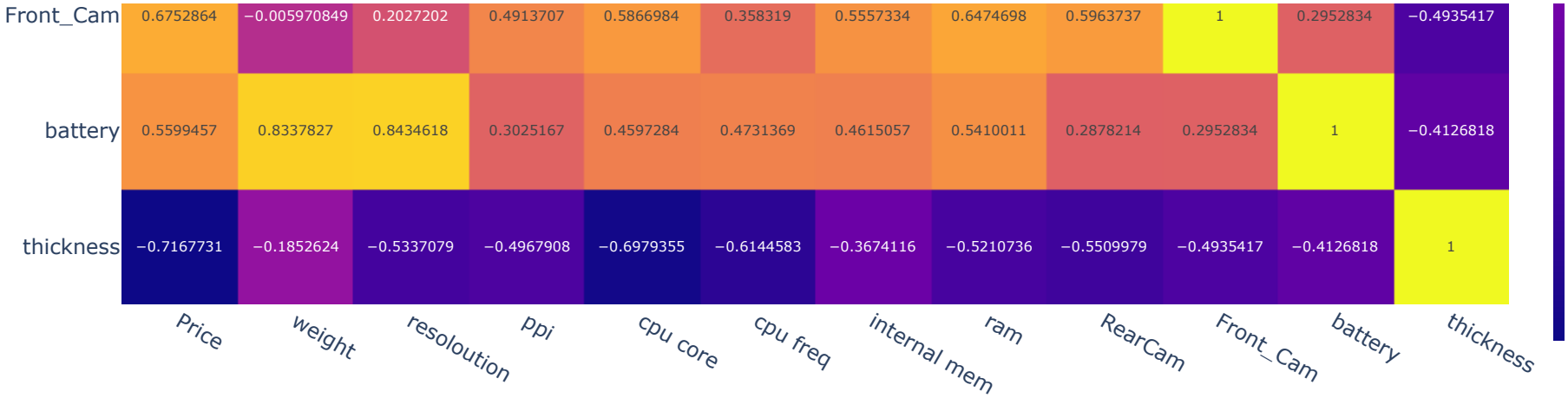

The pairplot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column. That creates plots as shown above

From the visulizations,we can deduce that almost all the features have a linear relationship

# Checking correlation between features

In [19]:
```
data=data.drop(columns=["Product_id","Sale"]) #Dropping unwanted columns as they don't add any value to our analysis
corr=data.corr()
import plotly.express as px
fig = px.imshow(corr, text_auto=True,width=1000, height=1000)
fig.show()
```


Price 1 0.1445551 0.4040096 0.8176145 0.6868106 0.7273828 0.7767378 0.8969151 0.7395376 0.6752864 0.5599457 −0.7167731
weight 0.1445551 1 0.8904165 −0.05468263 0.2162573 0.2227299 0.0988493 0.1492833 −0.02944819 −0.005970849 0.8337827 −0.1852624
resoloution 0.4040096 0.8904165 1 0.1812916 0.4722259 0.5054538 0.2026595 0.3272163 0.249892 0.2027202 0.8434618 −0.5337079
ppi 0.8176145 −0.05468263 0.1812916 1 0.4879901 0.7131683 0.6185596 0.7487245 0.7740081 0.4913707 0.3025167 −0.4967908
cpu core 0.6868106 0.2162573 0.4722259 0.4879901 1 0.491519 0.2762508 0.4831275 0.6113526 0.5866984 0.4597284 −0.6979355
cpu freq 0.7273828 0.2227299 0.5054538 0.7131683 0.491519 1 0.4414001 0.6335475 0.6251041 0.358319 0.4731369 −0.6144583
internal mem 0.7767378 0.0988493 0.2026595 0.6185596 0.2762508 0.4414001 1 0.8753536 0.4519166 0.5557334 0.4615057 −0.3674116
ram 0.8969151 0.1492833 0.3272163 0.7487245 0.4831275 0.6335475 0.8753536 1 0.6480732 0.6474698 0.5410011 −0.5210736
RearCam 0.7395376 −0.02944819 0.249892 0.7740081 0.6113526 0.6251041 0.4519166 0.6480732 1 0.5963737 0.2878214 −0.5509979




# Split datset into X and Y variables

In [20]:
```
x=data.drop('Price',axis=1)
y=data['Price']
```

In [21]:
```
x.shape
```

Out[21]:
```
(161, 11)
```

In [22]:
```
y.shape
```

Out[22]:
```
(161,)
```

# Perform 80/20 Data split

Since I have already imported train test split, we can split the data into 80:20. Most commonly the ratio used to split the data is 80:20. This is done so that we or our model don't see a particular set of data and is kept aside for testing our trained model. And the larger set is always used for

training and the latter for testing.

In [23]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

# Checking data dimensions

In [24]:
```
x_train.shape, y_train.shape
```

Out[24]:
```
((128, 11), (128,))
```

In [25]:
```
x_test.shape, y_test.shape
```

Out[25]:
```
((33, 11), (33,))
```

# Linear Regression Model

In [26]:
```
#Importing necessarylibraries
#from sklearn import linear_model-already imported
from sklearn.metrics import mean_squared_error, r2_score
```

# Build linear regression

### Defines the regression model

In [27]:
```
model = linear_model.LinearRegression()
```

### Build training model

In [28]:
```
model.fit(x_train, y_train)
```

Out[28]:
```
▾ LinearRegression
LinearRegression()
```

### Apply trained model to make prediction (on test set)

In [29]:
```
y_pred = model.predict(x_test)
```

# Prediction results

### Print model performance

In [30]:
```
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
print('Mean squared error (MSE): %.2f'
      % mean_squared_error(y_test, y_pred))
print('Coefficient of determination (R^2): %.2f'
      % r2_score(y_test, y_pred))
```

```
Coefficients: [-9.74842513e-02 -1.01941355e+02  1.11214223e+00  5.78396597e+01
  1.66947111e+02  4.61508113e+00  1.06313954e+02 -2.13485550e+00
  1.06044698e+01  1.26580867e-01 -7.70679310e+01]
Intercept: 1807.770632165054
Mean squared error (MSE): 36015.46
Coefficient of determination (R^2): 0.96
```

# String Formatting

By default r2_score returns a floating number

In [31]:
```
r2_score(y_test, y_pred).dtype
```

Out[31]:
```
dtype('float64')
```

In [32]:
```
r2_score(y_test, y_pred)
```

Out[32]:
```
0.9609474449922408
```

We will be using the modulo operator to format the numbers by rounding it off.

In [33]:
```
'%.2f' %0.9609474449922408
```

Out[33]:
```
'0.96'
```

the Result for R^2 is 0.96 that is very good and showing that Linear regression model is Right Choice. It means that independent variables describe 96% of dependent variable!

In [ ]: