Prediction of Mobile Model Price using Machine learining techniques

AIM:

In this Project,On the basis of the mobile Specification like Battery power, 3G enabled , wifi ,Bluetooth, Ram etc we are predicting Price range of the mobile

In this data:

* id:ID
* 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 you are
* three_g:Has 3G or not
* touch_screen:Has touch screen or not
* wifi:Has wifi or not

USE:

* This kind of prediction will help companies estimate price of mobiles to give tough competion to other mobile manufacturer
* Also it will be usefull for Consumers to verify that they are paying best price for a mobile.

Applied Models:

* Linear Regression
* KNN
* Logistic Regression
* Decision tree
* Random forest

In [10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
from scipy.stats import probplot
```

In [11]:
```
df=pd.read_csv("C:\\Users\\user\\Downloads\\train.csv")
df
```

Out[11]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2

2000 rows × 21 columns

Data Analysis

In [4]:
```
df.shape
```

Out[4]:
```
(2000, 21)
```

In [5]:
```
df.describe()
```

Out[5]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000	20
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	12
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	4
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	5
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	8
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	12
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	16
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	19

8 rows × 21 columns

In [6]:
```
df.dtypes
```

Out[6]:
```
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object
```

In [7]:
```
df.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
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
df.count()
```

Out[10]:
```
battery_power    2000
blue             2000
clock_speed      2000
dual_sim         2000
fc               2000
four_g           2000
int_memory       2000
m_dep            2000
mobile_wt        2000
n_cores          2000
pc               2000
px_height        2000
px_width         2000
ram              2000
sc_h             2000
sc_w             2000
talk_time        2000
three_g          2000
touch_screen     2000
wifi             2000
price_range      2000
dtype: int64
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

In [12]:
```
df.corr()
```

Out[12]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	
battery_power	1.000000	0.011252	0.011482	-0.041847	0.033334	0.015665	-0.004004	0.034085	0.001844	-0.029727	...	0.014901	-0.008402	-0
blue	0.011252	1.000000	0.021419	0.035198	0.003593	0.013443	0.041177	0.004049	-0.008605	0.036161	...	-0.006872	-0.041533	0
clock_speed	0.011482	0.021419	1.000000	-0.001315	-0.000434	-0.043073	0.006545	-0.014364	0.012350	-0.005724	...	-0.014523	-0.009476	0
dual_sim	-0.041847	0.035198	-0.001315	1.000000	-0.029123	0.003187	-0.015679	-0.022142	-0.008979	-0.024658	...	-0.020875	0.014291	0
fc	0.033334	0.003593	-0.000434	-0.029123	1.000000	-0.016560	-0.029133	-0.001791	0.023618	-0.013356	...	-0.009990	-0.005176	0
four_g	0.015665	0.013443	-0.043073	0.003187	-0.016560	1.000000	0.008690	-0.001823	-0.016537	-0.029706	...	-0.019236	0.007448	0
int_memory	-0.004004	0.041177	0.006545	-0.015679	-0.029133	0.008690	1.000000	0.006886	-0.034214	-0.028310	...	0.010441	-0.008335	0
m_dep	0.034085	0.004049	-0.014364	-0.022142	-0.001791	-0.001823	0.006886	1.000000	0.021756	-0.003504	...	0.025263	0.023566	-0
mobile_wt	0.001844	-0.008605	0.012350	-0.008979	0.023618	-0.016537	-0.034214	0.021756	1.000000	-0.018989	...	0.000939	0.000090	-0
n_cores	-0.029727	0.036161	-0.005724	-0.024658	-0.013356	-0.029706	-0.028310	-0.003504	-0.018989	1.000000	...	-0.006872	0.024480	0
pc	0.031441	-0.009952	-0.005245	-0.017143	0.644595	-0.005598	-0.033273	0.026282	0.018844	-0.001193	...	-0.018465	0.004196	0
px_height	0.014901	-0.006872	-0.014523	-0.020875	-0.009990	-0.019236	0.010441	0.025263	0.000939	-0.006872	...	1.000000	0.510664	-0
px_width	-0.008402	-0.041533	-0.009476	0.014291	-0.005176	0.007448	-0.008335	0.023566	0.000090	0.024480	...	0.510664	1.000000	0
ram	-0.000653	0.026351	0.003443	0.041072	0.015099	0.007313	0.032813	-0.009434	-0.002581	0.004868	...	-0.020352	0.004105	1
sc_h	-0.029959	-0.002952	-0.029078	-0.011949	-0.011014	0.027166	0.037771	-0.025348	-0.033855	-0.000315	...	0.059615	0.021599	0
sc_w	-0.021421	0.000613	-0.007378	-0.016666	-0.012373	0.037005	0.011731	-0.018388	-0.020761	0.025826	...	0.043038	0.034699	0
talk_time	0.052510	0.013934	-0.011432	-0.039404	-0.006829	-0.046628	-0.002790	0.017003	0.006209	0.013148	...	-0.010645	0.006720	0
three_g	0.011522	-0.030236	-0.046433	-0.014008	0.001793	0.584246	-0.009366	-0.012065	0.001551	-0.014733	...	-0.031174	0.000350	0
touch_screen	-0.010516	0.010061	0.019756	-0.017117	-0.014828	0.016758	-0.026999	-0.002638	-0.014368	0.023774	...	0.021891	-0.001628	-0
wifi	-0.008343	-0.021863	-0.024471	0.022740	0.020085	-0.017620	0.006993	-0.028353	-0.000409	-0.009964	...	0.051824	0.030319	0
price_range	0.200723	0.020573	-0.006606	0.017444	0.021998	0.014772	0.044435	0.000853	-0.030302	0.004399	...	0.148858	0.165818	0

21 rows × 21 columns

In [14]:
```
numerical_feature= [feature for feature in df.columns if df[feature].dtypes != 'O']
discrete_feature =[feature for feature in numerical_feature if len(df[feature].unique())<25]
continuous_feature = [feature for feature in numerical_feature if feature not in discrete_feature]
categorical_feature = [feature for feature in df.columns if feature not in numerical_feature]
print("Numerical Features Count {}".format(len(numerical_feature)))
print("Discrete feature Count {}".format(len(discrete_feature)))
print("Continuous feature Count {}".format(len(continuous_feature)))
print("Categorical feature Count {}".format(len(categorical_feature)))
```

```
Numerical Features Count 21
Discrete feature Count 14
Continuous feature Count 7
Categorical feature Count 0
```

In [15]:
```
print(numerical_feature)
```

```
['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_h
eight', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'price_range']
```

In [16]:
```
print(discrete_feature)
```

```
['blue', 'dual_sim', 'fc', 'four_g', 'm_dep', 'n_cores', 'pc', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi',
'price_range']
```

In [17]:
```
print(continuous_feature)
```

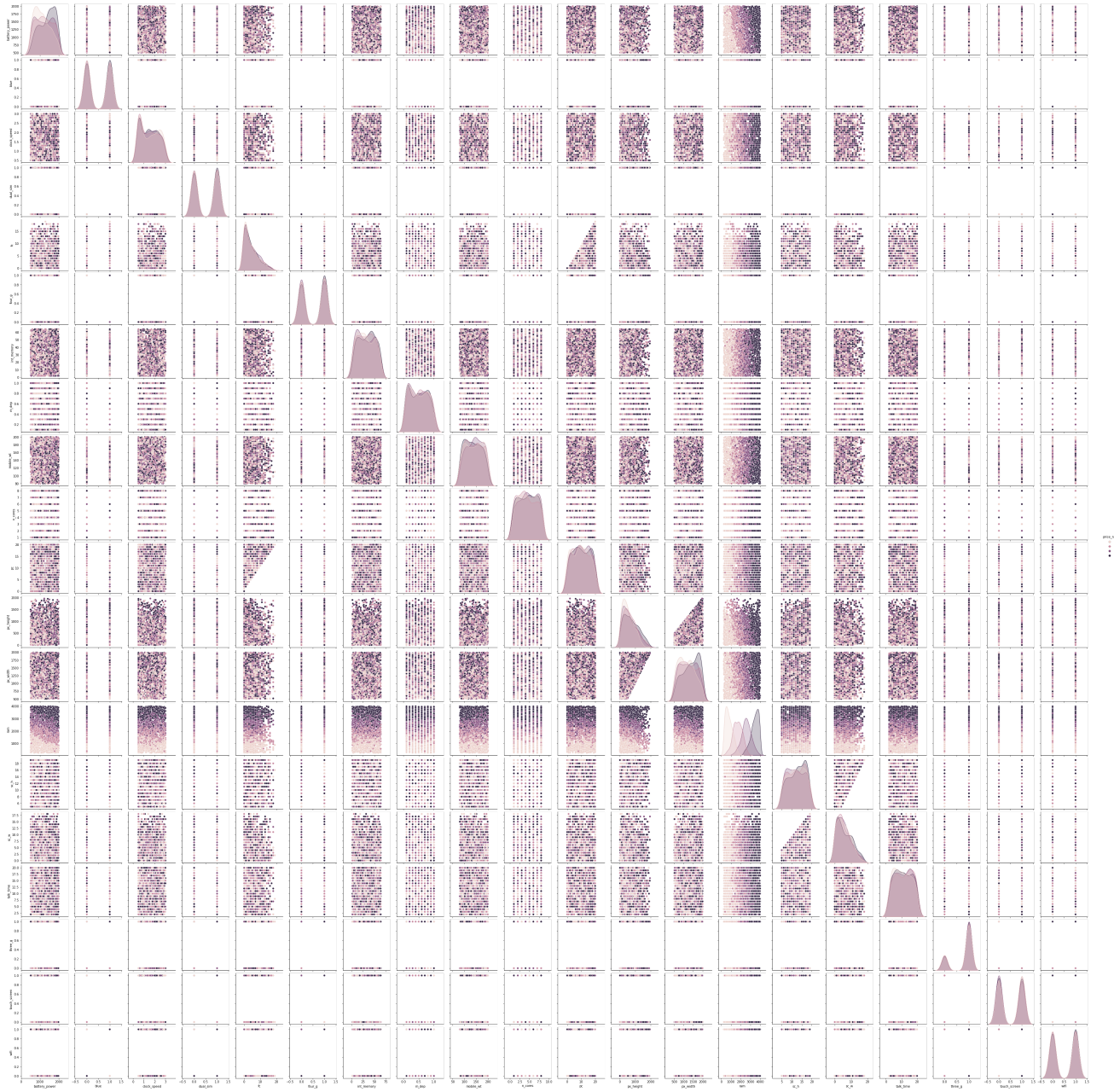
```
['battery_power', 'clock_speed', 'int_memory', 'mobile_wt', 'px_height', 'px_width', 'ram']
```

Data Visualization & Analysis

In [18]:
```
sns.clustermap(df.corr(), cmap = "vlag", dendrogram_ratio = (0.1, 0.2), annot = True, linewidths = .10, figsize = (14,16))
plt.show()
```

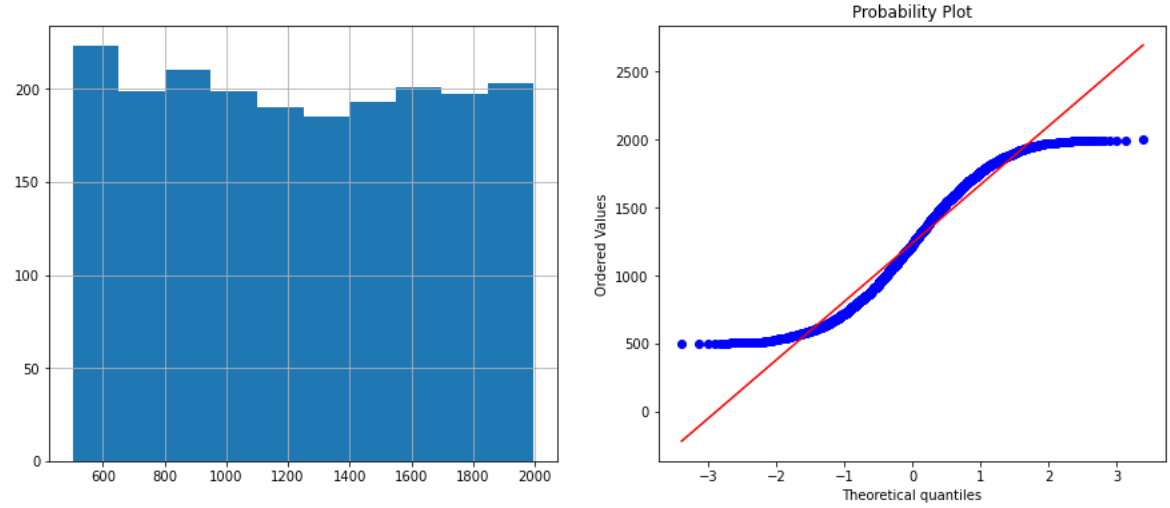
In [19]:
```
sns.pairplot(df,hue='price_range')
```

Out[19]: <seaborn.axisgrid.PairGrid at 0xd31301ea30>

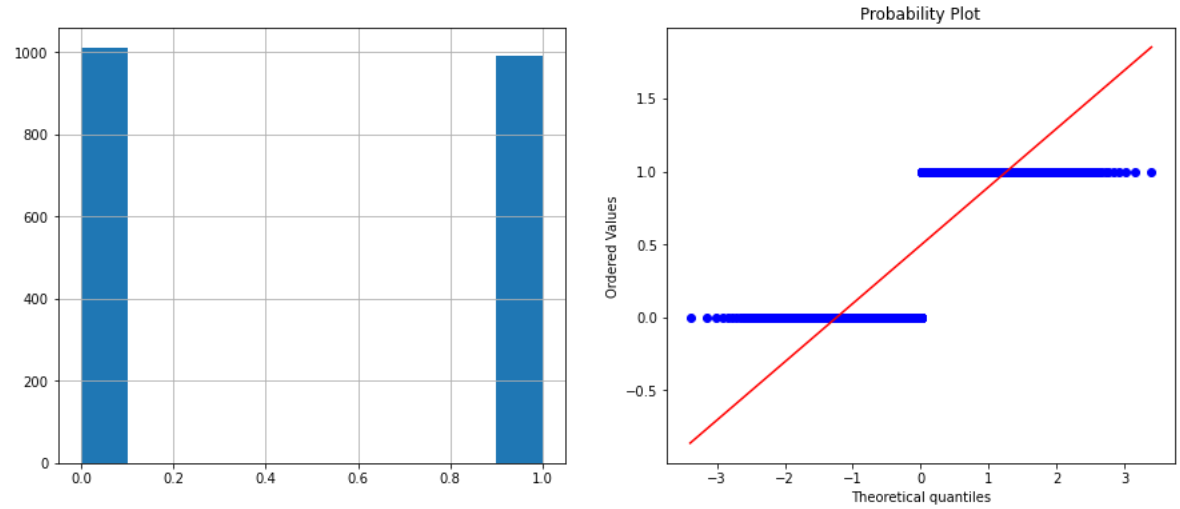


In [20]:
```
for feature in numerical_feature:
    print(feature)
    plt.figure(figsize=(15,6))
    plt.subplot(1, 2, 1)
    df[feature].hist()
    plt.subplot(1, 2, 2)
    stats.probplot(df[feature], dist="norm", plot=plt)
    plt.show()
```

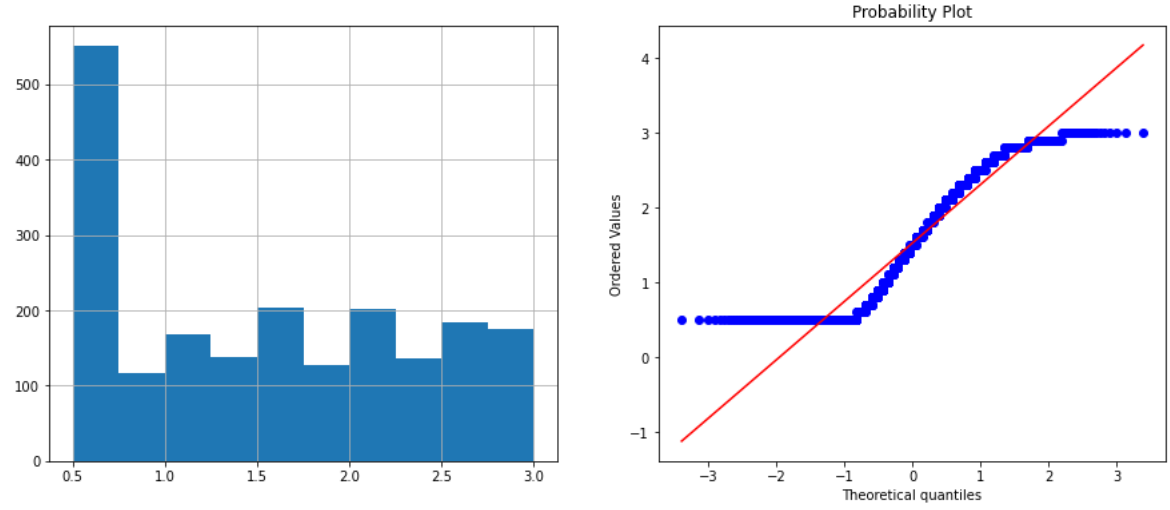
```
battery_power
```



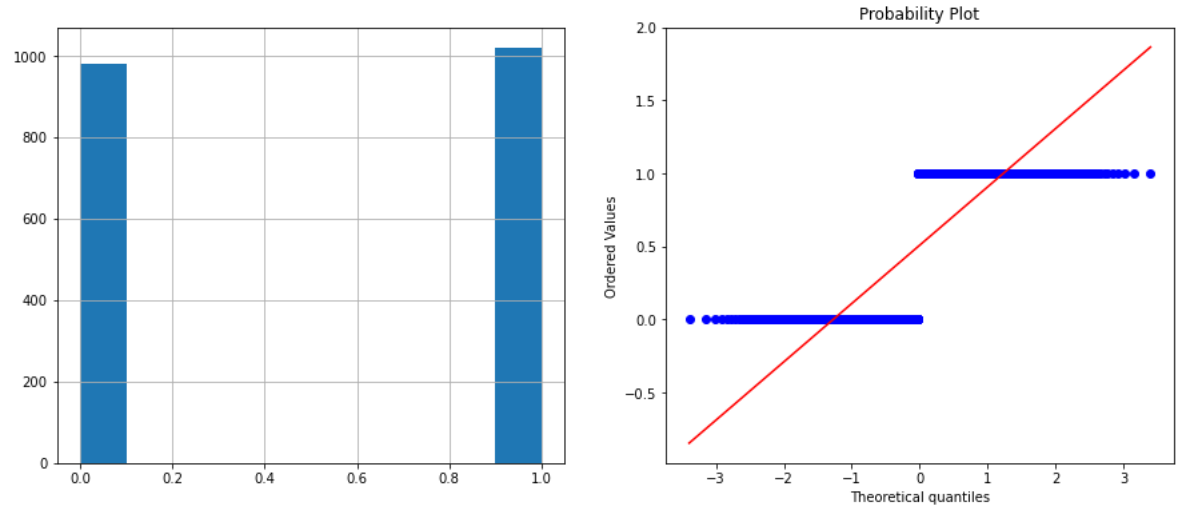

```
blue
```



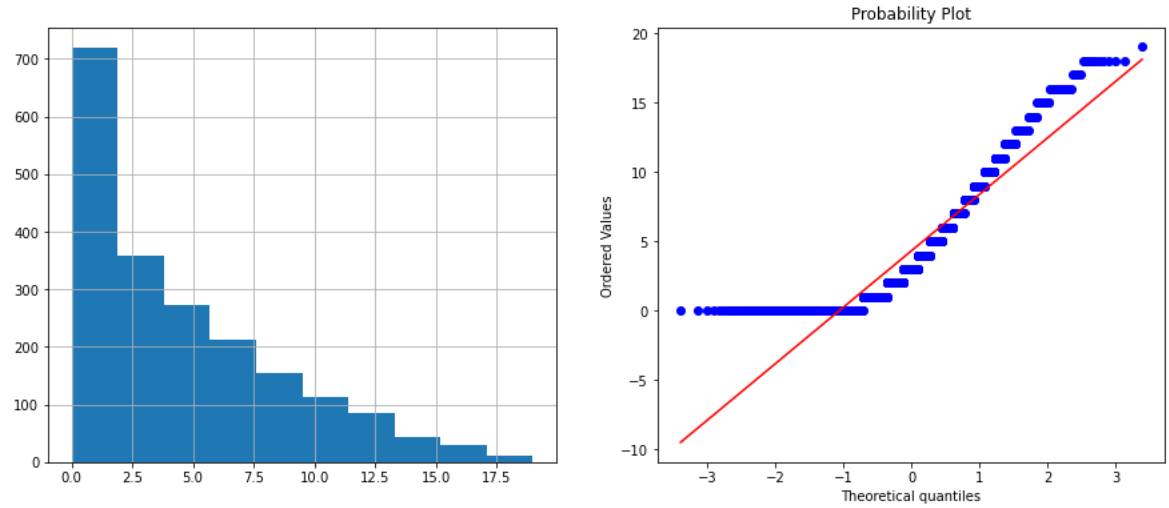

```
clock_speed
```



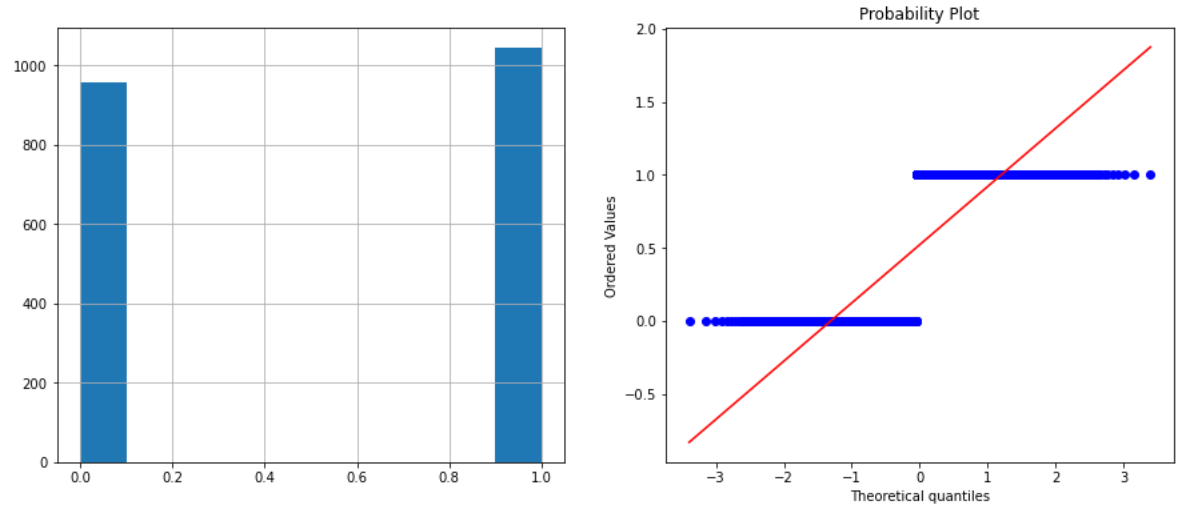
dual_sim



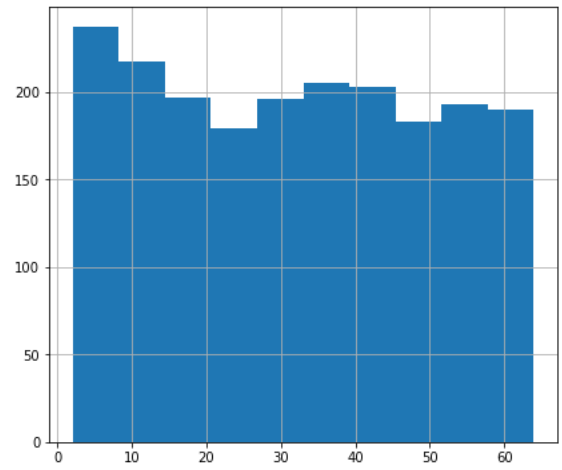
fc

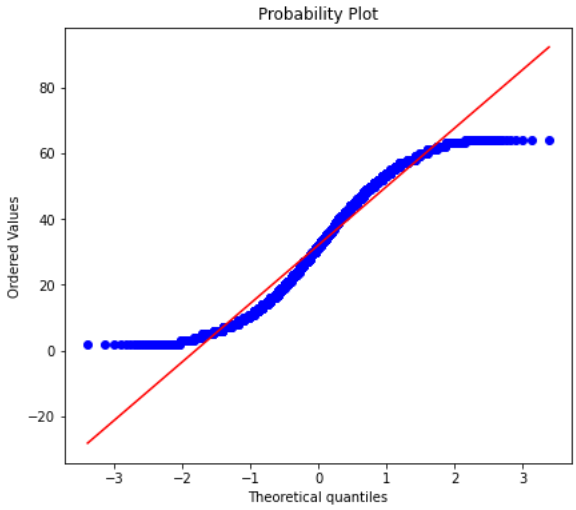


four_g

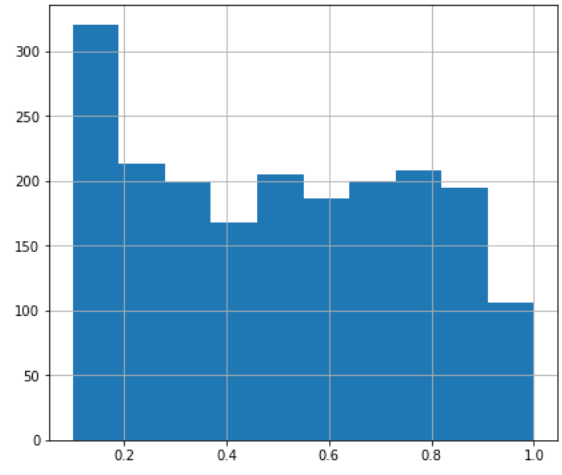


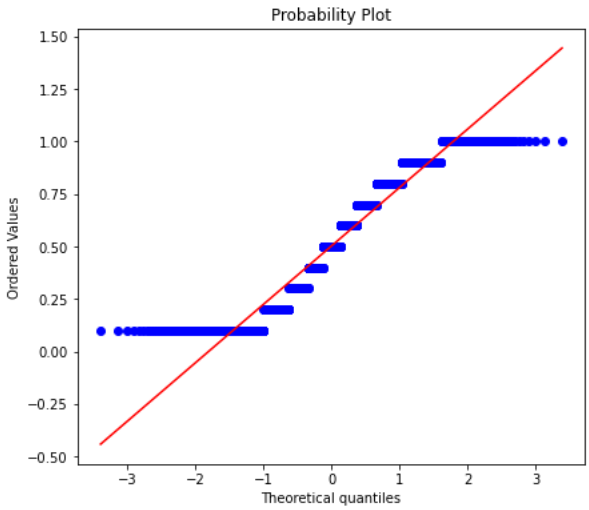
int_memory



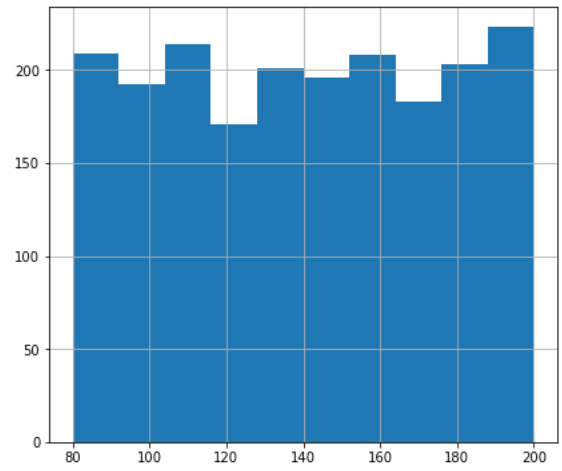


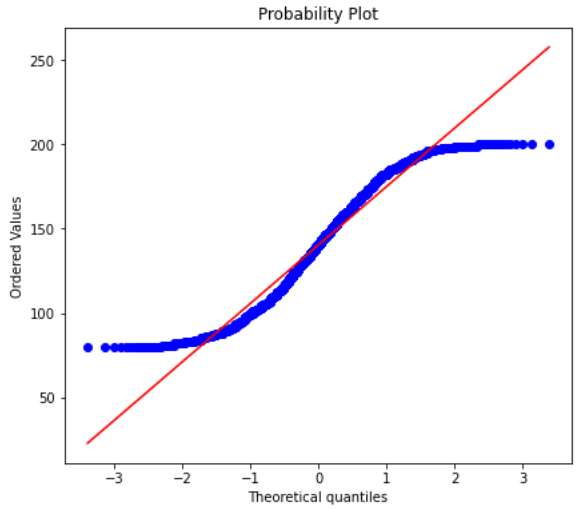
m_dep



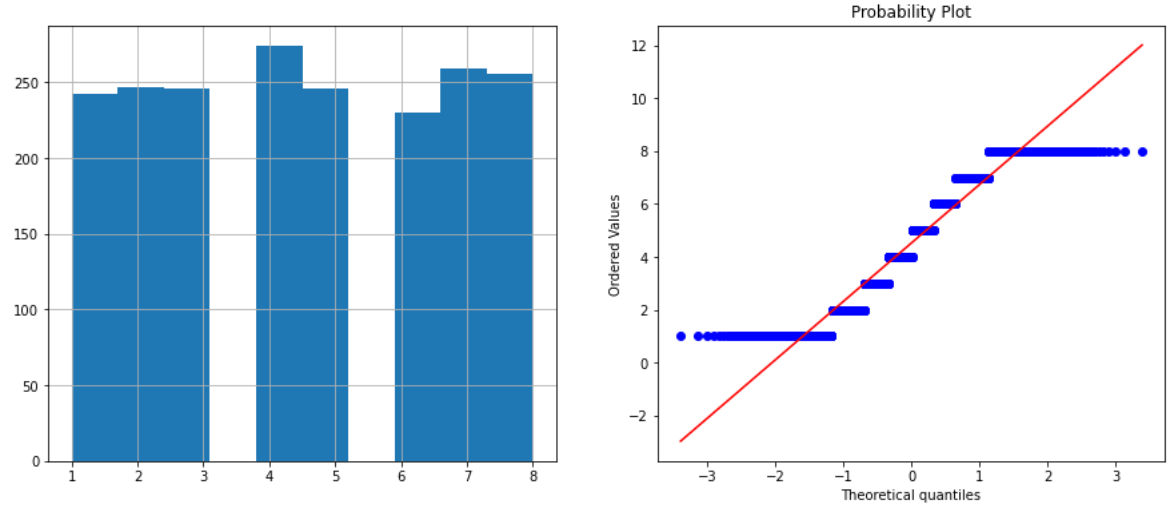


mobile_wt

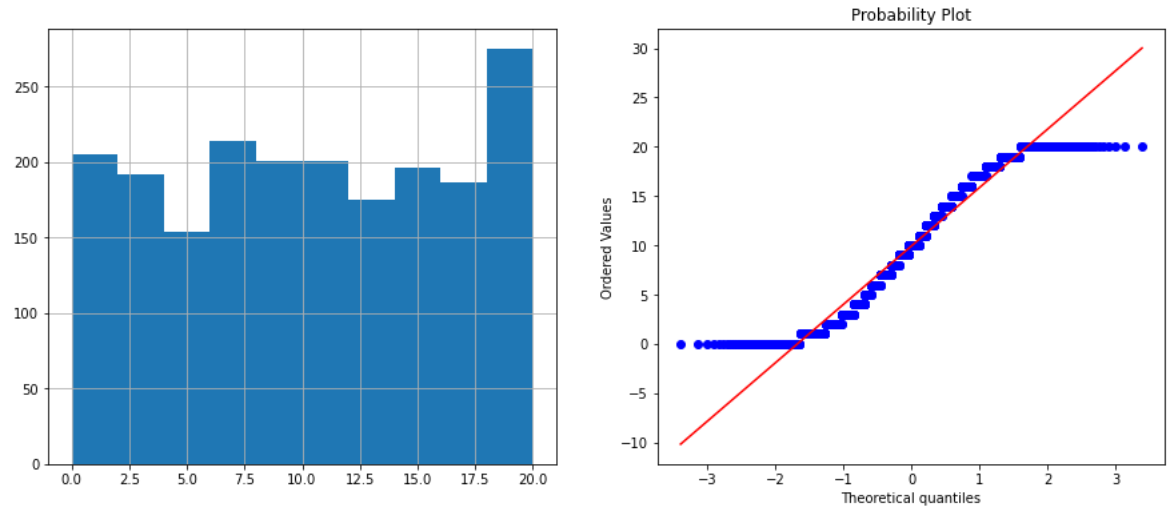




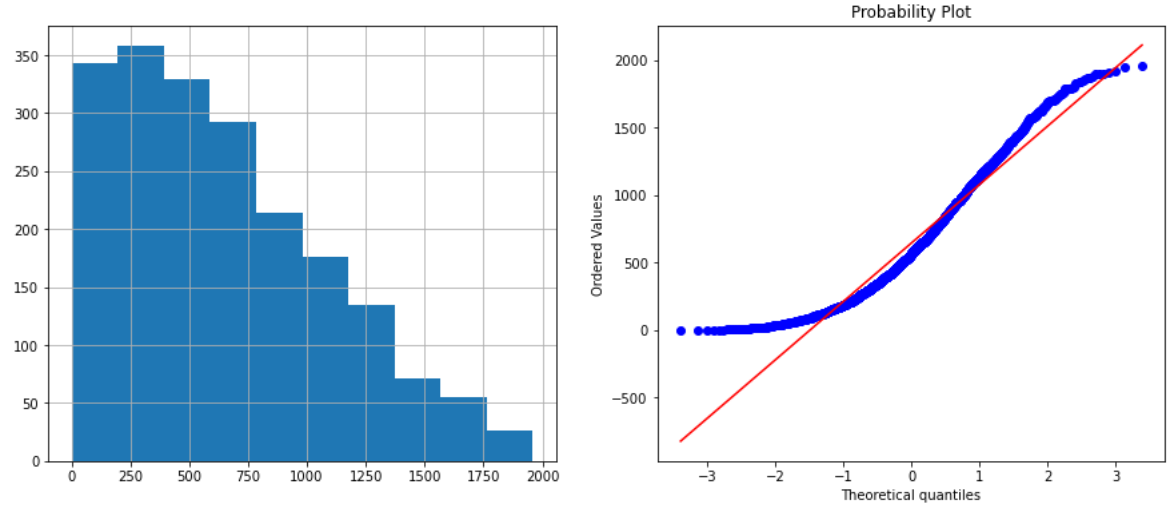
n_cores



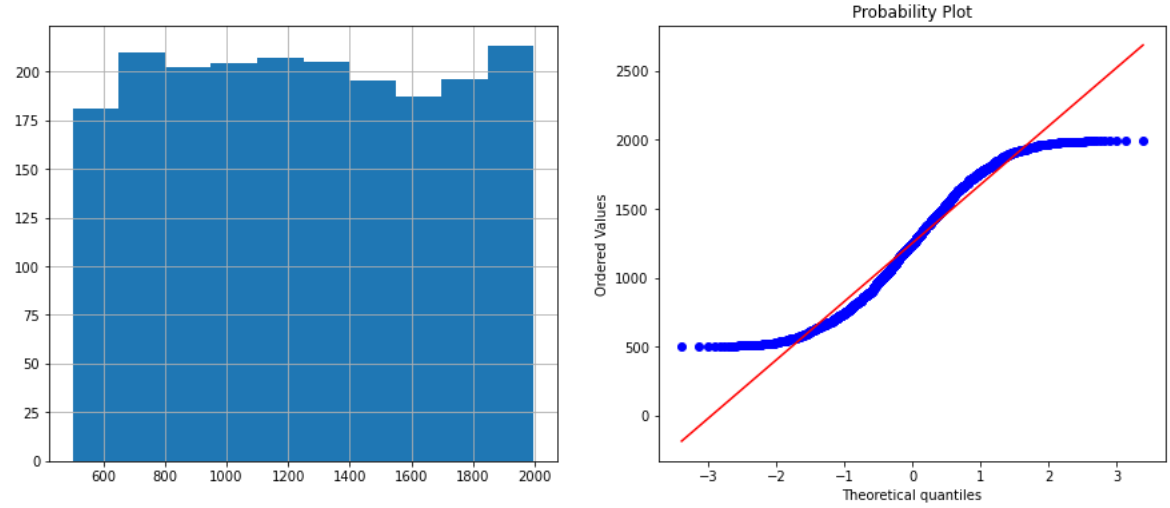
pc



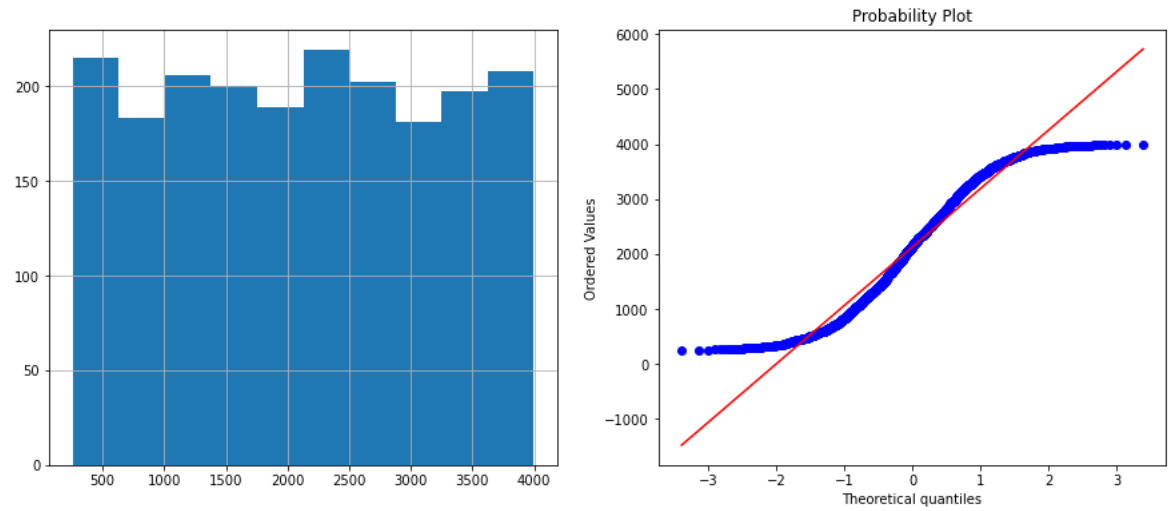
px_height



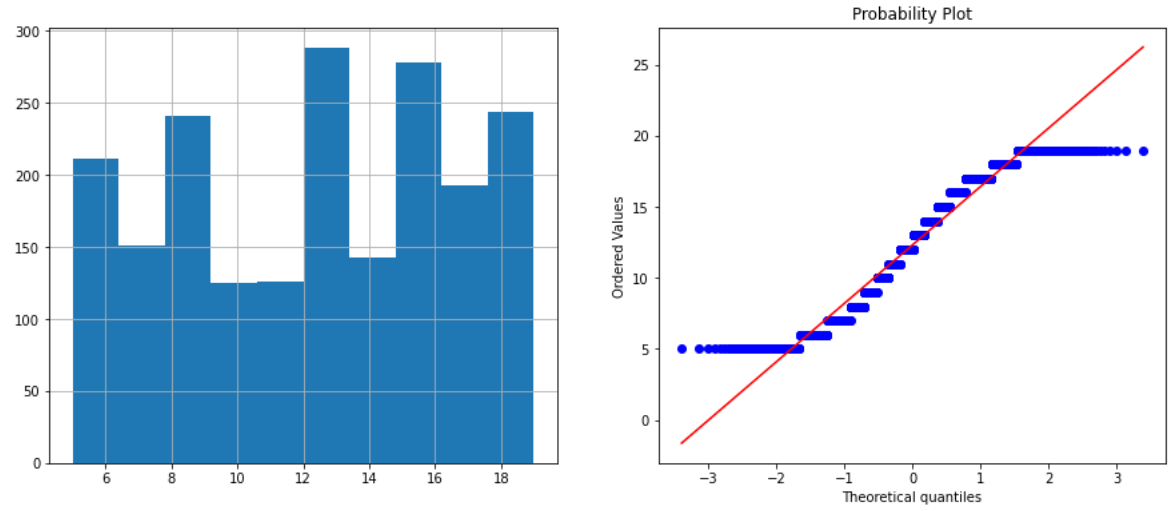
px_width


Probability Plot
Ordered Values
Theoretical quantiles

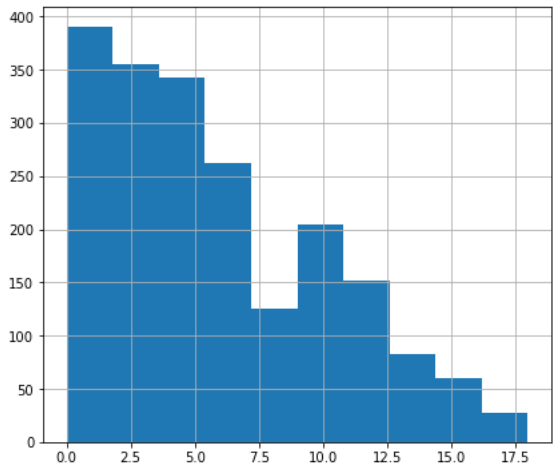
ram

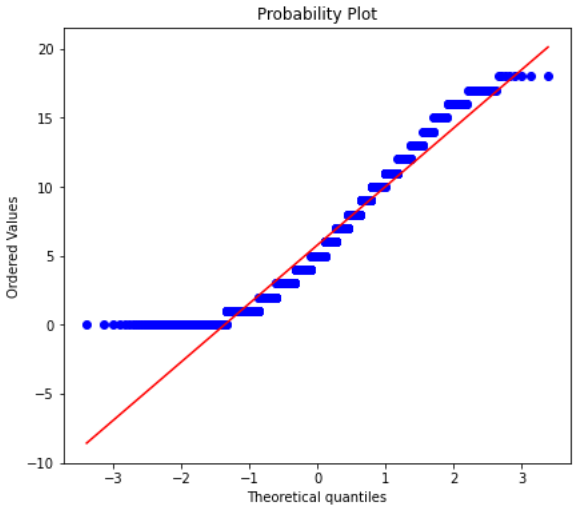

Probability Plot
Ordered Values
Theoretical quantiles

sc_h

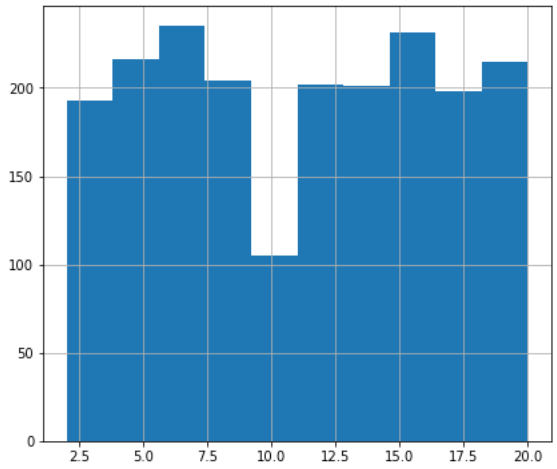

Probability Plot
Ordered Values
Theoretical quantiles

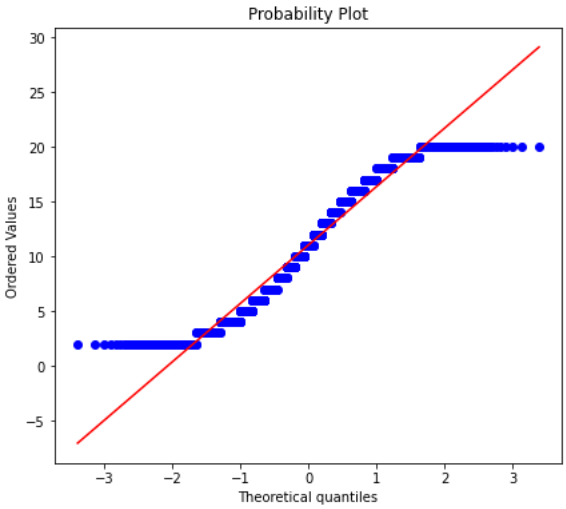
sc_w



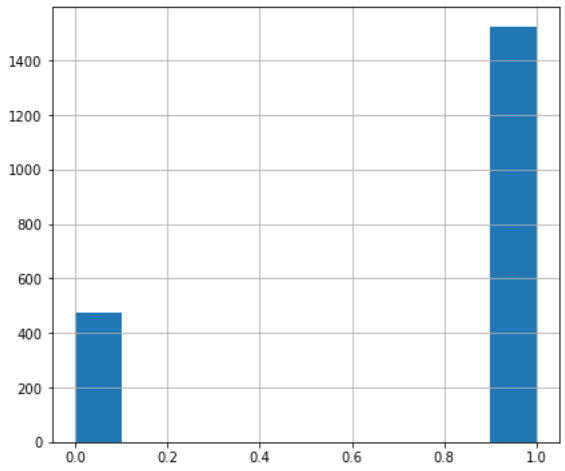


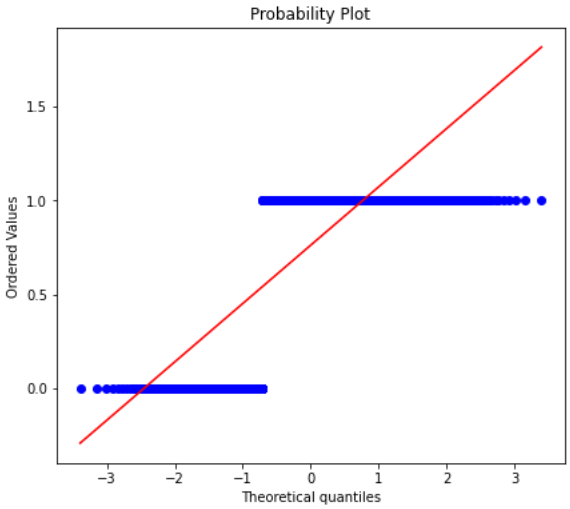
talk_time



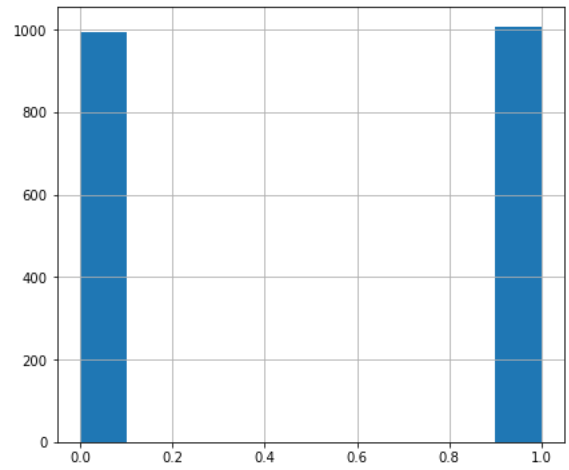


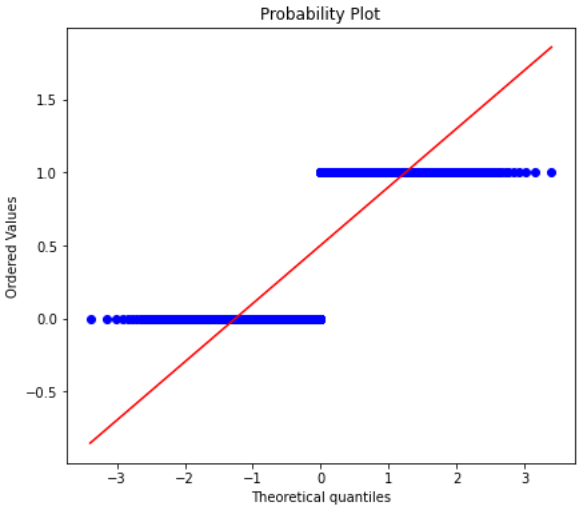
three_g



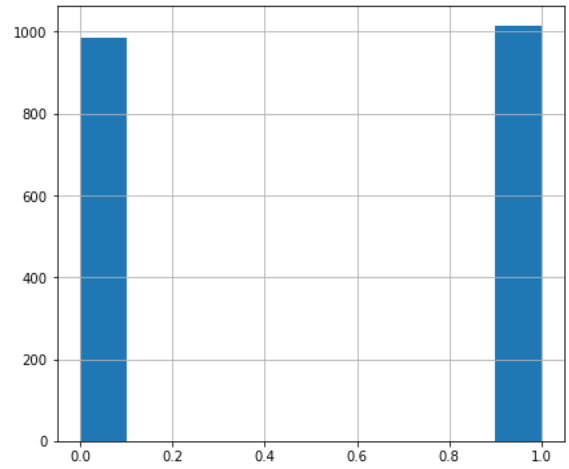


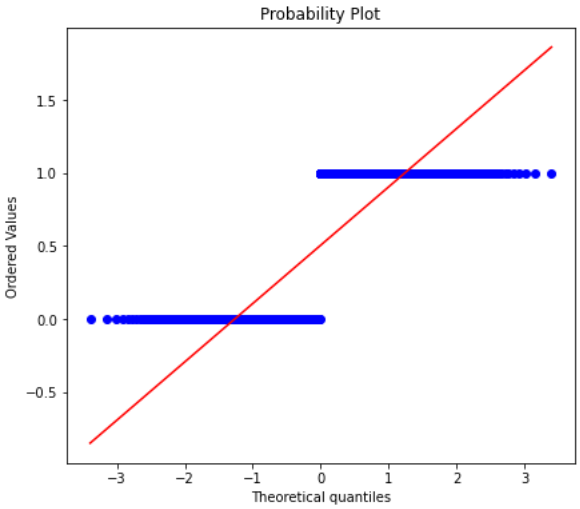
touch_screen



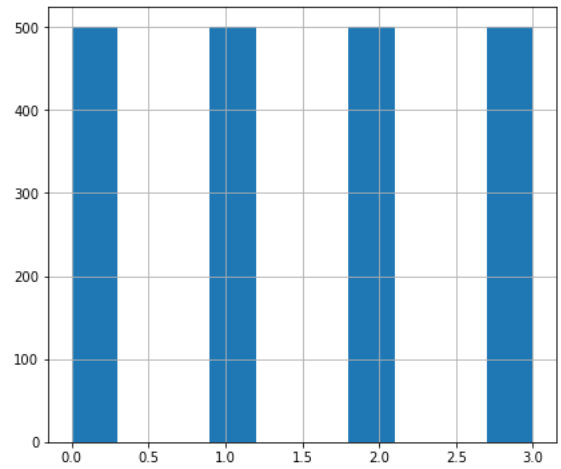


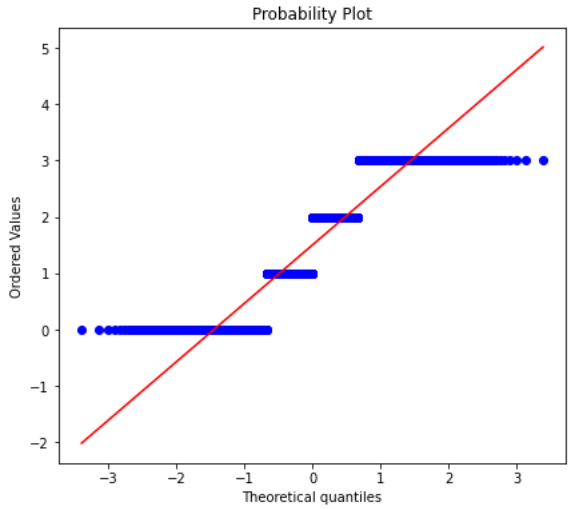
wifi





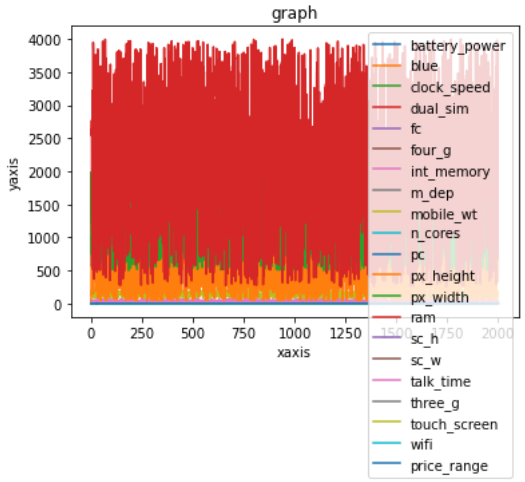
price_range



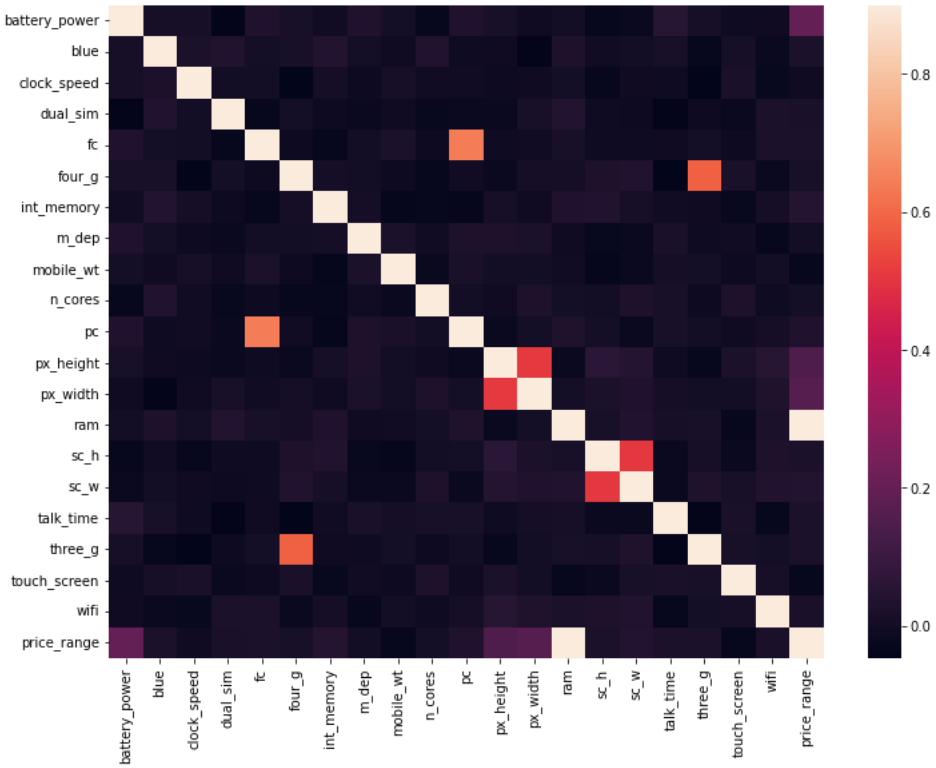


In [39]:
```
df.plot()
plt.xlabel('xaxis')
plt.ylabel('yaxis')
plt.title('graph')
```

Out[39]: Text(0.5, 1.0, 'graph')

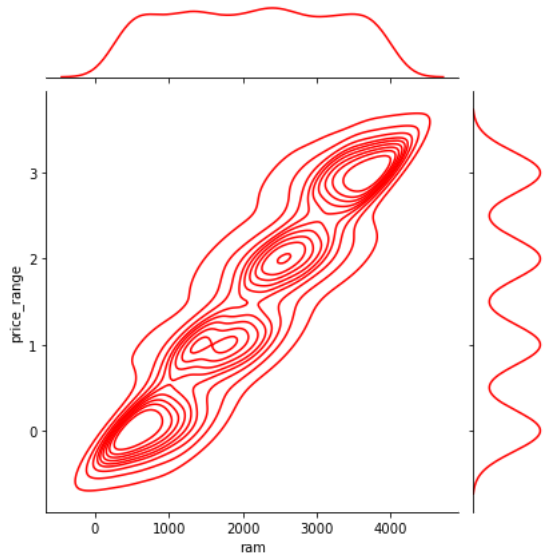


In [22]:
```
corr_matrix = df.corr()
f, ax1 = plt.subplots(figsize=(12,9))
ax1=sns.heatmap(corr_matrix,vmax = 0.9)
```



How does ram is affected by price

In [24]:
```
sns.jointplot(x='ram',y='price_range',data=df,color='red',kind='kde');
```



plotting relation between price range and battery power

In [24]:
```
plt.figure(figsize = (12,6))
sns.barplot(x = "price_range", y = "battery_power", data=df)
plt.show()
```

. . .

plotting relation between price range and pixel height/width

In [26]:
```
plt.figure(figsize = (14,6))
plt.subplot(1,2,1)
sns.barplot(x = "price_range",y ="px_height", data=df, palette ="Reds")
plt.subplot(1,2,2)
sns.barplot(x = "price_range",y = "px_width", data=df, palette ="Blues")
plt.show()
```

. . .

plotting relation between price range and RAM

In [28]:
```
plt.figure(figsize = (12,6))
sns.barplot(x = "price_range",y ="ram",data=df)
plt.show()
```

. . .

plotting relation between price range and 3G/4G

In [30]:
```
plt.figure(figsize = (12,6))
sns.countplot(df["three_g"],hue=df["price_range"],palette="pink")
plt.show()
```

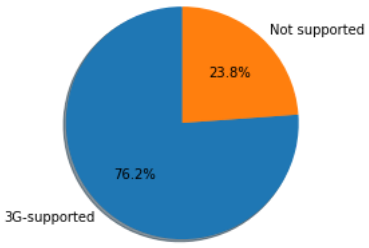
. . .

In [31]:
```
plt.figure(figsize = (12,6))
sns.countplot(df["four_g"],hue=df["price_range"],palette="ocean")
plt.show()
```

. . .

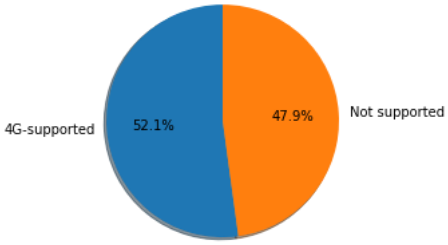
% of Phones which support 3G

In [26]:
```
labels = ["3G-supported",'Not supported']
values=df['three_g'].value_counts().values
fig1, ax1 = plt.subplots()
ax1.pie(values, labels=labels, autopct='%1.1f%%',shadow=True,startangle=90)
plt.show()
```



% of Phones which support 4G

In [27]:
```
labels4g = ["4G-supported",'Not supported']
values4g = df['four_g'].value_counts().values
fig1, ax1 = plt.subplots()
ax1.pie(values4g, labels=labels4g, autopct='%1.1f%%',shadow=True,startangle=90)
plt.show()
```



plotting relation between price range and memory

In [33]:
```
plt.figure(figsize = (12,6))
sns.lineplot(x = "price_range",y ="int_memory", data=df, hue="dual_sim")
plt.show()
```

. . .

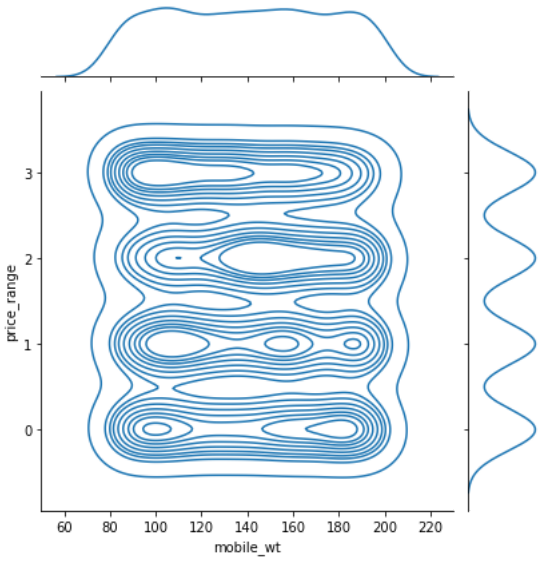
No of Phones vs Camera megapixels of front and primary camera

In [28]:
```
plt.figure(figsize=(10,6))
df['fc'].hist(alpha=0.5,color='blue',label='Front camera')
df['pc'].hist(alpha=0.5,color='red',label='Primary camera')
plt.legend()
plt.xlabel('MegaPixels')
```

. . .

Mobile Weght vs Price range

In [29]:
```
sns.jointplot(x='mobile_wt',y='price_range',data=df,kind='kde');
```



Talk time vs Price range

In [30]:
```
sns.pointplot(y="talk_time", x="price_range", data=df)
```

...

DATA PREPROCESSING

X & Y array

In [12]:
```
x=df.drop(["price_range"],axis=1)

y=df["price_range"]
```

Splitting the data

In [13]:
```
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 = 0)
print("x_train",x_train.shape)
print("x_test",x_test.shape)
print("y_train",y_train.shape)
print("y_test",y_test.shape)
```

```
x_train (1400, 20)
x_test (600, 20)
y_train (1400,)
y_test (600,)
```

Creating & Training Linear Regression Model

In [14]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
```

In [15]:
```
lm.fit(x_train,y_train)
```

Out[15]:
```
LinearRegression()
```

In [16]:
```
lm.score(x_test,y_test)
```

Out[16]:
```
0.9216228455157529
```

Creating & Training KNN Model(K-Nearest Neighbor)

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=10)
knn.fit(x_train,y_train)
```

Out[17]:
```
KNeighborsClassifier(n_neighbors=10)
```

In [18]:
```
knn.score(x_test,y_test)
```

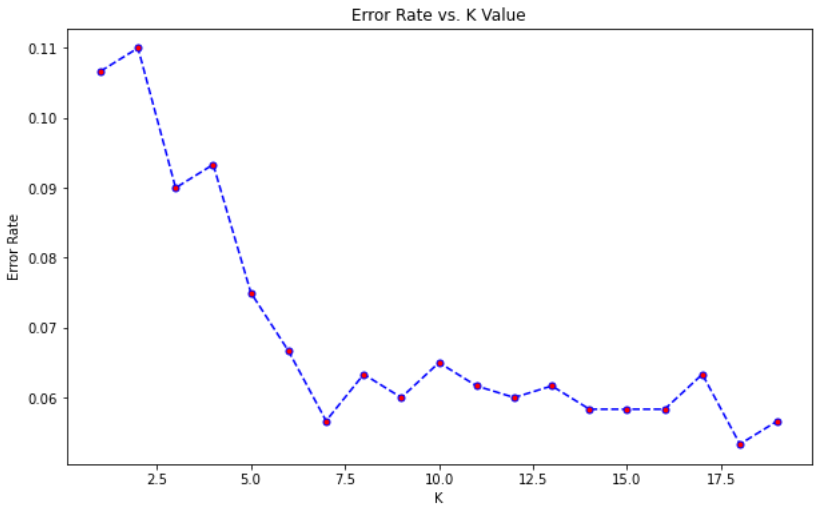
Out[18]:
```
0.935
```

Elbow Method For optimum value of K

In [19]:
```
error_rate = []
for i in range(1,20):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [20]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,20),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=5)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[20]:
```
Text(0, 0.5, 'Error Rate')
```



Creating & Training Logistic Regression Model

In [21]:
```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
```

In [22]:
```
logmodel.fit(x_train,y_train)
```

. . .

In [23]:
```
logmodel.score(x_test,y_test)
```

Out[23]:
```
0.63
```

Creating & Training Decision Tree Model

In [24]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
```

In [25]:
```
dtree.fit(x_train,y_train)
```

Out[25]:
```
DecisionTreeClassifier()
```

In [26]:
```
dtree.score(x_test,y_test)
```

Out[26]:
```
0.8266666666666667
```

Creating & Training Random Tree Model

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(x_train, y_train)
```

Out[27]:
```
RandomForestClassifier(n_estimators=200)
```

In [28]:
```
rfc.score(x_test,y_test)
```

Out[28]:
```
0.875
```

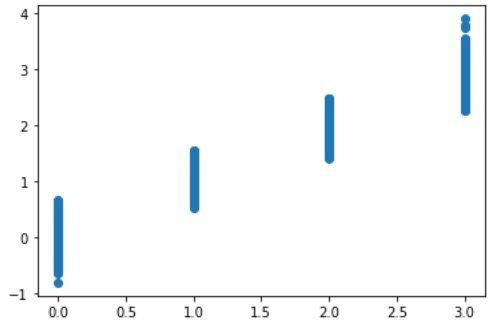
Conclusion: KNN & Linear Regression performed the best

RESULT : Linear Regression

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

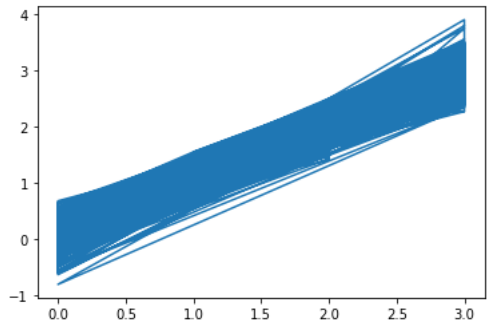
In [30]:
```
plt.scatter(y_test,y_pred)
```

Out[30]:
```
<matplotlib.collections.PathCollection at 0xad0316c820>
```



In [31]:
```
plt.plot(y_test,y_pred)
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0xad031ef250>]
```



RESULT: KNN

In [32]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [33]:
```
pred = knn.predict(x_test)
```

In [34]:
```
print(classification_report(y_test,pred))
```

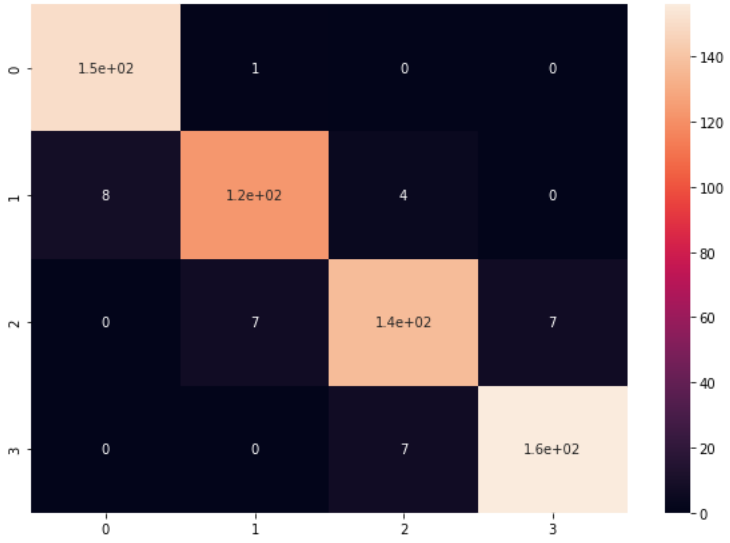
```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       151
           1       0.94      0.91      0.92       135
           2       0.93      0.91      0.92       151
           3       0.96      0.96      0.96       163

    accuracy                           0.94       600
   macro avg       0.94      0.94      0.94       600
weighted avg       0.94      0.94      0.94       600
```

In [35]:
```
matrix=confusion_matrix(y_test,pred)
print(matrix)
```

```
[[150   1   0   0]
 [  8 123   4   0]
 [  0   7 137   7]
 [  0   0   7 156]]
```

In [37]:
```
plt.figure(figsize = (10,7))
sns.heatmap(matrix,annot=True)
```

Out[37]: <AxesSubplot:>



In [ ]: