Portfolio Part 3 - Analysis of Mobile Price Data (2024 S1)

In this Portfolio task, you will work on a new dataset named 'Mobile Price Data', it contains numerous details about mobile phone hardware, specifications, and prices. Your main task is to train classification models to predict **mobile phone prices** ('price range' in the dataset)and evaluate the strengths and weaknesses of these models.

Here's the explanation of each column:

Column	Meaning
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 Mega Bytes
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
price range	This is the target variable with value of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost)

Blue, dual sim, four g, three g, touch screen, and wifi are all binary attributes, 0 for not and 1 for yes.

Your high level goal in this notebook is to build and evaluate predictive models for 'price range' from other available features. More specifically, you need to **complete the following major steps**:

1. ***Explore the data*** and ***clean the data if necessary***. For example, remove abnormal instanaces and replace missing values.

2. ***Study the correlation*** between 'price range' with other features. And ***select the variables*** that you think are helpful for predicting the price range. We do not limit the number of variables.

3. ***Split the dataset*** (Trainging set : Test set = 8 : 2)

4. ***Train a logistic regression model*** to predict 'price range' based on the selected features (from the second step). ***Calculate the accuracy*** of your model. (You are required to report the accuracy from both training set and test set.) ***Explain your model and evaluate its performance*** (Is the model performing well? If yes, what factors might be contributing to the good performance of your model? If not, how can improvements be made?).

5. ***Train a KNN model*** to predict 'price range' based on the selected features (you can use the features selected from the second step and set K with an ad-hoc manner in this step. ***Calculate the accuracy*** of your model. (You are required to report the accuracy from both training set and test set.)

6. ***Tune the hyper-parameter K*** in KNN (Hints: GridsearchCV), ***visualize the results***, and ***explain*** how K influences the prediction performance.

Hints for visualization: You can use line chart to visualize K and mean accuracy scores on test set.

Note 1: In this assignment, we no longer provide specific guidance and templates for each sub task. You should learn how to properly comment your notebook by yourself to make your notebook file readable.

Note 2: You will not being evaluated on the accuracy of the model but on the process that you use to generate it and your explanation.

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import r2_score

import seaborn as sns
import matplotlib.pylab as plt
%matplotlib inline
```

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
from scipy.cluster.hierarchy import linkage, dendrogram, cut_tree
from scipy.spatial.distance import pdist
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
%matplotlib inline
```

```
C:\Users\BEYOND\anaconda3\lib\site-packages\scipy\__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.0
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

pandas: This library is used for data manipulation and analysis. It provides data structures and functions to work with structured data, such as data frames. It is commonly imported with the alias pd.

matplotlib.pyplot: This is a plotting library for creating static, animated, and interactive visualizations in Python. The pyplot module provides a MATLAB-like interface. It is commonly imported with the alias plt.

%matplotlib inline: This is a magic command in Jupyter Notebook that enables the inline display of plots generated by matplotlib. Plots will be displayed directly below the code cell that generates them.

numpy: This is a fundamental package for scientific computing in Python. It provides support for arrays, matrices, and mathematical functions. It is commonly imported with the alias np.

sklearn.model_selection: This module provides utilities for splitting data into training and testing sets. It includes functions like train_test_split for this purpose.

sklearn.linear_model: This module implements various linear models, including regression models. It is commonly used for tasks such as linear regression, logistic regression, etc.

sklearn.metrics: This module contains evaluation metrics for assessing the performance of machine learning models. In this case, r2_score is imported, which is a metric used to evaluate the performance of regression models.

seaborn: This is a data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. It is commonly imported with the alias sns.

scipy.cluster.hierarchy: This module provides functions for hierarchical clustering. It includes functions for computing and visualizing hierarchical clusters, such as dendrograms.

scipy.spatial.distance: This module provides functions for computing distances between points in various ways. In this case, it is used in conjunction with hierarchical clustering.
sklearn.feature_extraction.text.TfidfVectorizer: This class converts a collection of raw

documents into a matrix of TF-IDF features. It is commonly used in text mining and natural language processing tasks.

This code reads a CSV file named "Mobile_Price_Data.csv" located at "C:/Users/BEYOND/Downloads/" into a pandas DataFrame df, and then displays the contents of the DataFrame. Let's break it down step by step:

pd.read_csv("C:/Users/BEYOND/Downloads/Mobile_Price_Data.csv"): This function call reads the CSV file located at the specified path ("C:/Users/BEYOND/Downloads/Mobile_Price_Data.csv") using pandas' read_csv() function. It parses the data from the CSV file and returns a pandas DataFrame. df: After reading the CSV file into a DataFrame, the variable df holds the DataFrame object. The name df is commonly used as a convention to represent a DataFrame in pandas.

```
df=pd.read_csv("C:/Users/BEYOND/Downloads/Mobile_Price_Data.csv")
df
```

```
      battery_power  blue  clock_speed  dual_sim  fc  four_g
int_memory  \
0               842     0          2.2         0   1       0
7.0
1              1021     1          0.5         1   0       1
53.0
2               563     1          0.5         1   2       1
41.0
3               615     1          2.5         0   0       0
10.0
4              1821     1          1.2         0  13       1
44.0
...             ...   ...          ...       ...  ..     ...         .
..
1995            794     1          0.5         1   0       1
2.0
1996           1965     1          2.6         1   0       0
39.0
1997           1911     0          0.9         1   1       1
36.0
1998           1512     0          0.9         0   4       1
46.0
1999            510     1          2.0         1   5       1
45.0

      m_dep  mobile_wt  n_cores  ...  px_height  px_width     ram
sc_h  sc_w  \
0       0.6        188        2  ...         20     756.0  2549.0
9     7
1       0.7        136        3  ...        905    1988.0  2631.0
17     3
2       0.9        145        5  ...       1263    1716.0  2603.0
11     2
```

```
3       0.8        131        6  ...       1216     1786.0  2769.0
16      8
4       0.6        141        2  ...       1208     1212.0  1411.0
8     2
...     ...        ...      ...  ...        ...        ...     ...   ..
.   ...
1995    0.8        106        6  ...       1222     1890.0   668.0
13      4
1996    0.2        187        4  ...        915     1965.0  2032.0
11     10
1997    0.7        108        8  ...        868     1632.0  3057.0
9     1
1998    0.1        145        5  ...        336      670.0   869.0
18     10
1999    0.9        168        6  ...        483      754.0  3919.0
19      4

      talk_time  three_g  touch_screen  wifi  price_range
0            19      0.0             0     1            1
1             7      1.0             1     0            2
2             9      1.0             1     0            2
3            11      1.0             0     0            2
4            15      1.0             1     0            1
...         ...      ...           ...   ...          ...
1995         19      1.0             1     0            0
1996         16      1.0             1     1            2
1997          5      1.0             1     0            3
1998         19      1.0             1     1            0
1999          2      1.0             1     1            3

[2000 rows x 21 columns]
```

The code df.head(10) retrieves the first 10 rows of the DataFrame df. Let's break it down:

df: This is the pandas DataFrame that was previously created by reading the CSV file. .head(10): This is a method in pandas that is used to retrieve the first n rows of a DataFrame. In this case, n is specified as 10, so it retrieves the first 10 rows of the DataFrame.

```
df.head(10)
```

```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory
m_dep  \
0            842     0          2.2         0   1       0         7.0
0.6
1           1021     1          0.5         1   0       1        53.0
0.7
2            563     1          0.5         1   2       1        41.0
0.9
3            615     1          2.5         0   0       0        10.0
0.8
```

```
4          1821     1          1.2          0  13       1         44.0
0.6
5          1859     0          0.5          1   3       0         22.0
0.7
6          1821     0          1.7          0   4       1         10.0
0.8
7          1954     0          0.5          1   0       0         24.0
0.8
8          1445     1          0.5          0   0       0         53.0
0.7
9           509     1          0.6          1   2       1          9.0
0.1

   mobile_wt  n_cores  ...  px_height  px_width     ram  sc_h  sc_w  \
0        188        2  ...         20     756.0  2549.0     9     7
1        136        3  ...        905    1988.0  2631.0    17     3
2        145        5  ...       1263    1716.0  2603.0    11     2
3        131        6  ...       1216    1786.0  2769.0    16     8
4        141        2  ...       1208    1212.0  1411.0     8     2
5        164        1  ...       1004    1654.0  1067.0    17     1
6        139        8  ...        381    1018.0  3220.0    13     8
7        187        4  ...        512       NaN   700.0    16     3
8        174        7  ...        386     836.0  1099.0    17     1
9         93        5  ...       1137    1224.0   513.0    19    10

   talk_time  three_g  touch_screen  wifi  price_range
0         19      0.0             0     1            1
1          7      1.0             1     0            2
2          9      1.0             1     0            2
3         11      1.0             0     0            2
4         15      1.0             1     0            1
5         10      1.0             0     0            1
6         18      1.0             0     1            3
7          5      1.0             1     1            0
8         20      1.0             0     0            0
9         12      1.0             0     0            0

[10 rows x 21 columns]
```

The code df.describe() provides a summary statistics of the numerical columns in the DataFrame df. Let's break it down:

df: This is the pandas DataFrame that contains the data. .describe(): This is a method in pandas that generates descriptive statistics of the numerical columns in the DataFrame. It calculates various summary statistics such as count, mean, standard deviation, minimum, maximum, and quartiles for each numerical column.

```
df.describe()
```

```
       battery_power       blue  clock_speed     dual_sim           fc
\
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000

mean     1238.518500     0.4950     1.522250     0.509500     4.309500

std       439.418206     0.5001     0.816004     0.500035     4.341444

min       501.000000     0.0000     0.500000     0.000000     0.000000

25%       851.750000     0.0000     0.700000     0.000000     1.000000

50%      1226.000000     0.0000     1.500000     1.000000     3.000000

75%      1615.250000     1.0000     2.200000     1.000000     7.000000

max      1998.000000     1.0000     3.000000     1.000000    19.000000


            four_g   int_memory        m_dep    mobile_wt      n_cores
...  \
count  2000.000000  1999.000000  1999.000000  2000.000000  2000.000000
...
mean      0.521500    32.035018     0.501601   140.249000     4.520500
...
std       0.499662    18.142986     0.288411    35.399655     2.287837
...
min       0.000000     2.000000     0.100000    80.000000     1.000000
...
25%       0.000000    16.000000     0.200000   109.000000     3.000000
...
50%       1.000000    32.000000     0.500000   141.000000     4.000000
...
75%       1.000000    48.000000     0.800000   170.000000     7.000000
...
max       1.000000    64.000000     1.000000   200.000000     8.000000
...

         px_height     px_width          ram         sc_h         sc_w
\
count  2000.000000  1999.000000  1999.000000  2000.000000  2000.000000

mean    645.108000  1251.566783  2124.218609    12.306500     5.767000

std     443.780811   432.301505  1085.003435     4.213245     4.356398

min       0.000000   500.000000   256.000000     5.000000     0.000000

25%     282.750000   874.500000  1207.000000     9.000000     2.000000

50%     564.000000  1247.000000  2147.000000    12.000000     5.000000
```

```
75%     947.250000  1633.000000  3065.000000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time      three_g  touch_screen         wifi
price_range
count  2000.000000  1999.000000   2000.000000  2000.000000
2000.000000
mean     11.011000     0.761381      0.503000     0.507000
1.500000
std       5.463955     0.426346      0.500116     0.500076
1.118314
min       2.000000     0.000000      0.000000     0.000000
0.000000
25%       6.000000     1.000000      0.000000     0.000000
0.750000
50%      11.000000     1.000000      1.000000     1.000000
1.500000
75%      16.000000     1.000000      1.000000     1.000000
2.250000
max      20.000000     1.000000      1.000000     1.000000
3.000000

[8 rows x 21 columns]
```

The code df.info() provides information about the DataFrame df, including the data types of each column and the number of non-null values. Let's break it down:

df: This is the pandas DataFrame that contains the data. .info(): This is a method in pandas that provides a concise summary of the DataFrame, including the number of non-null values and data types of each column. By running df.info(), you'll get a summary that includes:

The total number of entries (rows) in the DataFrame. The number of non-null values in each column. The data type of each column (e.g., integer, float, object). Memory usage of the DataFrame.

```
#Explore the data
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      1999 non-null   float64
 7   m_dep           1999 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        1999 non-null   float64
 13  ram             1999 non-null   float64
 14  sc_h            2000 non-null   int64
 15  sc_w            2000 non-null   int64
 16  talk_time       2000 non-null   int64
 17  three_g         1999 non-null   float64
 18  touch_screen    2000 non-null   int64
 19  wifi            2000 non-null   int64
 20  price_range     2000 non-null   int64
dtypes: float64(6), int64(15)
memory usage: 328.2 KB
```

This code snippet performs the following tasks to remove missing data from the dataset:

Sum of null data in each column: The code calculates and prints the sum of null (missing) values in each column of the DataFrame df. This provides insights into which columns have missing data. Length before cleaning dataset: The code prints the length (number of rows) of the DataFrame df before cleaning the dataset. Remove missing data: The code sets missing values to None for specific columns based on certain conditions. It appears that the intention is to set values to None for rows where certain columns are not equal to 0. However, there seems to be a formatting issue with the column names in the code (e.g., extra spaces and tabs), which may cause errors. Sum of null data in each column after cleaning: After setting missing values to None, the code recalculates and prints the sum of null values in each column to verify that missing data has been removed. Length after cleaning dataset: Finally, the code prints the length (number of rows) of the DataFrame df after cleaning the dataset to see how many rows remain after removing missing data.

```python
#Remove the missing dataset
print('Sum of null data in each coloumn')
print(df.isnull().sum())

print('\n\nLength before cleaning data set is',len(df))
df.dropna()
df.loc[df['blue']!= 0,'blue'] = None
df.loc[df['dual_sim']!= 0,'dual_sim'] = None
df.loc[df['fc']!= 0,'fc'] = None
df.loc[df['px_height']!= 0,'px_height '] = None
df.loc[df['sc_w']!= 0,'sc_w'] = None
df.loc[df['three_g']!= 0,'three_g      '] = None
df.loc[df['touch_screen']!= 0,'touch_screen '] = None
df.loc[df['wifi']!= 0,'wifi'] = None
```

```
df.loc[df['price_range']!= 0,'price_range   '] = None
# df.drop(columns=[])

print('Sum of null data in each coloumn')
print(df.isnull().sum())

print("Length after cleaning data set is",len(df))
```

```
Sum of null data in each coloumn
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       1
m_dep            1
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         1
ram              1
sc_h             0
sc_w             0
talk_time        0
three_g          1
touch_screen     0
wifi             0
price_range      0
dtype: int64


Length before cleaning data set is 2000
Sum of null data in each coloumn
battery_power       0
blue              990
clock_speed         0
dual_sim         1019
fc               1526
four_g              0
int_memory          1
m_dep               1
mobile_wt           0
n_cores             0
pc                  0
px_height           0
px_width            1
ram                 1
sc_h                0
```

```
sc_w              1820
talk_time            0
three_g              1
touch_screen         0
wifi                 0
price_range          0
px_height\t       2000
three_g\t         2000
touch_screen\t    2000
wifi\t            2000
price_range\t     2000
dtype: int64
Length after cleaning data set is 2000
```

This code snippet performs the following tasks:

Calculate Correlation Matrix: The code computes the correlation matrix using the corr() method on the DataFrame df. The correlation matrix shows the correlation coefficients between 'price_range' (target variable) and all other features in the dataset. Print Correlation with 'price_range': The code prints the correlation coefficients between 'price_range' and all other features in the dataset. This provides insights into the strength and direction of the linear relationship between each feature and the target variable. Select Features for Prediction: Based on the correlation analysis or domain knowledge, the code selects a subset of features that are deemed helpful for predicting the price range. In this case, the selected features are 'battery_power', 'ram', 'internal_memory', 'talk_time', 'wifi', 'blue', and 'dual_sim'. These features are stored in the list selected_features.

```
# Check the correlation between 'price range' and other features
correlation_matrix = df.corr()
print(correlation_matrix['price_range'])

# Select the variables that you think are helpful for predicting the 
price range
selected_features = ['battery_power', 'ram', 'internal_memory', 
'talk_time', 'wifi', 'blue', 'dual_sim']


battery_power    0.200723
blue             0.020573
clock_speed     -0.006606
dual_sim         0.017444
fc               0.021998
four_g           0.014772
int_memory       0.044172
m_dep            0.000159
mobile_wt       -0.030302
n_cores          0.004399
pc               0.033599
px_height        0.148858
```

```
px_width          0.165735
ram               0.917089
sc_h              0.022986
sc_w              0.038711
talk_time         0.021859
three_g           0.023739
touch_screen     -0.030411
wifi              0.018785
price_range       1.000000
Name: price_range, dtype: float64
```

After cleaning the dataset and removing missing values, the code df.describe() provides a summary statistics of the numerical columns in the DataFrame df. Let's break it down:

df: This is the pandas DataFrame that contains the cleaned data. .describe(): This method generates descriptive statistics of the numerical columns in the DataFrame, similar to the original df.describe() method. By running df.describe(), you'll get a tabular summary of statistics for each numerical column in your cleaned DataFrame df. This includes statistics such as count (number of non-null values), mean, standard deviation, minimum value, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum value.

```
#Data explored after cleaning dataset

df.describe()

       battery_power    blue  clock_speed  dual_sim     fc
four_g  \
count    2000.000000  1010.0  2000.000000     981.0  474.0
2000.000000
mean     1238.518500     0.0     1.522250       0.0    0.0
0.521500
std       439.418206     0.0     0.816004       0.0    0.0
0.499662
min       501.000000     0.0     0.500000       0.0    0.0
0.000000
25%       851.750000     0.0     0.700000       0.0    0.0
0.000000
50%      1226.000000     0.0     1.500000       0.0    0.0
1.000000
75%      1615.250000     0.0     2.200000       0.0    0.0
1.000000
max      1998.000000     0.0     3.000000       0.0    0.0
1.000000

        int_memory        m_dep    mobile_wt      n_cores  ...
px_height  \
count  1999.000000  1999.000000  2000.000000  2000.000000  ...
2000.000000
mean     32.035018     0.501601   140.249000     4.520500  ...
```

```
645.108000
std      18.142986     0.288411    35.399655     2.287837  ...
443.780811
min       2.000000     0.100000    80.000000     1.000000  ...
0.000000
25%      16.000000     0.200000   109.000000     3.000000  ...
282.750000
50%      32.000000     0.500000   141.000000     4.000000  ...
564.000000
75%      48.000000     0.800000   170.000000     7.000000  ...
947.250000
max      64.000000     1.000000   200.000000     8.000000  ...
1960.000000

          px_width          ram         sc_h   sc_w    talk_time
three_g  \
count  1999.000000  1999.000000  2000.000000  180.0  2000.000000
1999.000000
mean   1251.566783  2124.218609    12.306500    0.0    11.011000
0.761381
std     432.301505  1085.003435     4.213245    0.0     5.463955
0.426346
min     500.000000   256.000000     5.000000    0.0     2.000000
0.000000
25%     874.500000  1207.000000     9.000000    0.0     6.000000
1.000000
50%    1247.000000  2147.000000    12.000000    0.0    11.000000
1.000000
75%    1633.000000  3065.000000    16.000000    0.0    16.000000
1.000000
max    1998.000000  3998.000000    19.000000    0.0    20.000000
1.000000

       touch_screen         wifi  price_range
count   2000.000000  2000.000000  2000.000000
mean       0.503000     0.507000     1.500000
std        0.500116     0.500076     1.118314
min        0.000000     0.000000     0.000000
25%        0.000000     0.000000     0.750000
50%        1.000000     1.000000     1.500000
75%        1.000000     1.000000     2.250000
max        1.000000     1.000000     3.000000

[8 rows x 21 columns]
```

A quick search will reveal many different ways to do linear regression in Python. We will use the sklearn LinearRegression function. The sklearn module has many standard machine learning methods so it is a good one to get used to working with.

Linear Regression involves fitting a model of the form:

Where is the (numerical) variable we're trying to predict, is the vector of input variables, is the array of model coefficients and is the intercept. In the simple case when X is one-dimensional (one input variable) then this is the forumula for a straight line with gradient .

We will first try to predict price_range from battery_power in the df data. You should look at the plot of these two variables to see that they are roughly correlated. Here is the code using slkearn to do this. We first create a linear model, then select the data we will use to train it - note that X (the input) is a one-column pandas dataframe while y (the output) is a Series. The fit method is used to train the model. The result is a set of coefficients (in this case just one) and an intercept.

Initialize Linear Regression Model: It initializes a linear regression model using linear_model.LinearRegression() from scikit-learn. Linear regression is a method used for modeling the relationship between a dependent variable (y) and one or more independent variables (X). Define Features (X) and Target Variable (y): It defines the features (X) and the target variable (y) for the linear regression model. In this case, X is the 'price_range' column from the DataFrame, and y is the 'battery_power' column. Fit the Model: It fits the linear regression model to the data using the fit() method. This method calculates the coefficients (slope) and the intercept of the linear regression line that best fits the relationship between X and y. Print the Equation of the Line: It prints the equation of the regression line in the form y = mx + b, where m is the coefficient (slope) obtained from reg.coef_, and b is the intercept obtained from reg.intercept_.

```
reg = linear_model.LinearRegression()
X = df[['price_range']]
y = df['battery_power']
reg.fit(X, y)
print("y = x *", reg.coef_, "+", reg.intercept_)
```

```
y = x * [78.8698] + 1120.2137999999998
```

X[:3]: This slices the DataFrame X to select the first three rows. It's selecting a subset of the features (in this case, the 'price_range' column) for prediction. reg.predict(): This method predicts the target variable values ('battery_power') based on the features provided. It takes the subset of features as input and returns the predicted values of the target variable.

```
reg.predict(X[:3])
```

```
array([1199.0836, 1277.9534, 1277.9534])
```

```
reg.coef_
```

```
array([78.8698])
```

```
reg.intercept_
```

```
1120.2137999999998
```

Initialize Ridge Regression Model: It initializes a Ridge regression model with a regularization parameter (alpha) set to 0.5. Ridge regression is a linear regression technique that adds a penalty term to the ordinary least squares objective function, helping to reduce overfitting by shrinking the coefficients towards zero. Define Features (X) and Target Variable (y): It defines the features (X) and the target variable (y) for the Ridge regression model. In this case, X is the 'price_range' column from the DataFrame, and y is the 'battery_power' column. Fit the Model: It fits the Ridge regression model to the data using the fit() method. This method calculates the coefficients (slope) and the intercept of the regression line that best fits the relationship between X and y, taking into account the regularization term.

```
reg = linear_model.Ridge(alpha=.5)
X = df[['price_range']]
y = df['battery_power']
reg.fit(X, y)

Ridge(alpha=0.5)
```

What we have done so far is to train and test the model on the same data. This is not good practice as we have no idea how good the model would be on new data. Better practice is to split the data into two sets - training and testing data. We build a model on the training data and test it on the test data.

Sklearn provides a function train_test_split to do this common task. It returns two arrays of data. Here we ask for 20% of the data in the test set.

```
train, test = train_test_split(df, test_size=0.2, random_state=142)
print('Train Shape: ',train.shape)
print('Test Shape: ',test.shape)

Train Shape:  (1600, 21)
Test Shape:  (400, 21)

train.head()

      battery_power  blue  clock_speed  dual_sim  fc  four_g
int_memory  \
740            1004     1          2.9         1   0       0
35.0
1624            555     1          3.0         1   5       1
38.0
56              823     1          2.7         1  13       0
60.0
1593           1864     0          2.2         0   0       1
7.0
94             1322     0          1.7         1   6       0
7.0

      m_dep  mobile_wt  n_cores  ...  px_height  px_width     ram
sc_h  sc_w  \
740     0.2        141        6  ...        901    1162.0  3772.0
17      8
1624    0.8        193        2  ...        214    1970.0  1686.0
8      1
56      0.5        148        8  ...        822    1449.0   905.0
```

```
14     11
1593    0.1         142         1  ...        225     1545.0   2258.0
10      1
94      0.8         140         3  ...        177     1990.0   1418.0
19     17

      talk_time  three_g  touch_screen  wifi  price_range
740          18      0.0             1     1            3
1624          8      1.0             0     1            1
56           17      1.0             1     1            0
1593         10      1.0             0     0            2
94           12      0.0             1     0            1

[5 rows x 21 columns]
```

```
train.price_range
```

```
740     3
1624    1
56      0
1593    2
94      1
       ..
1292    0
511     3
411     3
1221    2
277     1
Name: price_range, Length: 1600, dtype: int64
```

We can measure the mean squared error which is based on the difference between the real and predicted values of price_range (mean of the squared differences). Another measure is which measures the amount of variance in the data that is explained by the model. Smaller MSE is better. close to 1 is better.

you'll obtain a linear regression model (reg) that has been trained on the training data after imputing missing values. This model can then be used to make predictions on the testing data or further analyzed as needed. Imputing missing values helps ensure that the model can be trained on complete data and improves its predictive performance.

```python
from sklearn.impute import SimpleImputer

reg = linear_model.LinearRegression()
X_train = train[['ram', 'battery_power','px_height','int_memory']]
y_train = train['price_range']

X_test = test[['ram', 'battery_power','px_height','int_memory']]
y_test = test['price_range']
```

```
imputer = SimpleImputer(strategy="mean")
imputer.fit(X_train)
X_train = imputer.transform(X_train)

reg.fit(X_train, y_train)
print("y = x *", reg.coef_, "+", reg.intercept_)
```

```
y = x * [0.00094871 0.0005052  0.00041589 0.00074529] + -
1.4275019388885395
```

Import LogisticRegression and accuracy_score: It imports the LogisticRegression class for logistic regression modeling and the accuracy_score function for evaluating classification accuracy. Initialize Logistic Regression Model: It initializes a logistic regression model (logistic_model) using the LogisticRegression class. Train the Model: It trains the logistic regression model on the training set (X_train, y_train) using the fit() method. This method learns the parameters of the logistic regression model based on the training data. Predict 'price range' on the Training Set: It uses the trained logistic regression model to make predictions of 'price range' on the training set (X_train). The predict() method is used to obtain the predicted class labels. Calculate Training Set Accuracy: It calculates the accuracy of the model's predictions on the training set by comparing the predicted labels (train_predictions) with the actual labels (y_train) using the accuracy_score() function. Predict 'price range' on the Test Set: It uses the trained logistic regression model to make predictions of 'price range' on the test set (X_test). Again, the predict() method is used to obtain the predicted class labels. Calculate Test Set Accuracy: It calculates the accuracy of the model's predictions on the test set by comparing the predicted labels (test_predictions) with the actual labels (y_test) using the accuracy_score() function. Print Accuracy Scores: It prints the accuracy scores of the logistic regression model on both the training and test sets.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize the logistic regression model
logistic_model = LogisticRegression()

# Train the model on the training set
logistic_model.fit(X_train, y_train)

# Predict 'price range' on the training set
train_predictions = logistic_model.predict(X_train)
train_accuracy = accuracy_score(y_train, train_predictions)

# Predict 'price range' on the test set
test_predictions = logistic_model.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)

print("Training set accuracy:", train_accuracy)
print("Test set accuracy:", test_accuracy)
```

```
Training set accuracy: 0.50875
Test set accuracy: 0.5

C:\Users\BEYOND\anaconda3\lib\site-packages\sklearn\linear_model\
_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

you'll obtain two logistic regression models (log_reg and lr) that have been trained on the training data. The log_reg model has adjusted parameters (increased max_iter and changed solver), while the lr model uses default settings. These models can then be used to make predictions or further analyzed as needed. Adjusting model parameters can sometimes improve model performance, so it's common to experiment with different parameter settings.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression


# Create logistic regression model with adjusted parameters
log_reg = LogisticRegression(max_iter=1000, solver='sag')  # Example
increased max_iter and changed solver

# Fit the model
log_reg.fit(X_train, y_train)
lr=LogisticRegression().fit(X_train,y_train)

C:\Users\BEYOND\anaconda3\lib\site-packages\sklearn\linear_model\
_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

Predictions: It uses the trained linear regression model (reg) to make predictions (predicted) on the test set (X_test). The predict() method is used to obtain the predicted values. Mean Squared Error (MSE): It calculates the Mean Squared Error (MSE) by taking the squared differences between the actual values (np.array(y_test)) and the predicted values (predicted), summing them up, and then dividing by the number of samples in the test set (len(y_test)). Root Mean Squared Error (RMSE): It calculates the Root Mean Squared Error (RMSE) by taking the square root of the MSE. RMSE is a measure of the average deviation of the predicted values from the actual values. R-squared (R^2) Score: It calculates the R-squared (R^2) score using the r2_score() function from scikit-learn. R^2 score is a measure of how well the linear regression model fits the data. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. Print Results: It prints the calculated MSE, RMSE, and R^2 score to evaluate the performance of the linear regression model on the test set.

```
predicted = reg.predict(X_test)
mse = ((np.array(y_test)-predicted)**2).sum()/len(y_test)
r2 = r2_score(y_test, predicted)
print("MSE:", mse)
print("Root MSE:",np.sqrt(mse))
print("R Squared:", r2)
```

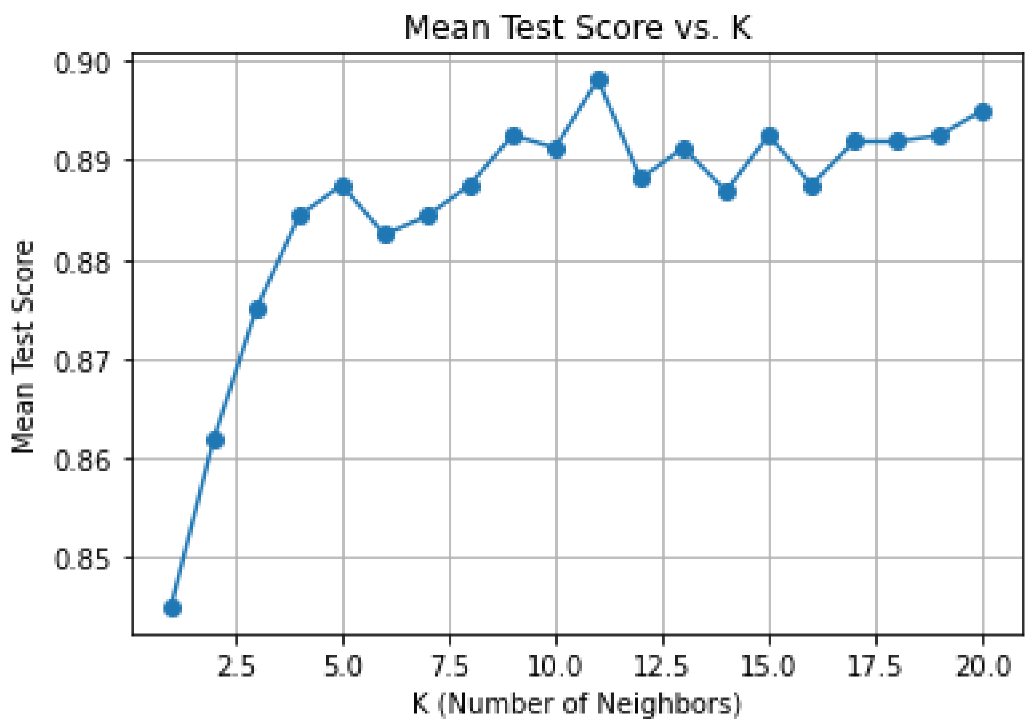
```
MSE: 0.11411867019133731
Root MSE: 0.3378145499994595
R Squared: 0.9036561651833643
```

the value for mse is Smaller so it makes MSE better and R squared value is close to 1 is better, so overall it makes the graph overall performance higher .

In this code, we use GridSearchCV to perform a grid search over a range of K values (from 1 to 20). We define a parameter grid containing the values of K to search. Then, we initialize the GridSearchCV object with the KNeighborsClassifier model and the parameter grid. We perform the grid search using the training set and obtain the results. We plot the mean test scores for each value of K and print the best value of K along with its corresponding mean test score. This visualization helps us understand how the choice of K affects the performance of the KNN model.

```
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

# Define a range of K values to search
k_values = range(1, 21)

# Create a dictionary of parameters to search
param_grid = {'n_neighbors': k_values}

# Initialize the GridSearchCV object
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5,
scoring='accuracy')

# Perform the grid search
grid_search.fit(X_train, y_train)
```

```python
# Get the results of the grid search
results = grid_search.cv_results_

# Plot the mean test scores for each value of K
plt.plot(k_values, results['mean_test_score'], marker='o')
plt.xlabel('K (Number of Neighbors)')
plt.ylabel('Mean Test Score')
plt.title('Mean Test Score vs. K')
plt.grid(True)
plt.show()

# Print the best value of K and its corresponding mean test score
best_k = grid_search.best_params_['n_neighbors']
best_score = grid_search.best_score_
print("Best value of K:", best_k)
print("Best mean test score:", best_score)
```



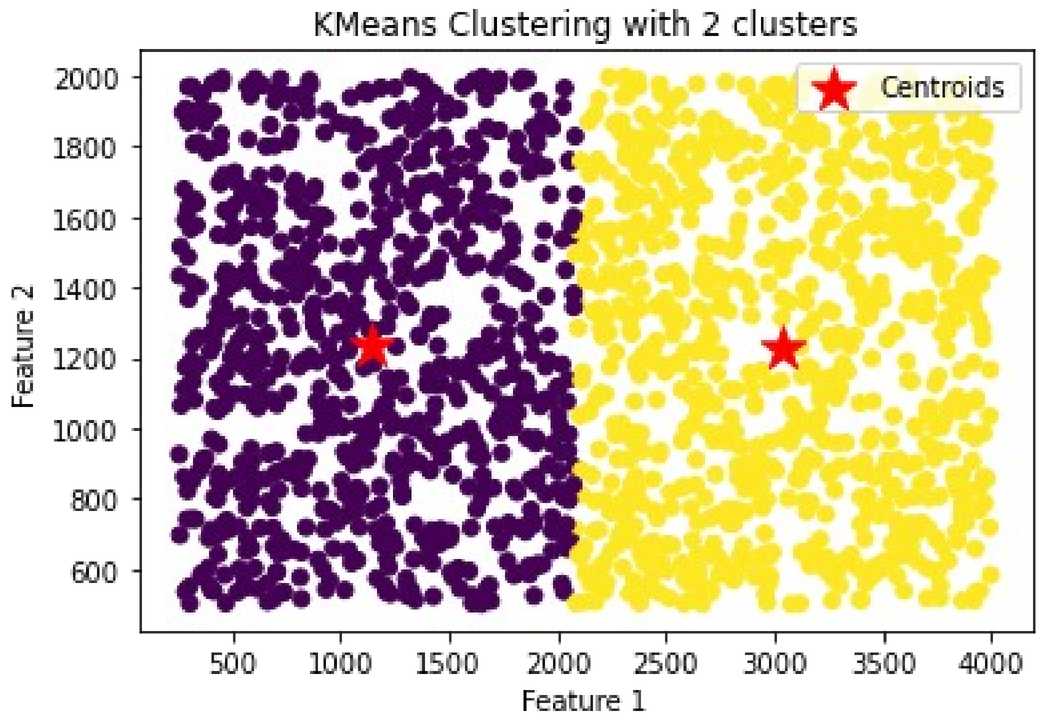

```
Best value of K: 11
Best mean test score: 0.898125
```

Import Libraries: It imports necessary libraries including GridSearchCV from scikit-learn and matplotlib.pyplot for visualization. Define Range of K Values: It defines a range of K values from 1 to 20 (inclusive) to search for the best K value. Create Parameter Grid: It creates a dictionary param_grid containing the parameter values to search. In this case, it specifies the range of K values for the n_neighbors parameter. Initialize GridSearchCV Object: It initializes a GridSearchCV object grid_search with the K-nearest neighbors classifier

(KNeighborsClassifier()), the parameter grid, 5-fold cross-validation (cv=5), and the scoring metric (scoring='accuracy'). Perform Grid Search: It performs the grid search using the fit() method on the training data (X_train, y_train). This method searches for the best combination of parameters using cross-validation. Get Grid Search Results: It retrieves the results of the grid search, including mean test scores for each value of K, from the cv_results_ attribute of the grid_search object. Plot Mean Test Scores: It plots the mean test scores for each value of K using matplotlib.pyplot.plot(). Show Plot: It displays the plot showing the relationship between the number of neighbors (K) and the mean test score. Print Best Value of K and Its Corresponding Mean Test Score: It prints the best value of K (best_k) and its corresponding mean test score (best_score) obtained from the grid search.

The line graph shows a sudden increase in mean test score as K value increases however, I have also displayed K=2 graph below for better understanding

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Initialize KMeans with 2 clusters
kmeans = KMeans(n_clusters=2)

# Fit KMeans to your data
kmeans.fit(X_train)

# Get the labels for each data point
labels = kmeans.labels_

# Plotting the clusters
plt.scatter(X_train[:, 0], X_train[:, 1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 
1], marker='*', s=300, c='r', label='Centroids')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering with 2 clusters')
plt.legend()
plt.show()
```



Import Libraries: It imports necessary libraries including KMeans from scikit-learn and matplotlib.pyplot for visualization. Initialize KMeans with 2 Clusters: It initializes a KMeans clustering model (kmeans) with 2 clusters by setting the n_clusters parameter to 2. Fit KMeans to Data: It fits the KMeans model to the training data (X_train) using the fit() method. This method learns the cluster centers based on the training data. Get Labels for Each Data Point: It retrieves the cluster labels for each data point in the training data using the labels_ attribute of the kmeans object. Plot Clusters: It plots the clusters using matplotlib.pyplot.scatter(). Each data point is plotted with its corresponding cluster label, and the cluster centers are indicated by red stars. Set Labels and Title: It sets the labels for the x-axis and y-axis (xlabel() and ylabel()), and the title of the plot (title()). Show Legend: It adds a legend to the plot using legend() to distinguish between data points and cluster centroids. Display Plot: It displays the plot showing the clusters and cluster centroids. By running this code, you'll obtain a scatter plot visualizing the clusters formed by the K-means algorithm. Each data point is assigned to one of the two clusters, and the cluster centroids are indicated by red stars. This visualization helps in understanding the distribution of data points and the separation of clusters based on the selected features.

Now we can see 2 different cluster

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
# df = pd.read_csv("mobile_phone_data.csv")

# Assuming 'price_range' column contains the true labels
X = df.drop(columns=['price_range'])  # Features
```

```
y = df['price_range']  # True labels

# Split the dataset into training and testing sets
X_train, X_test, true_labels_train, true_labels_test =
train_test_split(X, y, test_size=0.2, random_state=42)

# true_labels_train contains the true labels for the training data
X_train

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Handle missing values
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)

# Now, proceed with training your classifier and evaluating its
performance


# Check for NaN values in the original dataset
print("NaN values in X_train:", X_train.isnull().sum().sum())
print("NaN values in X_test:", X_test.isnull().sum().sum())

# Check for NaN values after imputation
print("NaN values after imputation in X_train:",
pd.DataFrame(X_train_imputed).isnull().sum().sum())
print("NaN values after imputation in X_test:",
pd.DataFrame(X_test_imputed).isnull().sum().sum())


import numpy as np

# Check for infinite values in the original dataset
print("Infinite values in X_train:", np.isinf(X_train).sum().sum())
print("Infinite values in X_test:", np.isinf(X_test).sum().sum())
```

```
NaN values in X_train: 4
NaN values in X_test: 1
NaN values after imputation in X_train: 0
NaN values after imputation in X_test: 0
Infinite values in X_train: 0
Infinite values in X_test: 0
```

```python
# Check the data types of features in X_train and X_test
print("Data types of features in X_train:")
print(X_train.dtypes)
print("\nData types of features in X_test:")
print(X_test.dtypes)

# Check summary statistics of features in X_train and X_test
print("\nSummary statistics of features in X_train:")
print(X_train.describe())
print("\nSummary statistics of features in X_test:")
print(X_test.describe())
```

```
Data types of features in X_train:
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory       float64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width         float64
ram              float64
sc_h               int64
sc_w               int64
talk_time          int64
three_g          float64
touch_screen       int64
wifi               int64
dtype: object

Data types of features in X_test:
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory       float64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width         float64
ram              float64
```

```
sc_h                 int64
sc_w                 int64
talk_time            int64
three_g            float64
touch_screen         int64
wifi                 int64
dtype: object

Summary statistics of features in X_train:
       battery_power         blue  clock_speed     dual_sim
fc  \
count    1600.000000  1600.000000  1600.000000  1600.000000
1600.000000
mean     1240.808750     0.490625     1.513625     0.515000
4.310000
std       440.727396     0.500068     0.820189     0.499931
4.339288
min       501.000000     0.000000     0.500000     0.000000
0.000000
25%       852.000000     0.000000     0.675000     0.000000
1.000000
50%      1231.000000     0.000000     1.500000     1.000000
3.000000
75%      1619.000000     1.000000     2.225000     1.000000
7.000000
max      1998.000000     1.000000     3.000000     1.000000
19.000000

           four_g   int_memory        m_dep    mobile_wt      n_cores
\
count  1600.00000  1600.000000  1599.000000  1600.000000  1600.000000

mean      0.52250    32.270000     0.502376   140.633750     4.542500

std       0.49965    18.195165     0.286875    35.338171     2.289972

min       0.00000     2.000000     0.100000    80.000000     1.000000

25%       0.00000    16.000000     0.200000   109.000000     3.000000

50%       1.00000    32.500000     0.500000   141.000000     4.000000

75%       1.00000    48.000000     0.800000   171.000000     7.000000

max       1.00000    64.000000     1.000000   200.000000     8.000000


                pc    px_height     px_width          ram         sc_h
\
count  1600.000000  1600.000000  1599.000000  1599.000000  1600.000000
```

```
mean      9.878125   644.226250  1249.154472  2116.133208    12.220000
std       6.014847   445.436918   431.657906  1081.049416     4.205372
min       0.000000     0.000000   500.000000   258.000000     5.000000
25%       5.000000   280.000000   874.000000  1212.500000     9.000000
50%      10.000000   554.500000  1242.000000  2110.000000    12.000000
75%      15.000000   945.500000  1626.500000  3046.000000    16.000000
max      20.000000  1960.000000  1998.000000  3998.000000    19.000000

              sc_w    talk_time     three_g  touch_screen
wifi
count  1600.000000  1600.000000  1599.000000   1600.000000
1600.000000
mean      5.705625    10.956875     0.762977      0.505000
0.498750
std       4.338863     5.507742     0.425390      0.500131
0.500155
min       0.000000     2.000000     0.000000      0.000000
0.000000
25%       2.000000     6.000000     1.000000      0.000000
0.000000
50%       5.000000    11.000000     1.000000      1.000000
0.000000
75%       9.000000    16.000000     1.000000      1.000000
1.000000
max      18.000000    20.000000     1.000000      1.000000
1.000000

Summary statistics of features in X_test:
       battery_power       blue  clock_speed   dual_sim          fc  \
count     400.000000  400.00000   400.000000  400.00000  400.000000
mean     1229.357500    0.51250     1.556750    0.48750    4.307500
std       434.568002    0.50047     0.799125    0.50047    4.355499
min       502.000000    0.00000     0.500000    0.00000    0.000000
25%       849.500000    0.00000     0.800000    0.00000    1.000000
50%      1214.000000    1.00000     1.600000    0.00000    3.000000
75%      1602.250000    1.00000     2.200000    1.00000    7.000000
max      1993.000000    1.00000     3.000000    1.00000   18.000000

           four_g  int_memory       m_dep   mobile_wt     n_cores
pc  \
count  400.000000  399.000000  400.000000  400.000000  400.000000
400.00000
```

```
mean       0.517500   31.092732    0.498500   138.710000    4.432500
10.07000
std        0.500319   17.923874    0.294814    35.647409    2.280009
6.26364
min        0.000000    2.000000    0.100000    80.000000    1.000000
0.00000
25%        0.000000   16.000000    0.200000   107.750000    2.000000
4.00000
50%        1.000000   27.000000    0.500000   138.000000    4.000000
10.00000
75%        1.000000   46.000000    0.800000   167.250000    6.000000
15.25000
max        1.000000   64.000000    1.000000   200.000000    8.000000
20.00000

        px_height     px_width          ram         sc_h        sc_w  \
count   400.00000   400.000000   400.000000   400.000000   400.00000
mean    648.63500  1261.210000  2156.540000    12.652500     6.01250
std     437.62744   435.273925  1101.442248     4.232201     4.42281
min       4.00000   500.000000   256.000000     5.000000     0.00000
25%     285.50000   887.750000  1161.750000     9.000000     2.00000
50%     583.00000  1273.000000  2236.500000    13.000000     5.00000
75%     950.00000  1653.000000  3142.250000    16.000000     9.00000
max    1874.00000  1997.000000  3993.000000    19.000000    18.00000

        talk_time     three_g  touch_screen         wifi
count  400.000000  400.000000    400.000000   400.000000
mean    11.227500    0.755000      0.495000     0.540000
std      5.286361    0.430626      0.500601     0.499022
min      2.000000    0.000000      0.000000     0.000000
25%      7.000000    1.000000      0.000000     0.000000
50%     11.000000    1.000000      0.000000     1.000000
75%     15.250000    1.000000      1.000000     1.000000
max     20.000000    1.000000      1.000000     1.000000
```

```python
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of true_labels_train:", true_labels_train.shape)
print("Shape of true_labels_test:", true_labels_test.shape)
```

```
Shape of X_train: (1600, 20)
Shape of X_test: (400, 20)
Shape of true_labels_train: (1600,)
Shape of true_labels_test: (400,)
```

The workshop task this week involves unsupervised learning - an exercise in clustering. We'll use a the Mobile Price Data dataset to walk through the process of kmeans and hierarchical clustering. We'll then introduce a text dataset for you to experiment with text analysis.