Industrial System Modeling

April 10, 2019

Industrial System Modeling
[Data Science]

0.1 Pesonal Branding

- **Name**: Abdullah Al Imran
- **Currently**: Data Scientist at DATFU Ltd. *[1 year]*
- **Previously**:
  - Research Assistant at Data & Design Lab, Dept. of CS, University of Dhaka. *[1.5 year]*
  - Data Scienist at Revvey Co, a New York based start-up. *[4 months]*
- **Education**: BSc. in Computer Science & Engineering, American Internaional University-Bangladesh
- **Connect** with me on LinkedIn

0.2 Table of contents

Data Science: Birds Eye View

What is Data Science (with a bit of history)? Data science is a **multi-disciplinary** field that uses scientific methods, processes, algorithms and systems to **extract knowledge** and insights from structured and unstructured **data**.

- The term "data science" has appeared in various contexts over the past thirty years but did not become an established term until recently.
- In an early usage, it was used as a substitute for computer science by Peter Naur in 1960.
- In 1974, Naur introduced the term "datalogy" in his publication named Concise Survey of Computer Methods, which freely used the term "data science" in a wide range of applications.
- In 1996, members of the International Federation of Classification Societies (IFCS) met in Kobe for their biennial conference. Here, for the first time, the term "data science" is included in the title of the conference ("Data Science, classification, and related methods").
- In April 2002, the International Council for Science (ICSU): Committee on Data for Science and Technology (CODATA) started the Data Science Journal.
- Around 2007, Turing award winner Jim Gray envisioned "data-driven science" as a "fourth paradigm" of science that uses the computational analysis of large data as primary scientific method.
- In the 2012 Harvard Business Review article "Data Scientist: The Sexiest Job of the 21st Century"!!

The Data Science Venn Diagram

What skills do you need to be a Data Scientist?

- **Mathematics**: Linear Algebra, Calculus (Univariate, Multivariate), Numerical Optimization
- **Statistics**: Descriptive Stats, Inferential Stats, Probability Theory, Bayesian
- **Programming**: Python or R or Julia
- **Machine Learning**: Regression, Classification, Clustering, NLP
- **Database**: SQL, NoSQL
- **Domain Expertise**
- Obviously **MOTIVATION** and **GRIT**!!

Machine Learning

What do you think about Machine Learning? Is it as following?

Machine learning is a **branch** of artificial intelligence (AI) that provides systems the ability to **automatically** learn and **improve from experience** without being **explicitly** programmed.

Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

0.2.1 Artificial Intelligence vs Machine Learning vs Deep Learning vs Data Mining

Artificial Intelligence: Artificial Intelligence (AI) is the **general** field of computer science that has to do with creating programs that **simulate intelligence**.

Machine Learning: Machine Learning (ML) is a **subfield** of AI that focuses on making programs that can learn **without being explicitly programmed**.

Deep Learning: Deep Learning focuses on a **specific** ML algorithm, **Neural Networks (NN)**, and more specifically Deep NNs(NNs with many layers).

Data mining: Data mining is a field of computer science that **discovers patterns** through "mining" in data, and come up with **insights**. Here, the data is mostly raw/unstructured data. Data mining uses techniques and algorithms from machine learning, statistics and database theory to mine large databases and come up with patterns.

0.2.2 Top Industrial Applications of Machine Learning

Manufacturing & Production: * Predictive maintenance or condition monitoring * Warranty reserve estimation * Propensity to buy * Demand forecasting * Process optimization * Telematics

Retail: * Predictive inventory planning * Recommendation engines * Upsetl and cross-channel marketing * Market segmentation and targeting * Customer Roland kfetime value

Travel and Hospitality: * Aircraft scheduling * Dynamic pricing * Social media - consumer feedback and interaction analysis * Customer complaint resolution * Traffic patterns and congestion management

Financial Services: * Risk analytics and regulation * Customer Segmentation * Cross-selling and up-selling * Sales and marketing campaign management * Credit worthiness evaluation

Healthcare and Life Sciences: * Alerts and diagnostics from real-time patient data * Disease Identification and risk stratification * Patient triage optimization * Proactive health management * Healthcare provider sentiment analysis

Enernergy, Feedstock, and Utilities * Power usage analytics * Seismic data processing * Carbon emissions and trading * Customer-specific pricing * Smart grid management * Energy demand and supply optimization

0.2.3 Branches of Machine Learning

1. Supervised Learning: You have a **target, a value or a class to predict**. For instance, let's say you want to predict the revenue of a store from different inputs (day of the week, advertising, promotion). Then your model will be trained on historical data and use them to forecast future revenues. Hence the model is **supervised**, it knows what to learn. Supervised learning includes regression and classification problems.

2. Unsupervised Learning: You have **unlabelled data and looks for patterns, groups** in these data. For example, you want to cluster to clients according to the type of products they order, how often they purchase your product, their last visit, ... Instead of doing it manually, **unsupervised** machine learning will automatically discriminate different clients. Unsupervised learning includes clustering, anomaly detection etc.

3. Reinforcement Learning: You want to **attain an objective**. For example, you want to find the best strategy to win a game with specified rules. Once these rules are specified, **reinforcement** learning techniques will play this game many times to find the best strategy. Some examples of reinforcement learning are - Self Driving Cars, Google Alpha Go etc.

The Data Science Pipeline

Source: Datacamp

1. Collection

- Collect historical/relevent data for your problem from reliable sources. Sometime may be from heterogeneous (multiple) sources.

2. Exploration

- Perform exploratory statistical analysis (EDA) to understand every bit of your data.
- Use different visualization techniques and descriptive statistics.

3. Munging

- It is mostly known as *Data Preprocessing or Data Cleaning*.
- This is the most tedious and important part of any data science project.
- You need to perform -
  - Data cleaning,
  - Missing value handling,
  - Feature encoding,
  - Normalization/Scaling/Standardization
  - Feature selection,
  - Feature engineering, and
  - Data segregation
- Efforts put in this phase will highly affect your ultimate results.

4. Modeling

- This is the Machine Learning part! YESSS!!
- Choose the right algorithm for your data.
- Find the best hyperparameters for your algorithm by tuning on the data.
- Be careful!! DON'T **OVERFIT** YOUR MODEL!

5. Validation

- Mostly called *Testing*.
- Choose the correct metric for the validation of your model.
- Analyse the training and validation performance and see if the model is overfitting or not. Is the result close/consistent?

6. Reporting

- Interpret your model.
- Discover the hidden patterns.
- Make documentation.
- Deploy your model to production.

An interesting stats on the time expenditure of a Data Scientist on different sections of the data science pipeline.

**We will try to follow this pipeline in the upcoming case studies.

Case Study 1: Regression

Problem Description:

A reputed mobile company is launching it's new smartphone. The company is confused how they should set the price for the new smartphone. They need to ensure that the price is properly optimized as well as attractive to the consumers. They have historical data of there previous smartphone with market prices. Now, the dataset has been given to you which contains several important features of the previously released phones and their market prices. Using the parameters of the historical data you need to build a model that can predict an otimized price for the upcoming new smartphone.

Let's CRACK it!!

0.2.4 Step 1: Data Collection

```
In [1]: import pandas as pd

        data_rgr = pd.read_csv("data/regression_mobile_price.csv")
```

0.2.5 Step 2: Exploration (Exploratory Data Analysis)

Let's have glance of the data.

```
In [2]: data_rgr.head()
```

```
Out[2]:    weight_g  weight_oz     SIM display_type  display_resolution  \
        0     260.0       9.17    Dual          IPS                 7.0
        1     169.0       5.96    Dual          IPS                 5.5
        2     166.0       5.86  Single          IPS                 5.5
        3     125.0       4.41    Dual          IPS                 5.0
        4     353.8      12.49  Single          IPS                 8.0

           display_size_ppi           OS        CPU  memory_card  internal_memory_GB  \
        0               210  Marshmallow  Quad-core          128                  16
        1               401  Marshmallow  Octa-core          256                  32
        2               267     Lollipop  Octa-core           32                  32
        3               294  Marshmallow  Quad-core           32                   8
        4               283     Lollipop  Quad-core          256                  32

           RAM_GB  primary_camera  secondary_camera  battery  approx_price_EUR
        0     2.0            13.0               2.0     3400               170
        1     3.0            13.0               5.0     4080               250
        2     3.0            13.0              13.0     4020               230
        3     1.0             8.0               5.0     2000               110
        4     2.0             5.0               2.0     4420               350
```

What about the data shape and datatypes?

```
In [3]: print("""
        No of columns: {}
        No of rows: {}
        No of categorical columns: {}
        No of numerical columns: {}""".format(data_rgr.shape[1],
                                              data_rgr.shape[0],
                                              len(data_rgr.select_dtypes('O').columns),
                                              len(data_rgr.select_dtypes(['int', 'float']).colu
```

```
No of columns: 15
No of rows: 894
No of categorical columns: 4
No of numerical columns: 11
```

Let's have a descriptive statistical tables of the numeric columns.

```
In [4]: num_cols = data_rgr.select_dtypes(['int', 'float']).columns
        data_rgr[num_cols].describe().transpose()
```

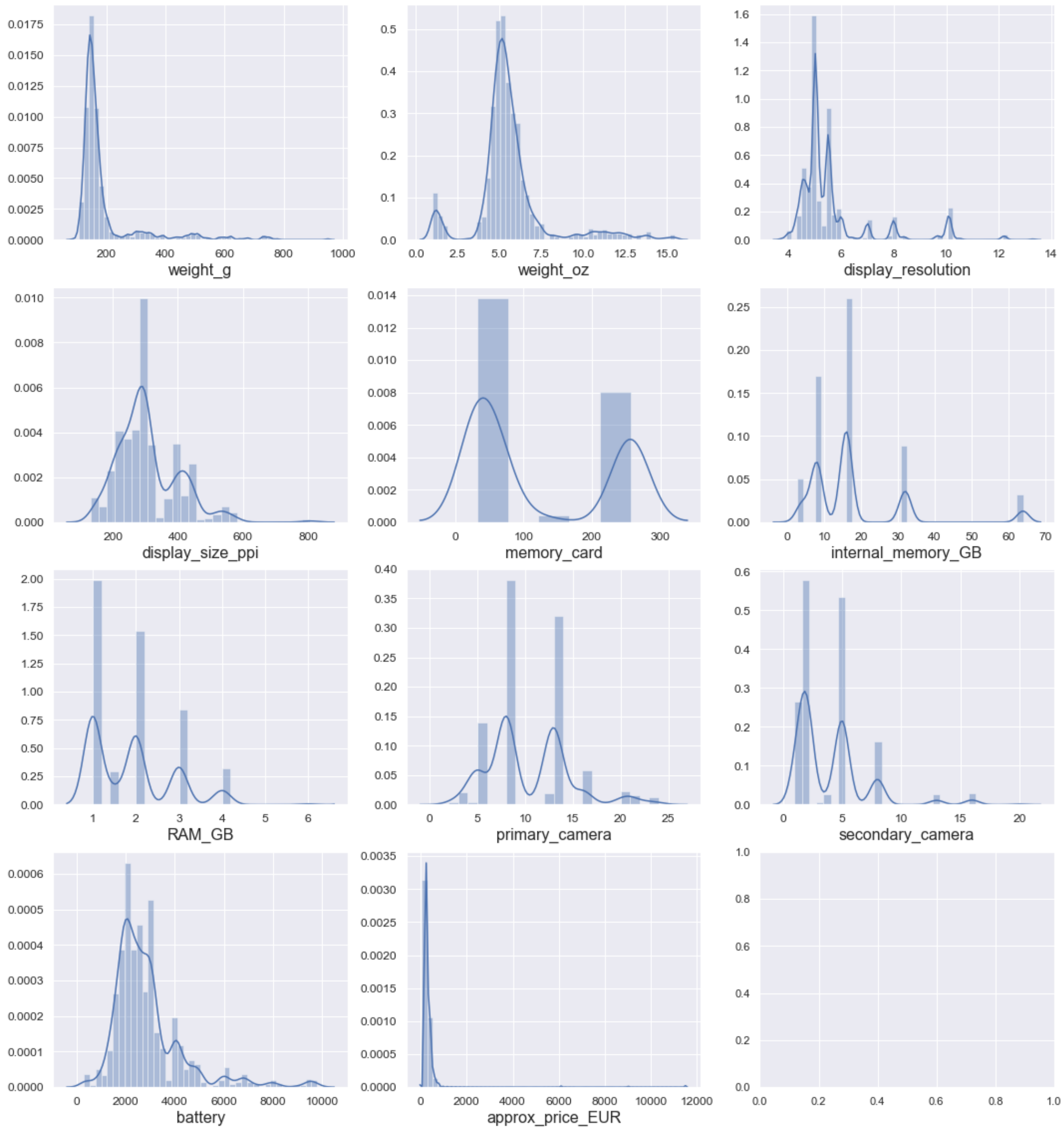
```
Out[4]:                     count         mean          std     min      25%      50%  \
        weight_g            894.0   191.867673   114.934276   88.20   140.00   154.00
        weight_oz           894.0     5.607416     2.161909    0.99     4.76     5.29
        display_resolution  894.0     5.603143     1.480392    3.70     5.00     5.00
        display_size_ppi    894.0   305.961969    93.823252  132.00   245.00   294.00
        memory_card         894.0   121.270694   102.535103   32.00    32.00    64.00
        internal_memory_GB  894.0    17.560403    13.680903    1.00     8.00    16.00
        RAM_GB              894.0     1.883110     0.936407    1.00     1.00     2.00
        primary_camera      894.0    10.308110     4.267153    2.00     8.00     8.00
        secondary_camera    894.0     3.977740     3.009084    1.00     2.00     2.10
        battery             894.0  2881.751678  1454.876941  300.00  2000.00  2540.00
        approx_price_EUR    894.0   274.439597   529.957657   60.00   150.00   220.00

                               75%       max
        weight_g             176.9    948.00
        weight_oz              6.0     15.49
        display_resolution     5.5     13.30
        display_size_ppi     367.0    807.00
        memory_card          256.0    256.00
        internal_memory_GB    16.0     64.00
        RAM_GB                 2.0      6.00
        primary_camera        13.0     24.00
        secondary_camera       5.0     20.00
        battery             3150.0   9800.00
        approx_price_EUR     300.0  11500.00
```

How's the distribution of numerical columns?

```
In [5]: import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        sns.set()
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')

        def plot_distribution(data, features):
```

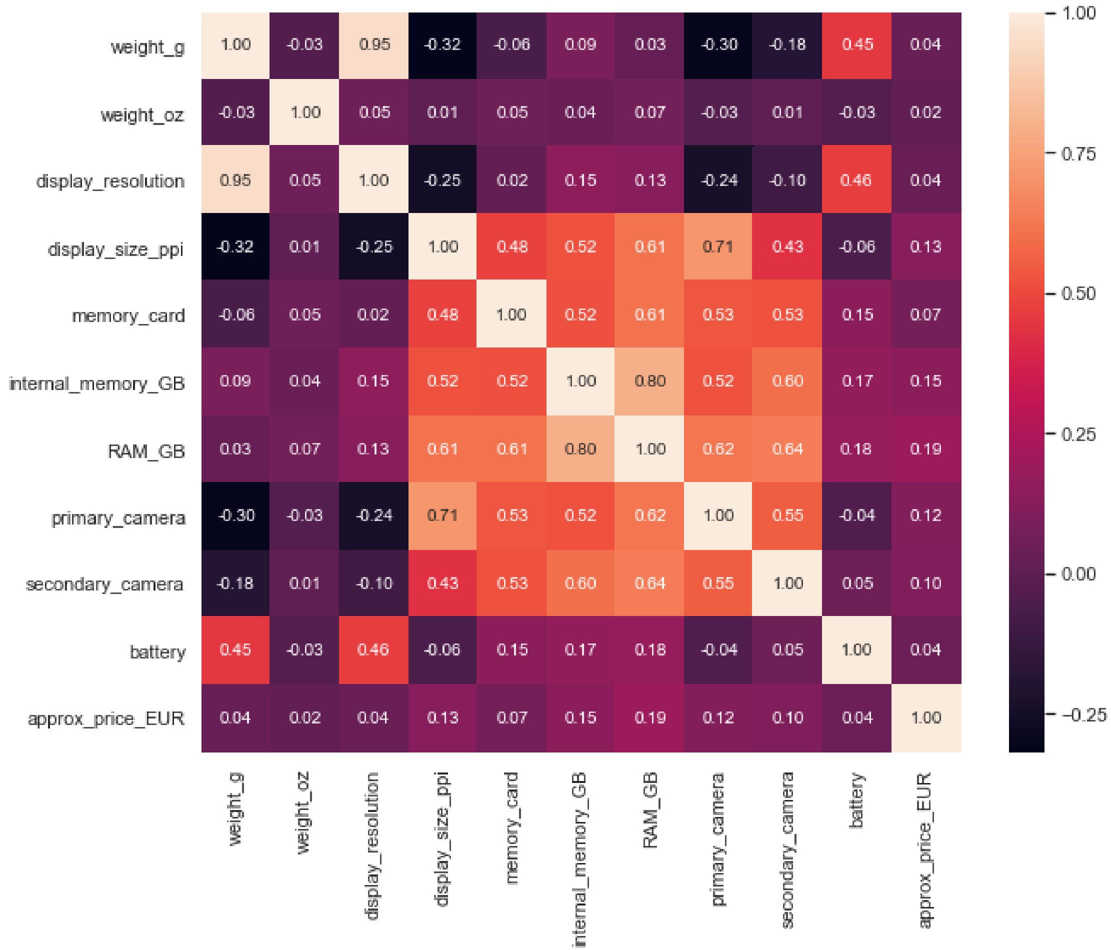
```
    i = 0
    plt.figure()
    col = 3
    row = int(np.ceil(len(features)/col))
    fig, ax = plt.subplots(row,col,figsize=(18,20))

    for feature in features:
        i += 1
        plt.subplot(row,col,i);
        sns.distplot(tuple(data[feature]))
        plt.xlabel(feature, fontsize=16)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', which='major', labelsize=12)
        plt.tick_params(axis='y', which='major', labelsize=12)
    plt.show();

plot_distribution(data_rgr, num_cols)
```

```
<matplotlib.figure.Figure at 0x7f1cedba7518>
```



Is there any correlation between the numeric columns?

```
In [6]: corr = data_rgr[num_cols].corr()
        fig, ax = plt.subplots(figsize=(10,8))
        sns.heatmap(corr,
                    xticklabels=corr.columns.values,
                    yticklabels=corr.columns.values,
                    annot=True, fmt=".2f");
```



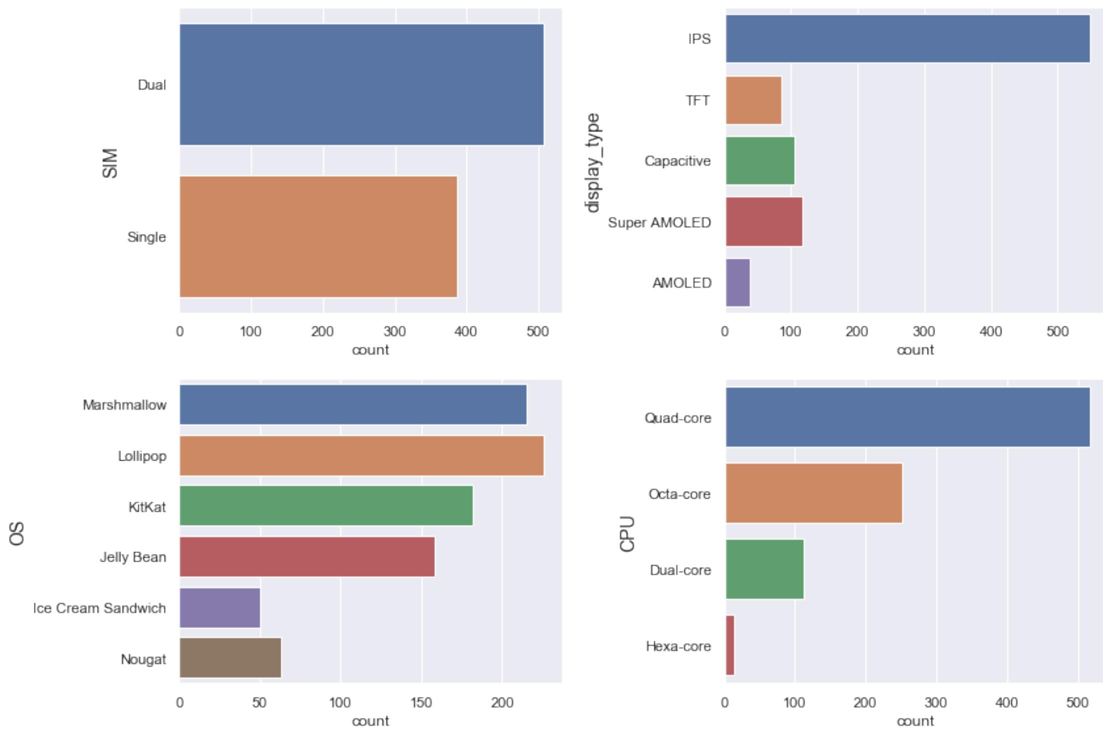
What about the categorical columns?

```
In [7]: cat_cols = data_rgr.select_dtypes('O').columns
```

```
In [8]: def count_plot(data, features):
            i = 0
            plt.figure()
            col = 2
            row = int(np.ceil(len(features)/col))
            fig, ax = plt.subplots(row,col,figsize=(12,8))

            for feature in features:
                i += 1
                plt.subplot(row,col,i);
                ax = sns.countplot(data=data_rgr,y=feature)
                ax.set_yticklabels(ax.get_yticklabels())
                plt.tight_layout()
                plt.ylabel(feature, fontsize=14)
```

```
        plt.show();

    count_plot(data_rgr, cat_cols)
```

```
<matplotlib.figure.Figure at 0x7f1cea607f98>
```



0.2.6 Step 3: Munging (Data Preprocessing)

Are there any **missing values**?

In [9]:
```
pd.DataFrame(data_rgr.isnull().sum(), columns=['Missing Values'])
```

Out[9]:

	Missing Values
weight_g	0
weight_oz	0
SIM	0
display_type	0
display_resolution	0
display_size_ppi	0
OS	0
CPU	0
memory_card	0
internal_memory_GB	0

```
RAM_GB                        0
primary_camera                0
secondary_camera              0
battery                       0
approx_price_EUR              0
```

Now, we have to encode the **categorical features**.

Note: * If you use any linear/distance/neural based model, use **OneHotEncoding**. * If you use tree based model, you can use **LabelEncoder**.

```
In [10]: data_rgr_encoded = pd.get_dummies(data_rgr, columns=cat_cols, prefix=cat_cols)

         print("Now our dataset has {} columns.".format(data_rgr_encoded.shape[1]))
```

```
Now our dataset has 28 columns.
```

```
In [11]: data_rgr_encoded.head()
```

```
Out[11]:    weight_g  weight_oz  display_resolution  display_size_ppi  memory_card  \
         0     260.0       9.17                 7.0               210          128
         1     169.0       5.96                 5.5               401          256
         2     166.0       5.86                 5.5               267           32
         3     125.0       4.41                 5.0               294           32
         4     353.8      12.49                 8.0               283          256

            internal_memory_GB  RAM_GB  primary_camera  secondary_camera  battery  \
         0                  16     2.0            13.0               2.0     3400
         1                  32     3.0            13.0               5.0     4080
         2                  32     3.0            13.0              13.0     4020
         3                   8     1.0             8.0               5.0     2000
         4                  32     2.0             5.0               2.0     4420

                ...         OS_Ice Cream Sandwich  OS_Jelly Bean  OS_KitKat  \
         0      ...                             0              0          0
         1      ...                             0              0          0
         2      ...                             0              0          0
         3      ...                             0              0          0
         4      ...                             0              0          0

            OS_Lollipop  OS_Marshmallow  OS_Nougat  CPU_Dual-core  CPU_Hexa-core  \
         0            0               1          0              0              0
         1            0               1          0              0              0
         2            1               0          0              0              0
         3            0               1          0              0              0
         4            1               0          0              0              0

            CPU_Octa-core  CPU_Quad-core
         0              0              1
```

```
1                1                0
2                1                0
3                0                1
4                0                1

[5 rows x 28 columns]
```

Now, it's time for **data segregation**. We need to separate the feature columns and target colum.

```
In [12]: X = data_rgr_encoded.loc[:, data_rgr_encoded.columns!='approx_price_EUR']
         y = data_rgr_encoded.loc[:, data_rgr_encoded.columns=='approx_price_EUR']
```

We have seen from the distribution plot (in Explore section) that most of the features are not well distributed and have variety of ranges. This form of data can confuse the linear/distance based models. That is why we need **Normalize** our data.

```
In [13]: from sklearn.preprocessing import Normalizer

         scaler = Normalizer()
         scaler.fit(X.values)
         X_standered = scaler.transform(X.values)
         X_scaled = pd.DataFrame(X_standered, columns=X.columns)
```

Now we have to make dataset for **traing and validation**.

```
In [14]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.20, rand

         print("""
         X_train has {} data points.
         y_train has {} data points.
         X_test has {} data points.
         y_test has {} data points.
         """.format(X_train.shape[0], y_train.shape[0], X_test.shape[0], y_test.shape[0]))
```

```
X_train has 715 data points.
y_train has 715 data points.
X_test has 179 data points.
y_test has 179 data points.
```

0.2.7 Step 4: Modeling

WOOOHAAA!! Here it is!

Finally, we've reached the modeling part! The easiest one!

We'll use **Linear Regression** algorithm for data modeling. This is the simplest, easiest and most utilized machine learning algorithms in the world!

```
In [15]: from sklearn.linear_model import LinearRegression

         lr_model = LinearRegression()
         lr_model.fit(X_train, y_train)
```

```
Out[15]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)
```

0.2.8 Step 5: Validation

```
In [16]: y_pred = lr_model.predict(X_test)
```

There are wide variety of evaluation metrics for validating regression models such as **Mean absolute error, Mean squared error, Mean squared logarithmic error, Median absolute error regression, R^2 score**. For our problem we'll use the R^2 metric. The close the R^2 score to 1, the better the model is.

```
In [17]: from sklearn.metrics import r2_score

         r2 = -r2_score(y_test, y_pred)

         print("The R^2 score is {}".format(r2))
```

```
The R^2 score is 0.8323921565731989
```

0.2.9 Step 6: Reporting

```
In [18]: print('Intercept of the Regression model:', -lr_model.intercept_[0])
         print('\nSlope/Coefficients of the Regression model:')
         slope = pd.DataFrame(lr_model.coef_.T, X_train.columns, columns=['Coefficient']).sort_
         slope
```

```
Intercept of the Regression model: 256.9621281595987

Slope/Coefficients of the Regression model:
```

Out[18]:

	Coefficient
RAM_GB	314146.417139
OS_Marshmallow	219744.325332
weight_oz	16843.314045
secondary_camera	13739.436259
primary_camera	9828.214847
weight_g	3385.033277
display_size_ppi	1054.604654
battery	543.797803
memory_card	-1902.315258
internal_memory_GB	-2771.667619

```
display_type_Super AMOLED    -3182.540357
display_type_Capacitive      -5296.577271
display_type_AMOLED         -17138.063191
CPU_Dual-core               -30655.327323
CPU_Quad-core               -33427.034232
OS_KitKat                   -35324.206293
OS_Nougat                   -47524.034154
OS_Jelly Bean               -60289.070235
display_type_TFT            -75262.998079
CPU_Octa-core               -81528.680484
CPU_Hexa-core               -83748.328118
SIM_Single                 -109383.746293
OS_Lollipop                -119568.049308
SIM_Dual                   -119975.623864
display_type_IPS           -128479.191260
OS_Ice Cream Sandwich      -186398.335500
display_resolution         -220558.855762
```

Case Study 2: Classification
Problem Description:
Have you ever heared about the job "**Wine Testing**"?

While a degree is not required to become a wine taster, it is difficult to land a top job without having some training in wine. Wine tasters work in wineries, bars, for magazines and even in hotels and restaurants. According to an article by Kathleen Green on the Bureau of Labor Statistics website, it is estimated that a wine tester (master sommelier) can earn as much as **$160,000 a year**. Simply Hired estimates that less-experienced wine tasters make an average of **$71,000** a year as of 2012.

The wine production companies don't want to pay such a big ammount to the wine testers anymore. They have a good collection of their previous wine quality data with ratings in a range of $0-2$. Now, they want you to build a model that can accurately predict ratings for a newly produced wine using the previous parameters.

Note: Here, the target is to predict a rating. Rating has a range of $0-2$. It is an ordinal categorical variable. Here, **0=Bad Quality, 1=Good Quality, and 2=Best Quality**. So, we can approch the problem as a classification problem.

Let your model HACK the job of wine testers!!

0.2.10 Step 1: Data Collection

```
In [19]: import pandas as pd

         data_clsf = pd.read_csv("data/classification_wine_quality.csv")
```

0.2.11 Step 2: Exploration (Exploratory Data Analysis)

Let's have a glance of the data.

```
In [20]: data_clsf.head()
```

Out[20]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        1
1      9.8        1
2      9.8        1
3      9.8        1
4      9.4        1
```

What about the data shape and datatypes?

In [21]:
```
print("""
No of columns: {}
No of rows: {}
No of categorical columns: {}
No of numerical columns: {}""".format(data_clsf.shape[1],
                                      data_clsf.shape[0],
                                      len(data_clsf.select_dtypes('O').columns),
                                      len(data_clsf.select_dtypes(['int', 'float']).co
```

```
No of columns: 12
No of rows: 1599
No of categorical columns: 0
No of numerical columns: 12
```

Here, we don't have any categorical columns. That's easy!!
Let's have a descriptive statistical tables of the numeric columns.

In [22]:
```
data_clsf.describe().transpose()
```

Out[22]:
```
                   count      mean       std      min     25%  \
fixed acidity     1599.0  8.319637  1.741096  4.60000  7.1000
volatile acidity  1599.0  0.527821  0.179060  0.12000  0.3900
citric acid       1599.0  0.270976  0.194801  0.00000  0.0900
residual sugar    1599.0  2.538806  1.409928  0.90000  1.9000
```

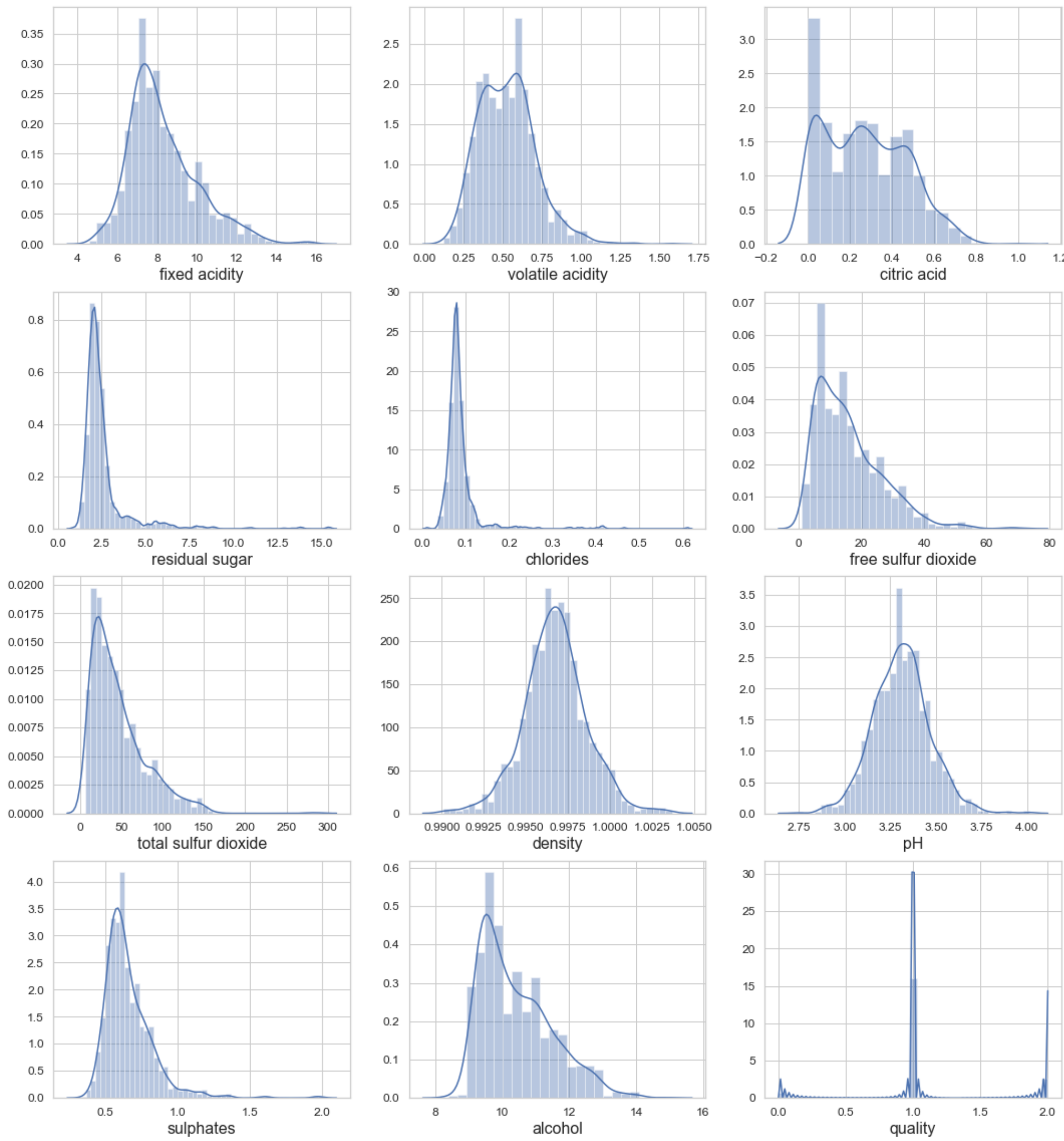
```
chlorides             1599.0   0.087467   0.047065  0.01200   0.0700
free sulfur dioxide   1599.0  15.874922  10.460157  1.00000   7.0000
total sulfur dioxide  1599.0  46.467792  32.895324  6.00000  22.0000
density               1599.0   0.996747   0.001887  0.99007   0.9956
pH                    1599.0   3.311113   0.154386  2.74000   3.2100
sulphates             1599.0   0.658149   0.169507  0.33000   0.5500
alcohol               1599.0  10.422983   1.065668  8.40000   9.5000
quality               1599.0   1.096310   0.407354  0.00000   1.0000

                           50%         75%        max
fixed acidity          7.90000    9.200000   15.90000
volatile acidity       0.52000    0.640000    1.58000
citric acid            0.26000    0.420000    1.00000
residual sugar         2.20000    2.600000   15.50000
chlorides              0.07900    0.090000    0.61100
free sulfur dioxide   14.00000   21.000000   72.00000
total sulfur dioxide  38.00000   62.000000  289.00000
density                0.99675    0.997835    1.00369
pH                     3.31000    3.400000    4.01000
sulphates              0.62000    0.730000    2.00000
alcohol               10.20000   11.100000   14.90000
quality                1.00000    1.000000    2.00000
```

How's the distribution of numerical columns?

In [23]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
import numpy as np
import warnings
warnings.filterwarnings('ignore')

def plot_distribution(data, features):
    i = 0
    plt.figure()
    col = 3
    row = int(np.ceil(len(features)/col))
    fig, ax = plt.subplots(row,col,figsize=(18,20))
    sns.set_style("whitegrid")

    for feature in features:
        i += 1
        plt.subplot(row,col,i);
        sns.distplot(data[feature])
        plt.xlabel(feature, fontsize=16)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', which='major', labelsize=12)
```

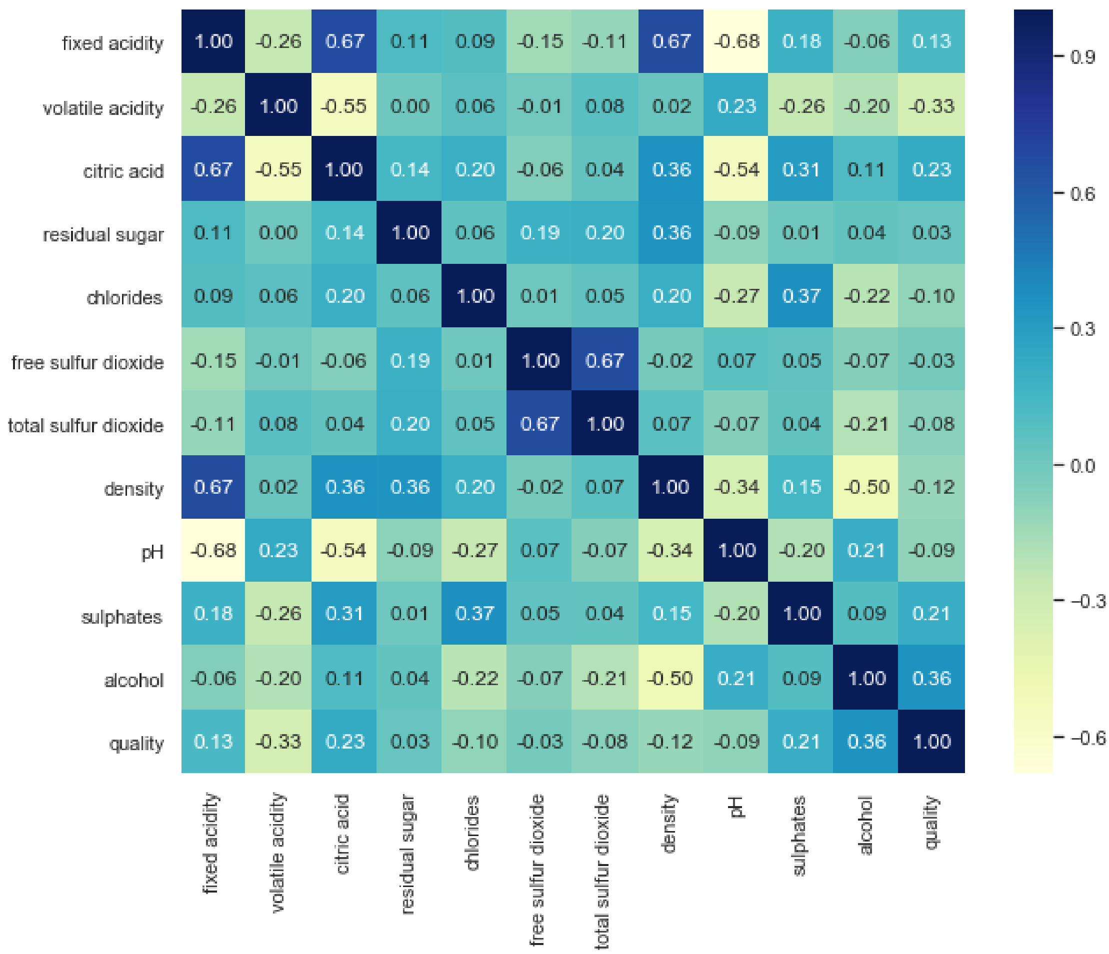
```
            plt.tick_params(axis='y', which='major', labelsize=12)
        plt.show();

    plot_distribution(data_clsf, data_clsf.columns)
```

```
<matplotlib.figure.Figure at 0x7f1cedc052b0>
```



OOHH MY GOOOOOSSHHH!!
Very well distributed data!
Let's quickly check the correlations between the columns?

```
In [24]: corr = data_clsf.corr()
         fig, ax = plt.subplots(figsize=(10,8))
         sns.heatmap(corr,
                     xticklabels=corr.columns.values,
                     yticklabels=corr.columns.values,
                     annot=True, fmt=".2f", cmap="YlGnBu");
```



0.2.12 Step 3: Munging (Data Preprocessing)

Are there any **missing values**?

```
In [25]: pd.DataFrame(data_clsf.isnull().sum(), columns=['Missing Values'])

Out[25]:                       Missing Values
         fixed acidity                     0
         volatile acidity                  0
         citric acid                       0
         residual sugar                    0
```

```
chlorides                  0
free sulfur dioxide        0
total sulfur dioxide       0
density                    0
pH                         0
sulphates                  0
alcohol                    0
quality                    0
```

DAMN!! We are very lucky!! Still we've got no missing values!

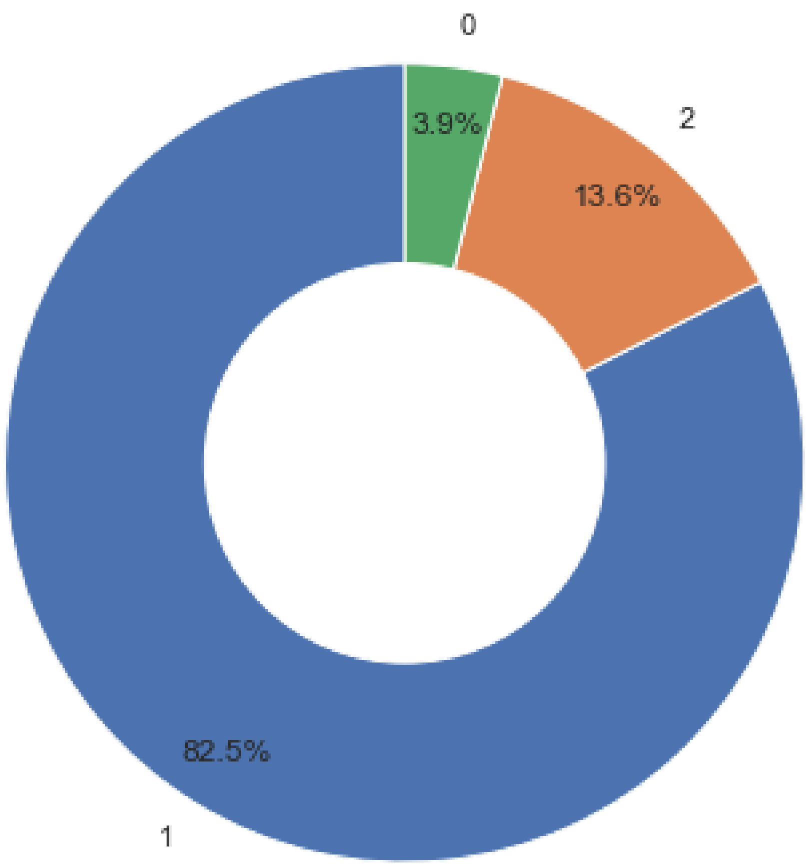
What percentage belongs to what ratings? Let's understand the rating distribution in our data. It is very important to check wheather your classification dataset is balanced or not.

In [26]:
```
val_counts = data_clsf['quality'].value_counts()
sizes = val_counts.values
labels = val_counts.index

class_dist = pd.DataFrame({"rating":labels,
                           "count":sizes,
                           "percentage": np.round((sizes/sum(sizes))*100, 2)})
class_dist
```

Out[26]:

	rating	count	percentage
0	1	1319	82.49
1	2	217	13.57
2	0	63	3.94

In [27]:
```
def plot_class_dist(sizes, labels):
    # explode = [0.05]*len(val_counts.index)
    fig1, ax1 = plt.subplots(figsize=(5,5))
    ax1.pie(sizes,
            labels=labels,
            autopct='%1.1f%%',
            startangle=90,
    #         explode=explode,
            pctdistance=0.85)

    centre_circle = plt.Circle((0,0),0.50, fc='white')
    fig = plt.gcf()
    fig.gca().add_artist(centre_circle)

    ax1.axis('equal')
    plt.tight_layout()
    plt.show();

plot_class_dist(sizes, labels);
```



So, the data is not well distributed. It is called the **imbalanced class** problem. The dominant rating is 1=Good Quality. To get a better accuracy from the model, we might need to balance the dataset using **sampling techniques(oversampling/undersampling)**.

Now, it's time for **data segregation**. We need to separate the feature columns and target column.

```
In [28]: X = data_clsf.loc[:, data_clsf.columns!='quality']
         y = data_clsf.loc[:, data_clsf.columns=='quality']
```

For this problem, we don't need to normalize/standardize the feaures. Because wwe have very well-distributed features. And also we'll train a **tree based model** this time, **which is not affected by feature transformation**.

So, let's jump to making the **traing and validation** dataset.

```
In [29]: from sklearn.model_selection import train_test_split


         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
                                                             stratify=y, random_state=1234)
```

```
print("""
X_train has {} data points.
y_train has {} data points.
X_test has {} data points.
y_test has {} data points.
""".format(X_train.shape[0], y_train.shape[0], X_test.shape[0], y_test.shape[0]))
```

```
X_train has 1279 data points.
y_train has 1279 data points.
X_test has 320 data points.
y_test has 320 data points.
```

0.2.13 Step 4: Modeling

WOOOHAAA!!

Again, we've reached the modeling part! The easiest one!

We'll use **Decision Tree** algorithm for data modeling. This is one of the most powerful yet an easy model. The best thing about Decision Tree is you can extract the **feature importance** from the model and also interpret the tree for **extracting the underlying patterns** in data.

```
In [30]: from sklearn.tree import DecisionTreeClassifier

         clf_model = DecisionTreeClassifier(random_state=1234)
         clf_model.fit(X_train, y_train)
```

```
Out[30]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=1234,
                     splitter='best')
```

0.2.14 Step 5: Validation

```
In [31]: y_pred = clf_model.predict(X_test)
```

There are wide variety of evaluation metrics for validating regression models such as **Accuracy, AUC, Precision, Recall, F1 score**. For our problem we'll use the **accuracy** metric.

Remember: Accuracy is not always the best metric to valiate classification model. Sometime, depending on the problem, you'll need to choose other metrics.

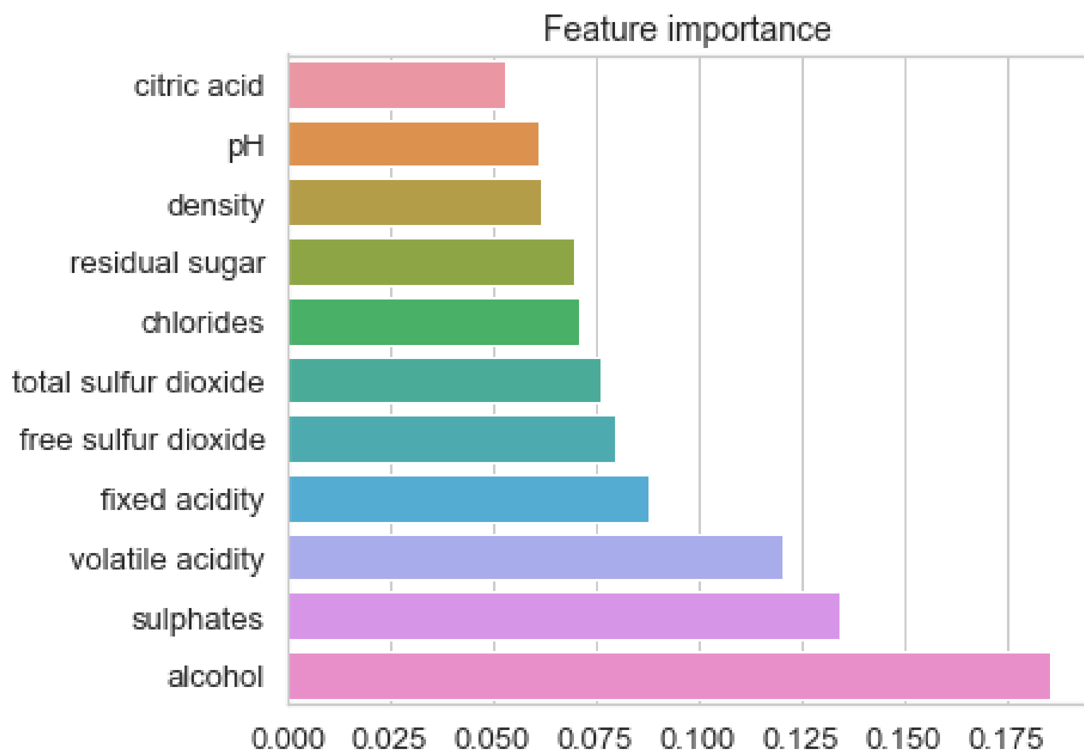
```
In [32]: from sklearn.metrics import accuracy_score

         acc_score = accuracy_score(y_test, y_pred)

         print("The accuracy score is {}".format(acc_score))
```

```
The accuracy score is 0.821875
```

0.2.15 Step 6: Reporting

Get the Feature importance

```
In [33]: coef = pd.Series(clf_model.feature_importances_, index = X.columns)
         fig, ax = plt.subplots(figsize=(6,5))
         imp_coef = coef.sort_values()
         sns.barplot(imp_coef.values, imp_coef.index)
         plt.title("Feature importance", fontsize=14)
         plt.tick_params(axis='x', labelsize=13)
         plt.tick_params(axis='y', labelsize=13)
```



Let's generate a visualization of the tree.

```
In [34]: from sklearn import tree
         import pydotplus

         dot_data = tree.export_graphviz(clf_model,
                                         feature_names=X_train.columns,
                                         out_file=None,
                                         filled=True,
```

```
                                         rounded=True)
        graph = pydotplus.graph_from_dot_data(dot_data)
        graph.write_svg('tree.svg')
```

Out[34]: True

Case Study 3: Clustering

Applications of Clustering Algorithm: * Behavioural Segmentation * Anomaly Detection * Social Network Analysis * Market Segmentation

Problem Description:

In this project, we will analyze a dataset containing data on various customers' annual spending amounts (reported in monetary units) of diverse product categories for internal structure. One goal of this project is to best describe the variation in the different types of customers that a wholesale distributor interacts with. Doing so would equip the distributor with insight into how to best structure their delivery service to meet the needs of each customer.

Let's find the customer segments!!

0.2.16 Step 1: Data Collection

```
In [35]: import pandas as pd

         data_clst = pd.read_csv("data/clustering_wholesale_customers.csv")
```

0.2.17 Step 2: Exploration (Exploratory Data Analysis)

Let's have a glance of the data.

```
In [36]: data_clst.head()
```

Out[36]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	12669	9656	7561	214	2674	1338
1	7057	9810	9568	1762	3293	1776
2	6353	8808	7684	2405	3516	7844
3	13265	1196	4221	6404	507	1788
4	22615	5410	7198	3915	1777	5185

What about the data shape and datatypes?

```
In [37]: print("""
         No of columns: {}
         No of rows: {}
         No of categorical columns: {}
         No of numerical columns: {}""".format(data_clst.shape[1],
                                              data_clst.shape[0],
                                              len(data_clst.select_dtypes('O').columns),
                                              len(data_clst.select_dtypes(['int', 'float']).co
```

```
No of columns: 6
No of rows: 435
No of categorical columns: 0
No of numerical columns: 6
```

Again, we don't have any categorical columns. That's easy!!
Let's have a descriptive statistical tables of the numeric columns.

In [38]:
```
data_clst.describe().transpose()
```

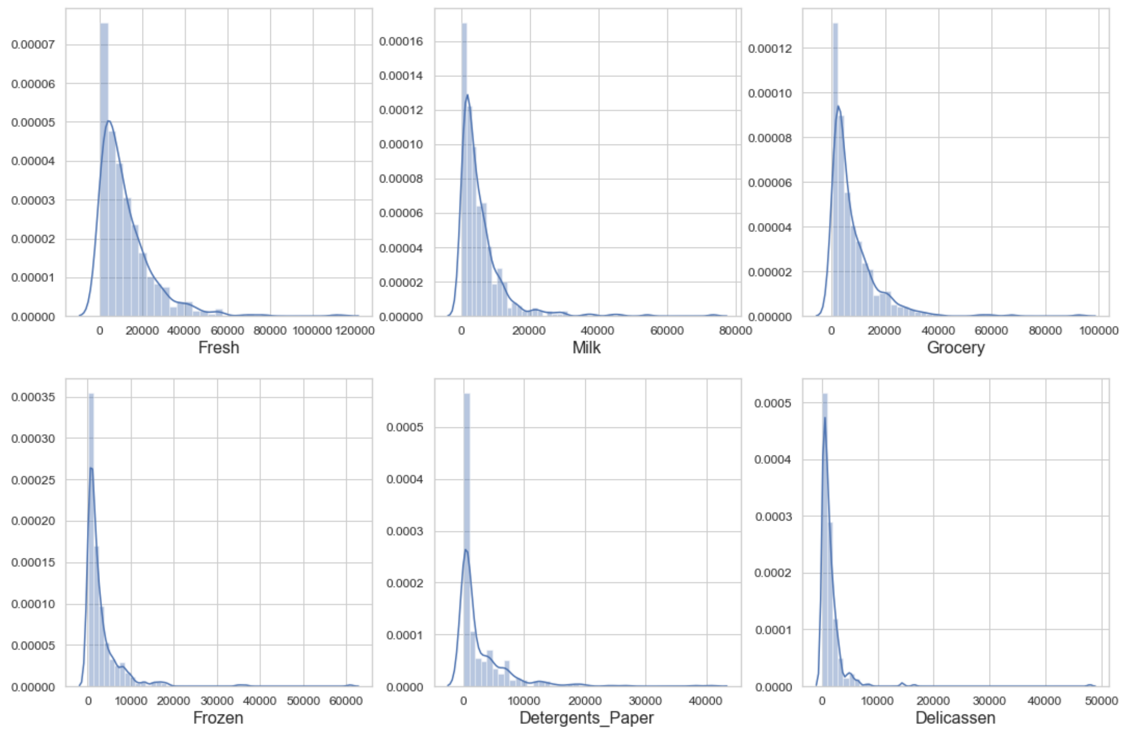
Out[38]:
```
                  count          mean           std    min     25%     50%  \
Fresh             435.0  12089.372414  12662.796341    3.0  3208.0  8565.0
Milk              435.0   5788.103448   7374.172350  112.0  1579.5  3634.0
Grocery           435.0   7911.158621   9365.740973  218.0  2156.0  4757.0
Frozen            435.0   3096.126437   4873.769559   25.0   770.5  1541.0
Detergents_Paper  435.0   2848.473563   4679.364623    3.0   260.0   813.0
Delicassen        435.0   1536.797701   2833.363881    3.0   411.5   967.0

                      75%       max
Fresh             16934.5  112151.0
Milk               7168.0   73498.0
Grocery           10665.5   92780.0
Frozen             3559.5   60869.0
Detergents_Paper   3935.0   40827.0
Delicassen         1825.5   47943.0
```

How's the distribution of numerical columns?

In [39]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
import numpy as np
import warnings
warnings.filterwarnings('ignore')

def plot_distribution(data, features):
    i = 0
    plt.figure()
    col = 3
    row = int(np.ceil(len(features)/col))
    fig, ax = plt.subplots(row,col,figsize=(18,12))
    sns.set_style("whitegrid")

    for feature in features:
        i += 1
        plt.subplot(row,col,i);
```

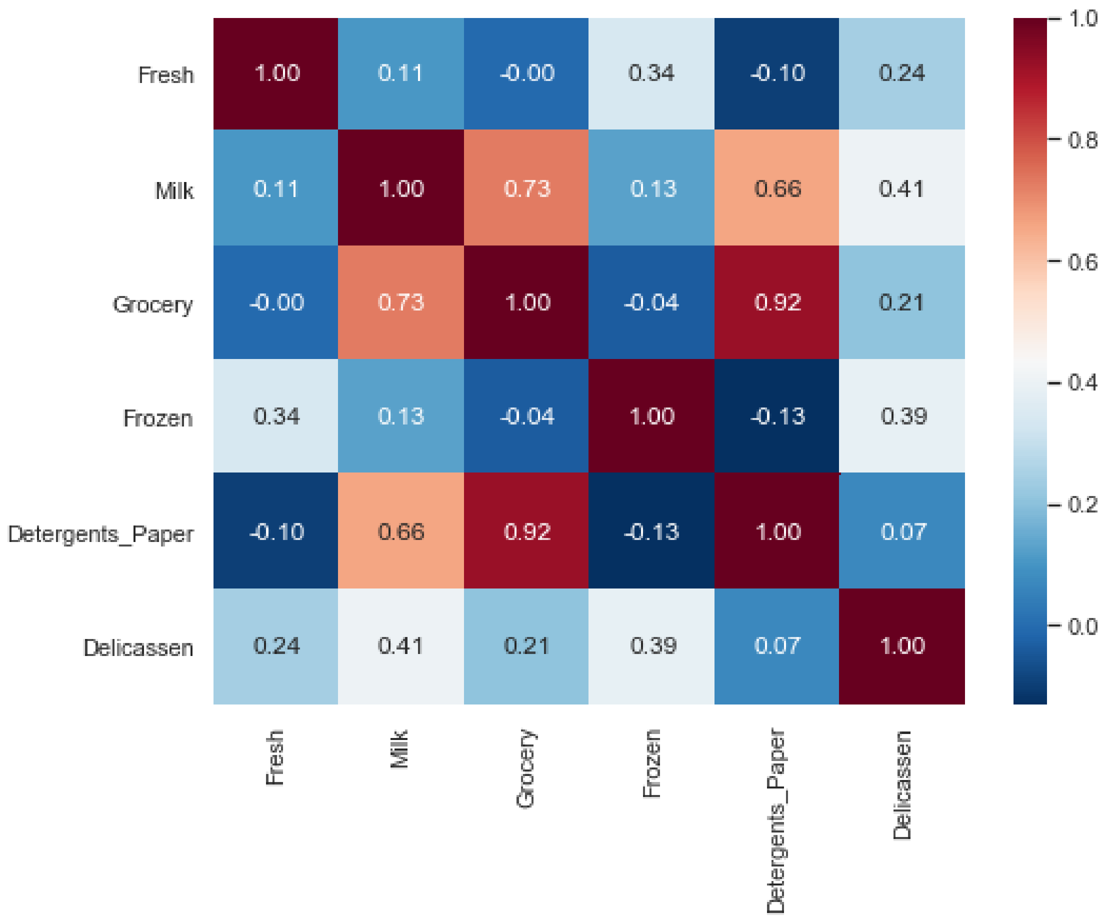
```
            sns.distplot(data[feature])
            plt.xlabel(feature, fontsize=16)
            locs, labels = plt.xticks()
            plt.tick_params(axis='x', which='major', labelsize=12)
            plt.tick_params(axis='y', which='major', labelsize=12)
        plt.show();

    plot_distribution(data_clst, data_clst.columns)
```

```
<matplotlib.figure.Figure at 0x7f1ce97477f0>
```



Well distributed data!
Let's quickly check the correlations between the columns?

```
In [40]: corr = data_clst.corr()
         fig, ax = plt.subplots(figsize=(8,6))
         sns.heatmap(corr,
                     xticklabels=corr.columns.values,
                     yticklabels=corr.columns.values,
                     annot=True, fmt=".2f", cmap="RdBu_r");
```



0.2.18 Step 3: Munging (Data Preprocessing)

Are there any **missing values**?

```
In [41]: pd.DataFrame(data_clst.isnull().sum(), columns=['Missing Values'])
```

```
Out[41]:                   Missing Values
         Fresh                          0
         Milk                           0
         Grocery                        0
         Frozen                         0
         Detergents_Paper               0
         Delicassen                     0
```

DAMN!! We are very lucky!! No missing values!

Wait, this is a unsupervised clustering problem. So we don't need any **data segregation**.

Clustering algorithms **calculates distance from the centroids**. So, it is **highly affective to normalization**. Now, we need to perform log transformation on the data.

In [42]:
```
from sklearn.preprocessing import FunctionTransformer
import numpy as np

scaler = FunctionTransformer()
scaler.fit(data_clst)
data_clst_norm = scaler.transform(data_clst.values)
data_clst_norm = pd.DataFrame(data_clst_norm, columns=data_clst.columns)
```

0.2.19 Step 4: Modeling

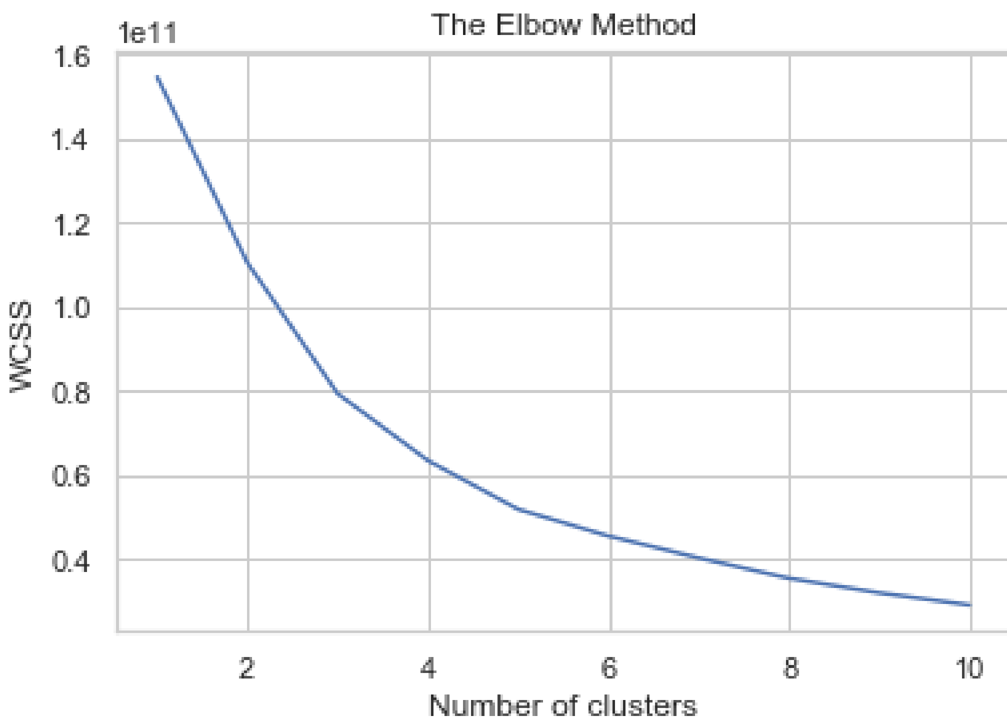
Now, the modeling part!

In this section, we will use a **K-Means** clustering algorithm to identify the various customer segments hidden in the data. We will then recover specific data points from the clusters to understand their significance by transforming them back into their original dimension and scale.

Advantages of K-Means clustering: * Simple, easy to implement and interpret results. * Good for hard cluster assignments i.e. when a data point only belongs to one cluster over the others.

Now, we ned to find the number of clusters using **elbow method**.

In [43]:
```
from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 1234)
    kmeans.fit(data_clst_norm)
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(6,4))
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In [44]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2).fit(data_clst)
reduced_data = pca.transform(data_clst)
reduced_data = pd.DataFrame(reduced_data, columns=['Dimension 1', 'Dimension 2'])
```

In [45]:
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=2)
km.fit(reduced_data)
```

Out[45]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

0.2.20 Step 5: Validation

In [46]:
```
preds = km.predict(reduced_data)
labels = km.labels_
centers = km.cluster_centers_
```

In [47]:
```
norm_centers = pca.inverse_transform(centers)
true_centers = scaler.inverse_transform(norm_centers)
```

```
segments = ['Segment {}'.format(i) for i in range(0,len(centers))]
true_centers = pd.DataFrame(np.round(true_centers), columns=data_clst.keys())
true_centers.index = segments

true_centers
```

Out[47]:

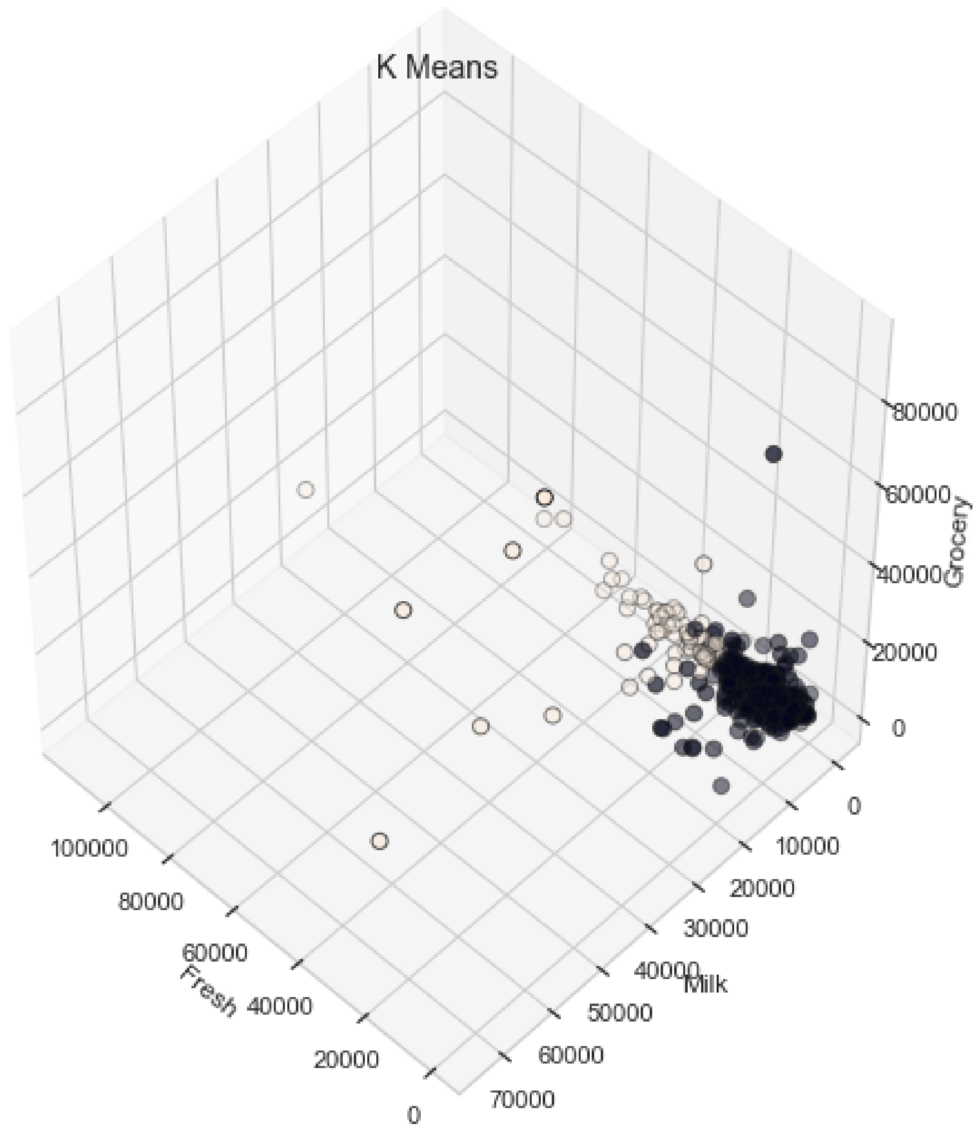
	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Segment 0	7996.0	5162.0	7480.0	2461.0	2795.0	1239.0
Segment 1	35389.0	9353.0	10364.0	6710.0	3151.0	3235.0

Now, let's make inference from the segments:

- **Segment 0**: This segment best represents restaurants. Their spend on Fresh, and Frozen is higher than the median, and lower, but still close to median on Delicassen. Their spend on Milk, Grocery and Detergents_Paper is lower than median, which adds to our assessment.
- **Segment 1**: This segment best represents supermarkets. They spend a higher than median amount on Milk, Grocery, Detergents_Paper and Delicassen, which are both essential to be stocked in such places.

In [48]:

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(1, figsize=(7,7))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
ax.scatter(data_clst_norm.iloc[:, 0],
           data_clst_norm.iloc[:, 1],
           data_clst_norm.iloc[:, 2],
#            data_clst_norm.iloc[:, 3],
#            data_clst_norm.iloc[:, 4],
#            data_clst_norm.iloc[:, 5],
           c=labels.astype(np.float),
           edgecolor="k", s=50)
ax.set_xlabel("Fresh")
ax.set_ylabel("Milk")
ax.set_zlabel("Grocery")
plt.title("K Means", fontsize=14)
plt.show();
```



Learning Resources

- Machine Learning
- Applied Data Science with Python Specialization
- Mathematics for Machine Learning Specialization