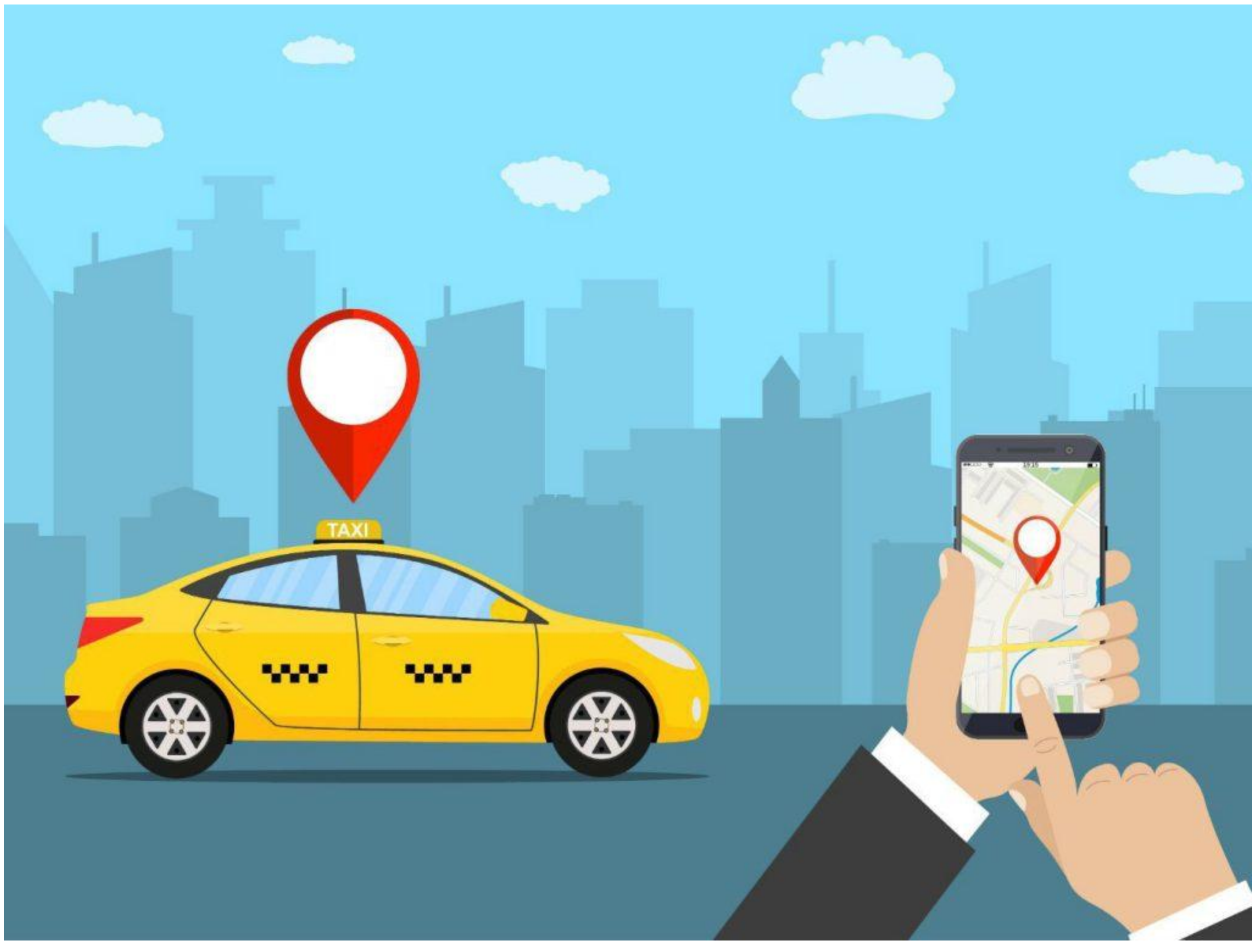
Project 2

Market Segmentation of Cab Service Industry

Contributions: Mohammed Shoaib Khan, Diksha Aggarwal, Soumyajit Ghosh, Sarthak Gupta, Rutuja Pabale



Problem Statement

You are a team working under an Online Vehicle Booking Product Startup. Due to heavy competition in Cab booking from Ola and Uber in India, the startup is looking for an alternate segment which can generate an early foot in the market and revenue. You have to analyze the Vehicle market in India using Segmentation analysis and come up with a feasible strategy to enter the market, targeting the segments where there can be possible profit by offering Vehicle booking service.

Introduction

Online Car Booking management System is developed to manage all cab hiring work online. It is useful for car booking agencies that are specialized in Hiring cabs to customers. Using this system many car-booking agencies are moving ahead to become a pioneer in the vehicle rental industry by completely focusing on customers. Using this system it is very easy for customers to book a car online and car-booking agencies can also track their booking online. So it is also very useful for car booking agencies. It is an online system through which customers can view available cabs; register the cabs, view profile and book cabs. Most people use cab service for their daily transportations needs. Car booking agencies can also check which car is free for booking and which cars are on booking at present time. The objective and scope of my project Online Cab or car booking System is to record the details of various activities of the user. It will simplify the task and reduce the paperwork. Using this car booking management system car owners can also become partners of car booking agencies by giving their car for booking. Online Car rental management system is a web based application that allows users to book a car online. From this system car rental companies can manage all car bookings and customer information. Users can book cars and the admin can confirm the booking and cancel the booking on the basis of availability of the cars and drivers.

We have developed this system to produce a web-based system that allows customers to register and reserve cabs online and for the company to effectively manage their Cab hiring business. Presently car booking agencies do all work offline when a customer comes to them they take the booking order and call the car driver to check their availability with their car. If they manage to find a car for booking they confirm the order otherwise they cancel the order as they have no car for the booking. This process wastes a lot of time of the customer and also of the car booking agency and it also gives a bad name to the agency but with our system car agency can confirm the order within a minute by checking the availability of cars for booking. So this car booking system is helpful to ease customer’s tasks whenever they need to rent a cab or hire a cab.

Fermi Estimation

A Fermi estimate is one done using back-of-the-envelope calculations and rough generalizations to estimate values which would require extensive analysis or experimentation to determine exactly.

Physics is celebrated for its ability to make extremely accurate predictions about tough problems such as the magnetic moment of electrons, the deflection of light by the Sun's gravity, or the orbit of the planets around the Sun. However, accuracy often comes at the cost of great difficulty in calculation.

For example, calculating even a poor estimate for the temperature at which a Bose-Einstein condensate forms requires most people about 15 years of preparation, and an accurate calculation for the flight of a commercial jet is entirely intractable without the aid of sophisticated software systems that handle the difficult numerical calculations. If a nice, aesthetically pleasing analytical result were required, much of physics would not have happened. Indeed, the seduction of formal calculations can be a serious hindrance. Whenever the math gets out of hand, it is usually a good idea to relax demands and accept an approach that, while imprecise, offers a prayer of moving forward.

Fermi Estimation of Cab Industry

Total Population of India = 100,00,00,000

Probability of a person using a cab during a year = 1/10

Number of People using a cab a year = 100,00,00,000*1/10

= 10,00,00,000

Number of People using a cab monthly = 1/10 (12 months in a year)

Number of Rides Monthly = 10,00,00,000*1/10

= 1,00,00,000

= 10^7

Actually rides monthly according to statistica is 50,000,000 which is $5*10^7$

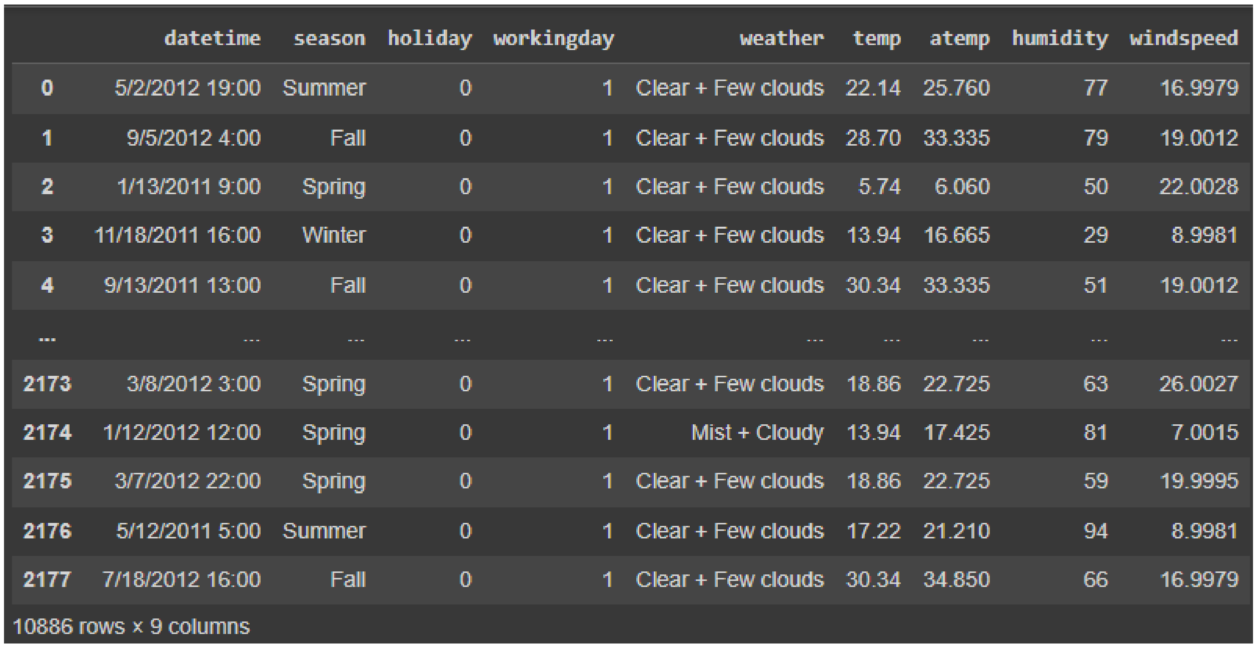
Data Source

The data set used for this project has been collected from kaggle the link for this data set has been provided below this dataset contains two csv files train and test each containing (2178x9) values and (8708x9) respectively, which are combined to form a single dataset containing (10886x9)

https://www.kaggle.com/datasets/sravanidevarakonda/cab-booking-dataset

This data set contains the following attributes

1. **Datetime:** Contains the date and time of the booking of the cab.
2. **Season:** This contains four values: summer, winter, spring ,and fall.
3. **Holiday:** Contains two values 0 and 1 representing whether it was a holiday on the day of booking or not.
4. **Working Day:** Contains two values 0 and 1 representing whether it was a working day on the day of booking or not.
5. **Humidity:** humidity on the day of booking a cab.
6. **Windspeed:** The speed of wind on the day of booking of a cab.



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	5/2/2012 19:00	Summer	0	1	Clear + Few clouds	22.14	25.760	77	16.9979
1	9/5/2012 4:00	Fall	0	1	Clear + Few clouds	28.70	33.335	79	19.0012
2	1/13/2011 9:00	Spring	0	1	Clear + Few clouds	5.74	6.060	50	22.0028
3	11/18/2011 16:00	Winter	0	1	Clear + Few clouds	13.94	16.665	29	8.9981
4	9/13/2011 13:00	Fall	0	1	Clear + Few clouds	30.34	33.335	51	19.0012
...	...	...	...	...	...	...	...	...	...
2173	3/8/2012 3:00	Spring	0	1	Clear + Few clouds	18.86	22.725	63	26.0027
2174	1/12/2012 12:00	Spring	0	1	Mist + Cloudy	13.94	17.425	81	7.0015
2175	3/7/2012 22:00	Spring	0	1	Clear + Few clouds	18.86	22.725	59	19.9995
2176	5/12/2011 5:00	Summer	0	1	Clear + Few clouds	17.22	21.210	94	8.9981
2177	7/18/2012 16:00	Fall	0	1	Clear + Few clouds	30.34	34.850	66	16.9979

10886 rows × 9 columns

Exploratory Data Analysis

What is Exploratory Data Analysis?

Exploratory Data Analysis is investigating data and drawing out insights from it to study its main characteristics. EDA can be done using statistical and visualization techniques.

Why is EDA important?

We simply can't make sense of such huge datasets if we don't explore the data.

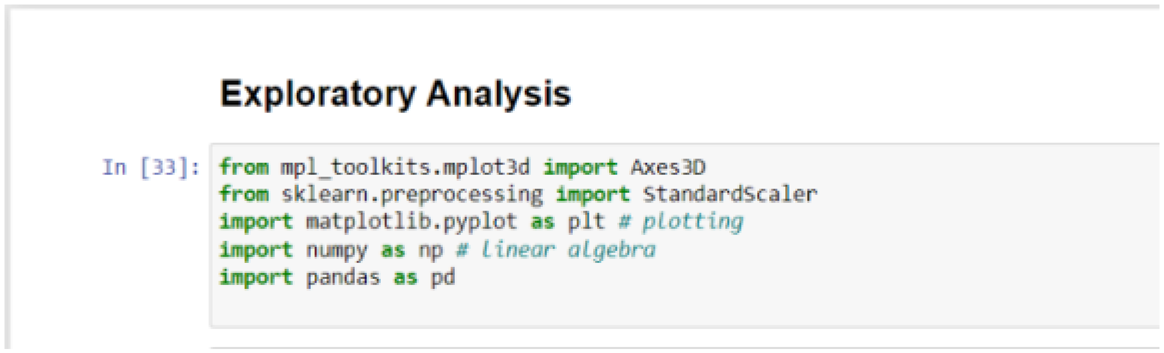
Exploring and analyzing the data is important to see how features are contributing to the target variable, identifying anomalies and outliers to treat them lest they affect our model, to study the nature of the features, and be able to perform data cleaning so that our model building process is as efficient as possible.

If we don't perform exploratory data analysis, we won't be able to find inconsistent or incomplete data that may pose trends incorrectly to our model.

From a business point of view, business stakeholders often have certain assumptions about data. Exploratory Data Analysis helps us look deeper and see if our intuition matches with the data. It helps us see if we are asking the right questions.

This step also serves as the basis for answering our business questions.

To begin this exploratory analysis, we first import the required libraries and define functions for plotting the data using matplotlib.

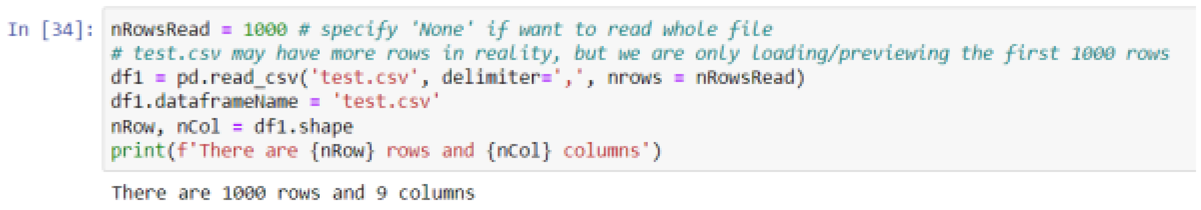


Exploratory Analysis

```
In [33]: from mpl_toolkits.mplot3d import Axes3D
         from sklearn.preprocessing import StandardScaler
         import matplotlib.pyplot as plt # plotting
         import numpy as np # linear algebra
         import pandas as pd
```

Importing the Dataset

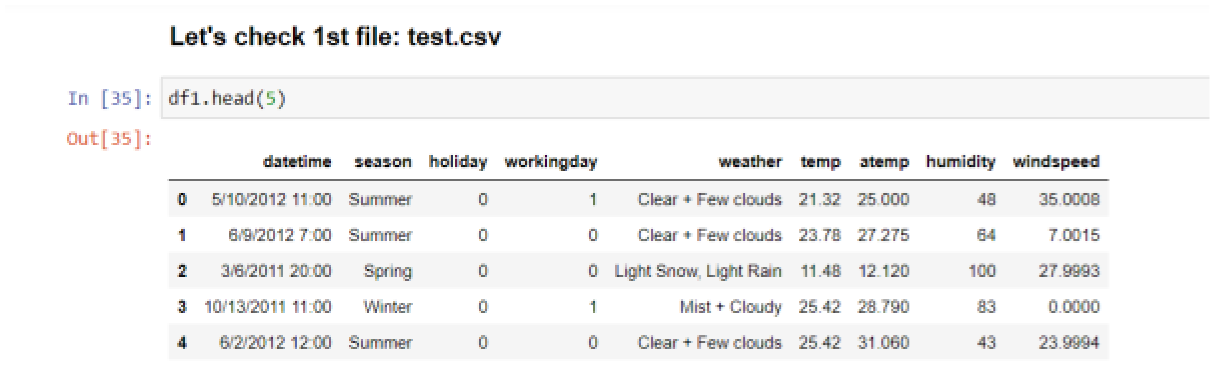
Let us now import the dataset.



```
In [34]: nRowsRead = 1000 # specify 'None' if want to read whole file
         # test.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows
         df1 = pd.read_csv('test.csv', delimiter=',', nrows = nRowsRead)
         df1.dataframeName = 'test.csv'
         nRow, nCol = df1.shape
         print(f'There are {nRow} rows and {nCol} columns')

         There are 1000 rows and 9 columns
```

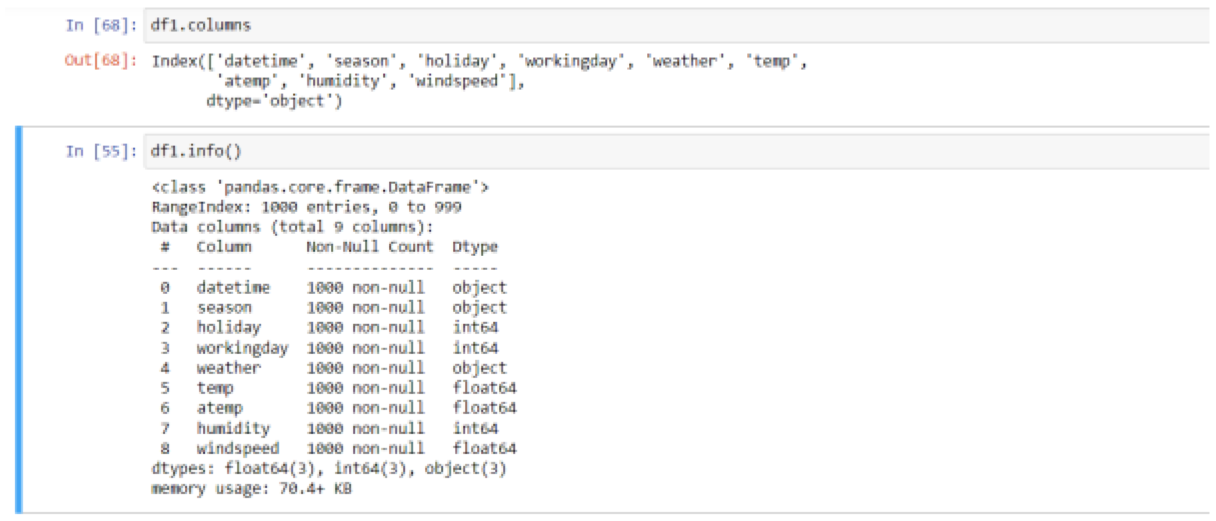
Exploring the Dataset



Let's check 1st file: test.csv

```
In [35]: df1.head(5)
Out[35]:
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	5/10/2012 11:00	Summer	0	1	Clear + Few clouds	21.32	25.000	48	35.0008
1	6/9/2012 7:00	Summer	0	0	Clear + Few clouds	23.78	27.275	64	7.0015
2	3/6/2011 20:00	Spring	0	0	Light Snow, Light Rain	11.48	12.120	100	27.9993
3	10/13/2011 11:00	Winter	0	1	Mist + Cloudy	25.42	28.790	83	0.0000
4	6/2/2012 12:00	Summer	0	0	Clear + Few clouds	25.42	31.060	43	23.9994



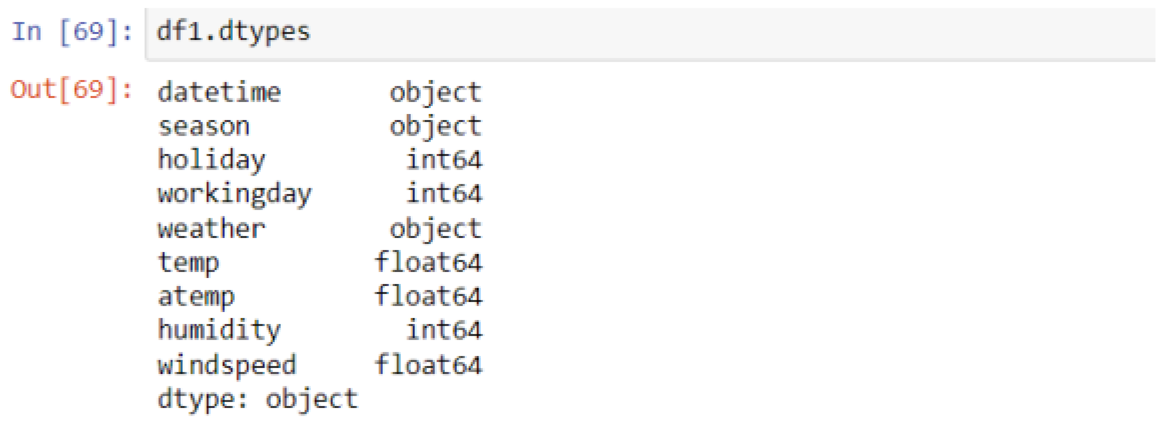
```
In [68]: df1.columns

Out[68]: Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
               'atemp', 'humidity', 'windspeed'],
              dtype='object')
```

```
In [55]: df1.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 1000 entries, 0 to 999
         Data columns (total 9 columns):
          #   Column      Non-Null Count  Dtype
         ---  ------      --------------  -----
          0   datetime    1000 non-null   object
          1   season      1000 non-null   object
          2   holiday     1000 non-null   int64
          3   workingday  1000 non-null   int64
          4   weather     1000 non-null   object
          5   temp        1000 non-null   float64
          6   atemp       1000 non-null   float64
          7   humidity    1000 non-null   int64
          8   windspeed   1000 non-null   float64
         dtypes: float64(3), int64(3), object(3)
         memory usage: 70.4+ KB
```

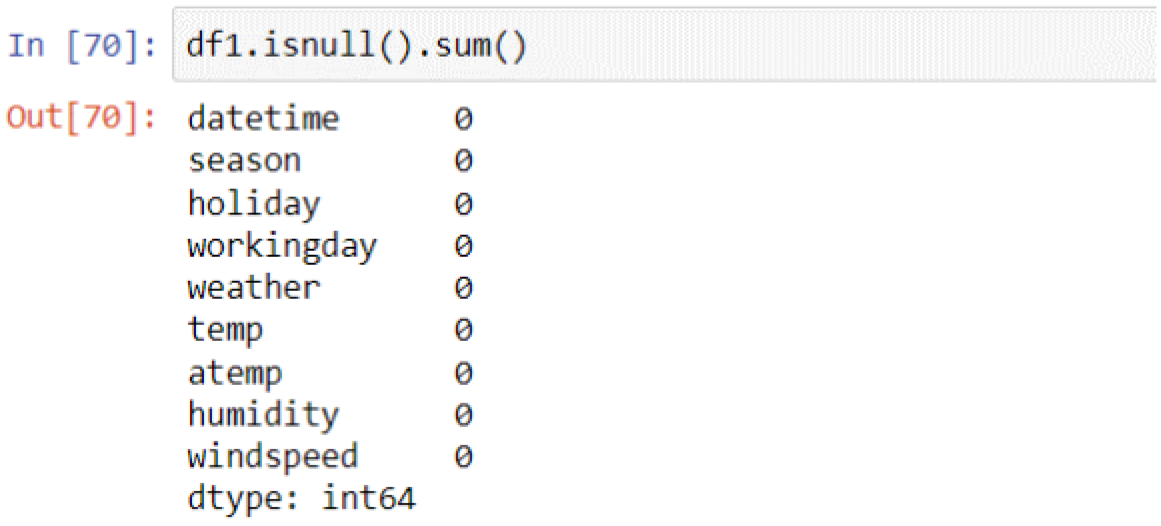
Let us now look at the datatypes of all these columns.



```
In [69]: df1.dtypes

Out[69]: datetime       object
         season         object
         holiday         int64
         workingday      int64
         weather        object
         temp          float64
         atemp         float64
         humidity        int64
         windspeed     float64
         dtype: object
```

Let us see if there are any null values in our dataset.

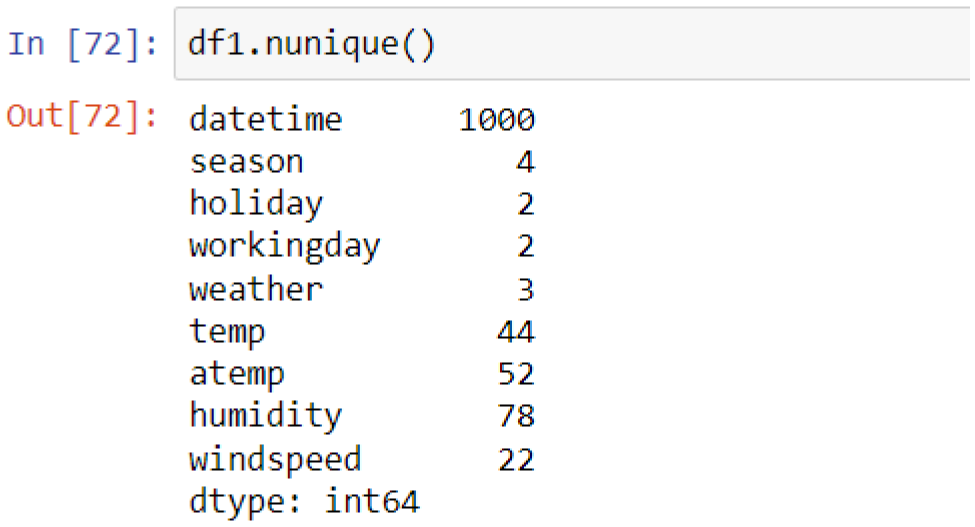


```
In [70]: df1.isnull().sum()

Out[70]: datetime      0
         season        0
         holiday       0
         workingday    0
         weather       0
         temp          0
         atemp         0
         humidity      0
         windspeed     0
         dtype: int64
```

There are no null values in this dataset which saves us a step of imputing.

Let us check for unique values of all columns.

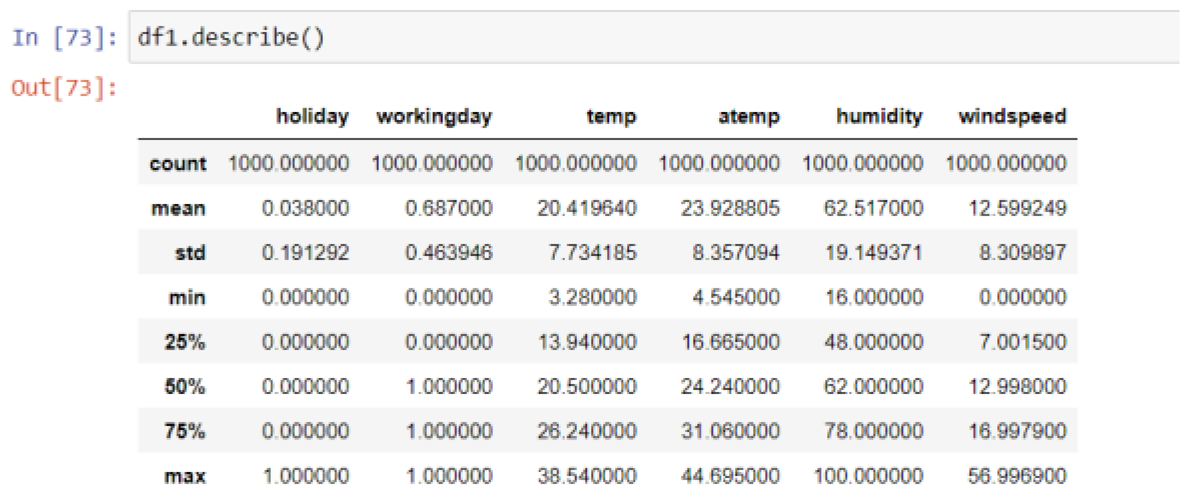


```
In [72]: df1.nunique()

Out[72]: datetime      1000
         season           4
         holiday          2
         workingday       2
         weather          3
         temp            44
         atemp           52
         humidity        78
         windspeed       22
         dtype: int64
```

Let us finally check for a statistical summary of our dataset.

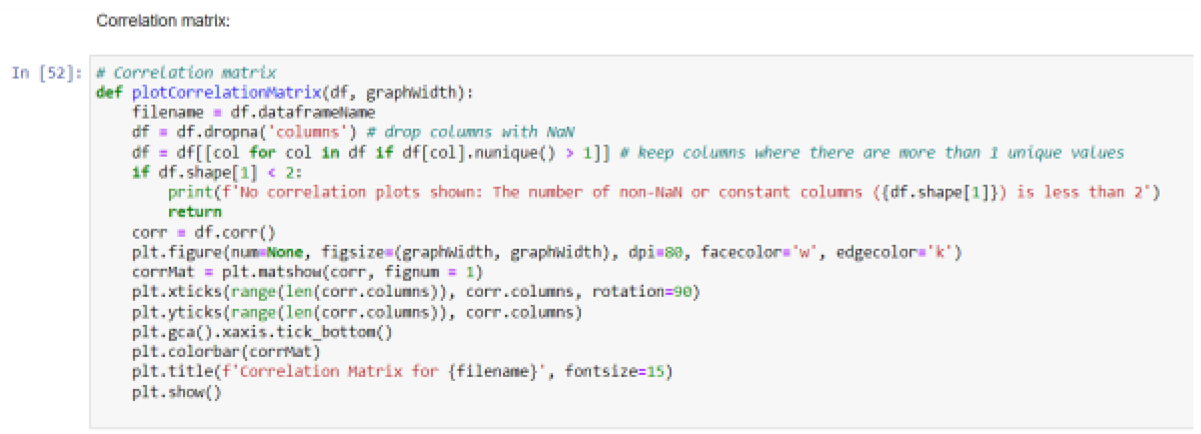
This function can provide statistics for numerical features only.



```
In [73]: df1.describe()
Out[73]:
```

	holiday	workingday	temp	atemp	humidity	windspeed
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.038000	0.687000	20.419640	23.928805	62.517000	12.599249
std	0.191292	0.463946	7.734185	8.357094	19.149371	8.309897
min	0.000000	0.000000	3.280000	4.545000	16.000000	0.000000
25%	0.000000	0.000000	13.940000	16.665000	48.000000	7.001500
50%	0.000000	1.000000	20.500000	24.240000	62.000000	12.998000
75%	0.000000	1.000000	26.240000	31.060000	78.000000	16.997900
max	1.000000	1.000000	38.540000	44.695000	100.000000	56.996900

Now you're ready to read in the data and use the plotting functions to visualize the data.



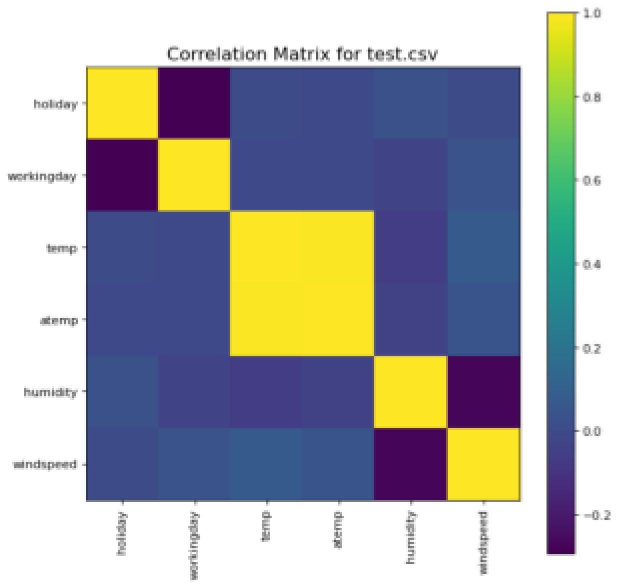
Correlation matrix:

```
In [52]: # Correlation matrix
         def plotCorrelationMatrix(df, graphWidth):
             filename = df.dataframeName
             df = df.dropna('columns') # drop columns with NaN
             df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
             if df.shape[1] < 2:
                 print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less than 2')
                 return
             corr = df.corr()
             plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
             corrMat = plt.matshow(corr, fignum = 1)
             plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
             plt.yticks(range(len(corr.columns)), corr.columns)
             plt.gca().xaxis.tick_bottom()
             plt.colorbar(corrMat)
             plt.title(f'Correlation Matrix for {filename}', fontsize=15)
             plt.show()
```

Now, we will analyze the same results for test_label.csv

```
In [53]: plotCorrelationMatrix(df1, 8)
```

```
C:\Users\HOME\AppData\Local\Temp\ipykernel_8112\128494285.py:4: FutureWarning: In a future version of pandas all arguments of D
ataFrame.dropna will be keyword-only.
  df = df.dropna('columns') # drop columns with NaN
```



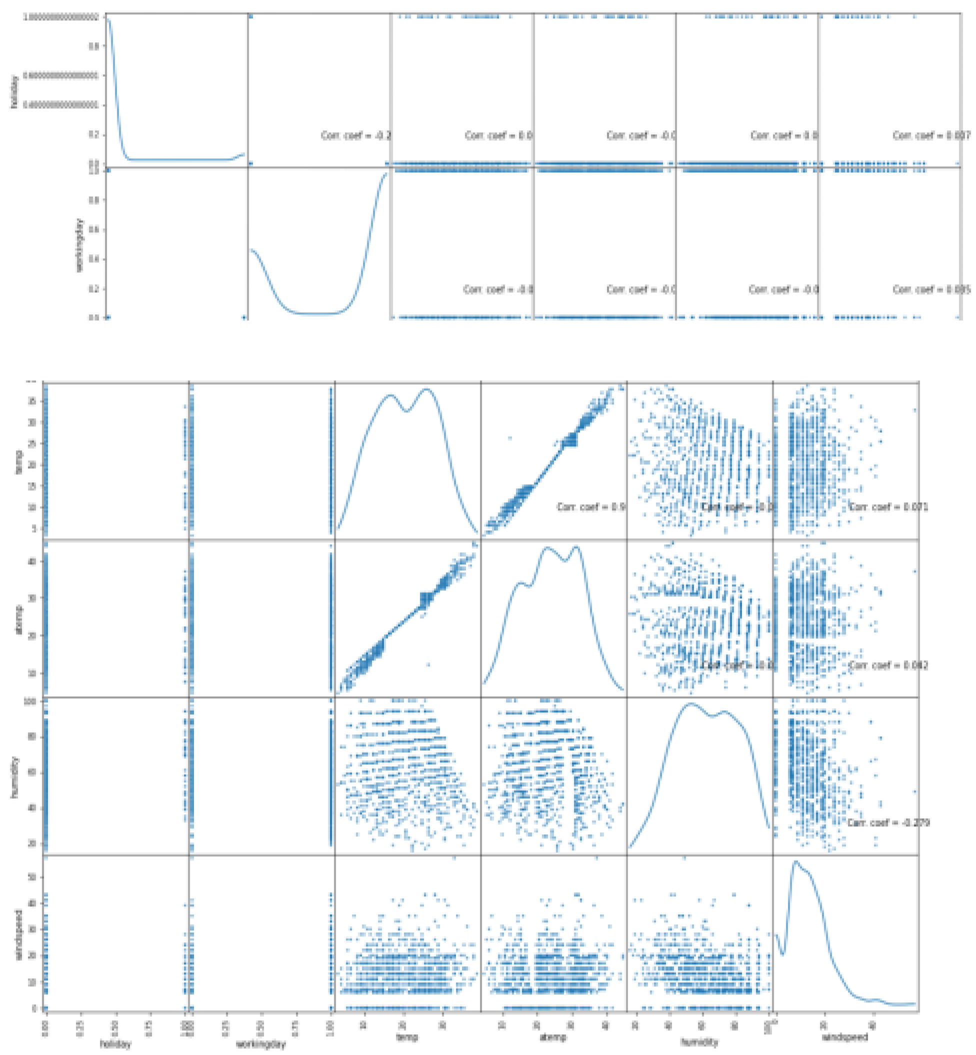
Scatter and density plots:

```
In [56]: # Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center', va='center', siz
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

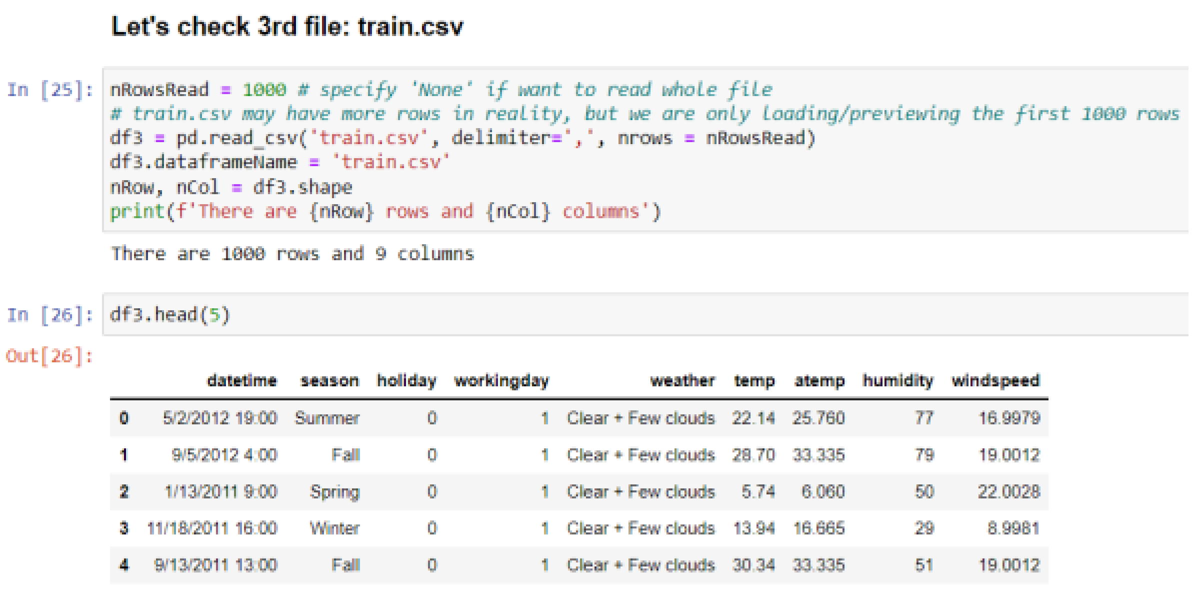
```
In [57]: plotScatterMatrix(df1, 18, 10)
```

```
C:\Users\HOME\AppData\Local\Temp\ipykernel_8112\950191735.py:5: FutureWarning: In a future version of pandas all arguments of D
ataFrame.dropna will be keyword-only.
  df = df.dropna('columns')
```

Scatter and Density Plot



Now, we will analyze the same results for train.csv



Let's check 3rd file: train.csv

```
In [25]: nRowsRead = 1000 # specify 'None' if want to read whole file
         # train.csv may have more rows in reality, but we are only loading/previewing the first 1000 rows
         df3 = pd.read_csv('train.csv', delimiter=',', nrows = nRowsRead)
         df3.dataframeName = 'train.csv'
         nRow, nCol = df3.shape
         print(f'There are {nRow} rows and {nCol} columns')

         There are 1000 rows and 9 columns
```

```
In [26]: df3.head(5)

Out[26]:
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	5/2/2012 19:00	Summer	0	1	Clear + Few clouds	22.14	25.760	77	16.9979
1	9/5/2012 4:00	Fall	0	1	Clear + Few clouds	28.70	33.335	79	19.0012
2	1/13/2011 9:00	Spring	0	1	Clear + Few clouds	5.74	6.060	50	22.0028
3	11/18/2011 16:00	Winter	0	1	Clear + Few clouds	13.94	16.665	29	8.9981
4	9/13/2011 13:00	Fall	0	1	Clear + Few clouds	30.34	33.335	51	19.0012

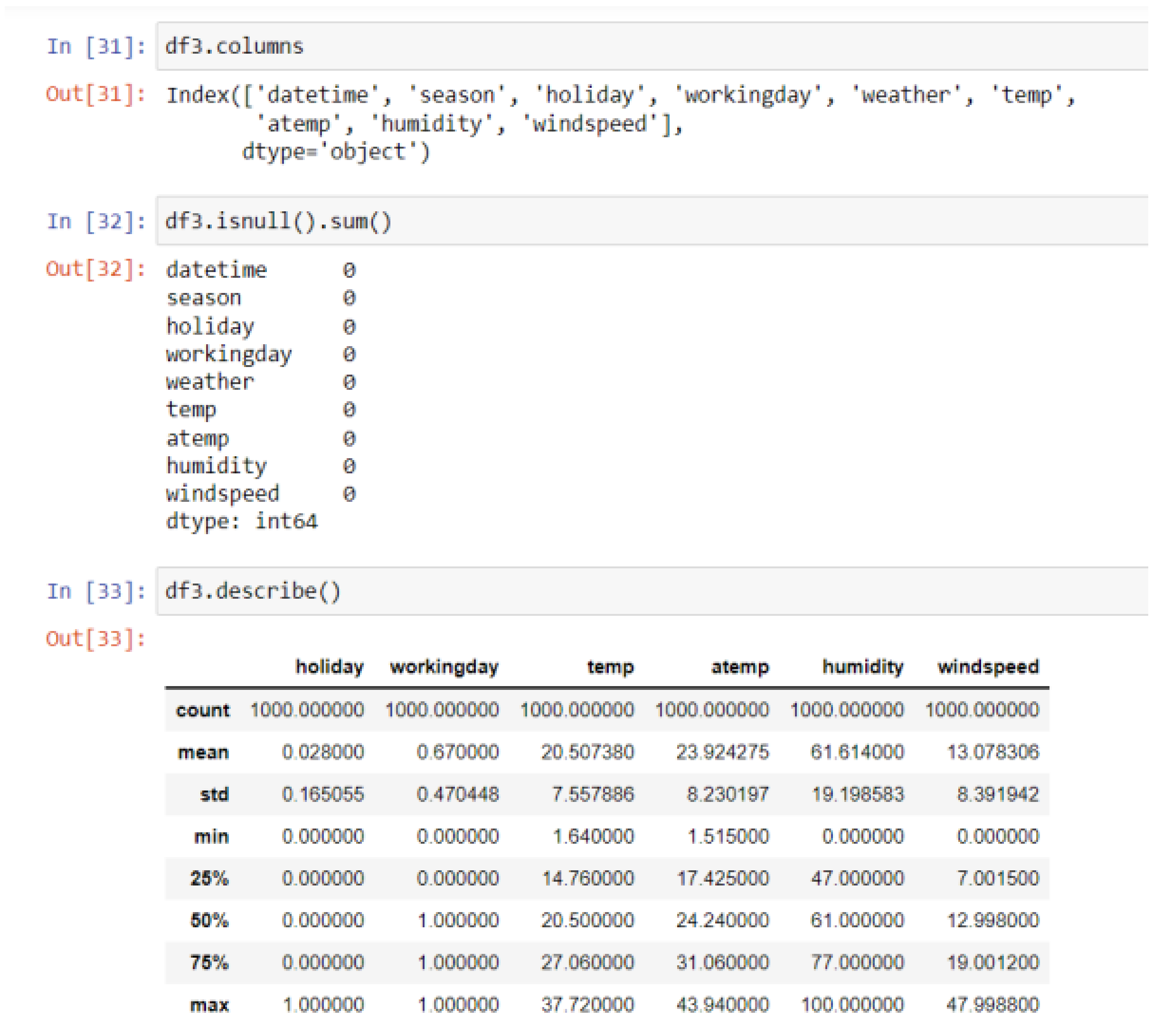
```
In [27]: df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    1000 non-null   object 
 1   season      1000 non-null   object 
 2   holiday     1000 non-null   int64  
 3   workingday  1000 non-null   int64  
 4   weather     1000 non-null   object 
 5   temp        1000 non-null   float64
 6   atemp       1000 non-null   float64
 7   humidity    1000 non-null   int64  
 8   windspeed   1000 non-null   float64
dtypes: float64(3), int64(3), object(3)
memory usage: 70.4+ KB
```

Correlation matrix:

```
In [29]: df3.dtypes
```

```
Out[29]: datetime       object
         season         object
         holiday         int64
         workingday      int64
         weather        object
         temp          float64
         atemp         float64
         humidity        int64
         windspeed     float64
         dtype: object
```

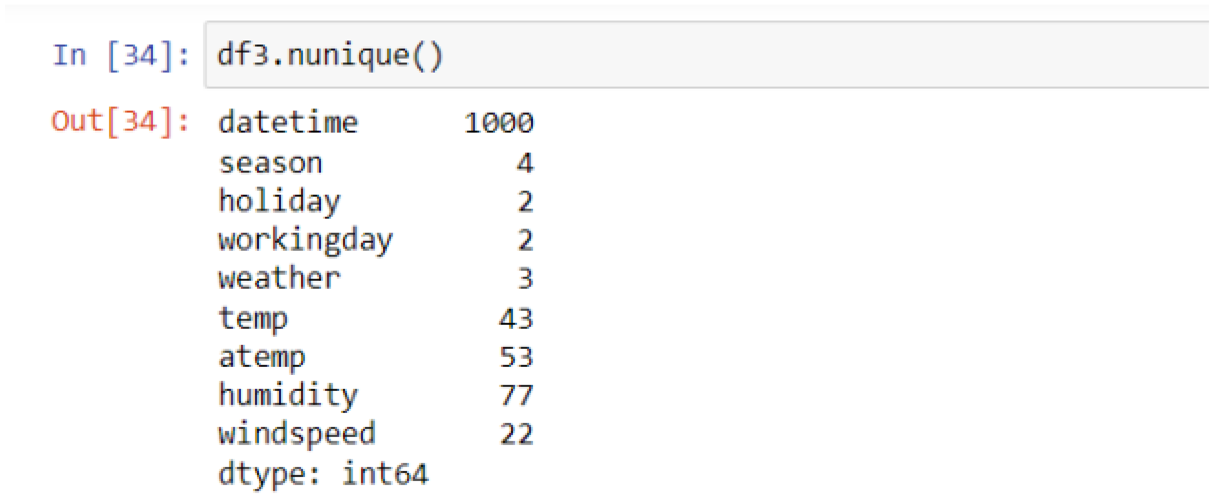


```
In [31]: df3.columns
Out[31]: Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
               'atemp', 'humidity', 'windspeed'],
              dtype='object')
```

```
In [32]: df3.isnull().sum()
Out[32]: datetime      0
         season        0
         holiday       0
         workingday    0
         weather       0
         temp          0
         atemp         0
         humidity      0
         windspeed     0
         dtype: int64
```

```
In [33]: df3.describe()
Out[33]:
```

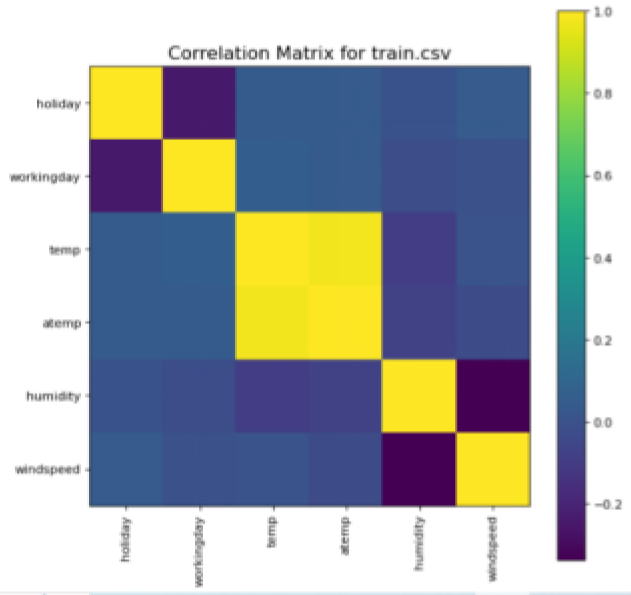
	holiday	workingday	temp	atemp	humidity	windspeed
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.028000	0.670000	20.507380	23.924275	61.614000	13.078306
std	0.165055	0.470448	7.557886	8.230197	19.198583	8.391942
min	0.000000	0.000000	1.640000	1.515000	0.000000	0.000000
25%	0.000000	0.000000	14.760000	17.425000	47.000000	7.001500
50%	0.000000	1.000000	20.500000	24.240000	61.000000	12.998000
75%	0.000000	1.000000	27.060000	31.060000	77.000000	19.001200
max	1.000000	1.000000	37.720000	43.940000	100.000000	47.998800

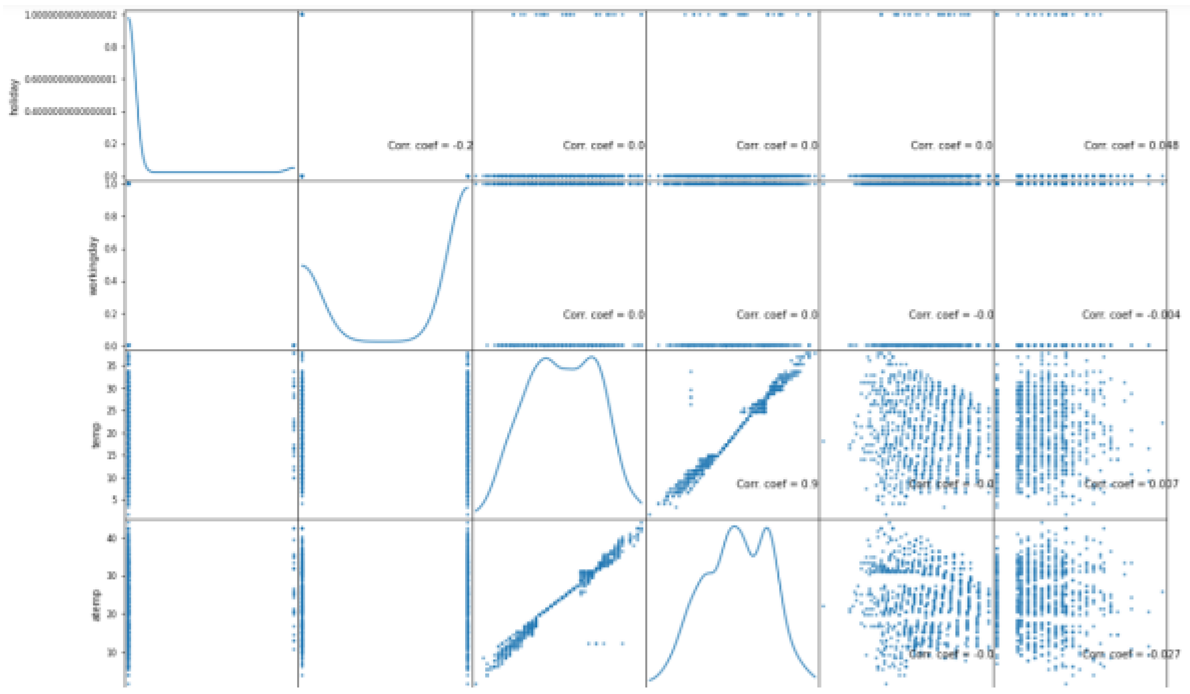


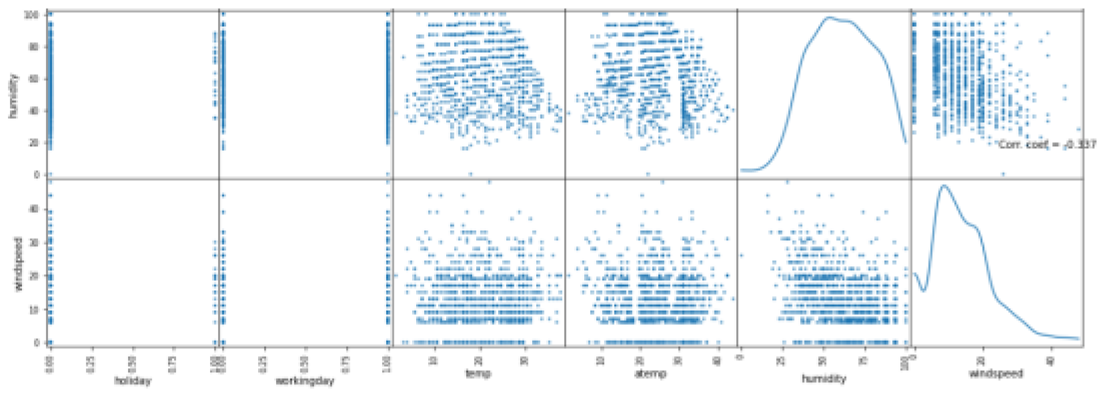
```
In [34]: df3.nunique()
Out[34]: datetime      1000
         season           4
         holiday          2
         workingday       2
         weather          3
         temp            43
         atemp           53
         humidity        77
         windspeed       22
         dtype: int64
```

```
In [65]: plotCorrelationMatrix(df3, 8)
```

```
C:\Users\HOME\AppData\Local\Temp\ipykernel_8112\128494285.py:4: FutureWarning: In a future version of pandas all arguments of D
ataFrame.dropna will be keyword-only.
  df = df.dropna('columns') # drop columns with NaN
```



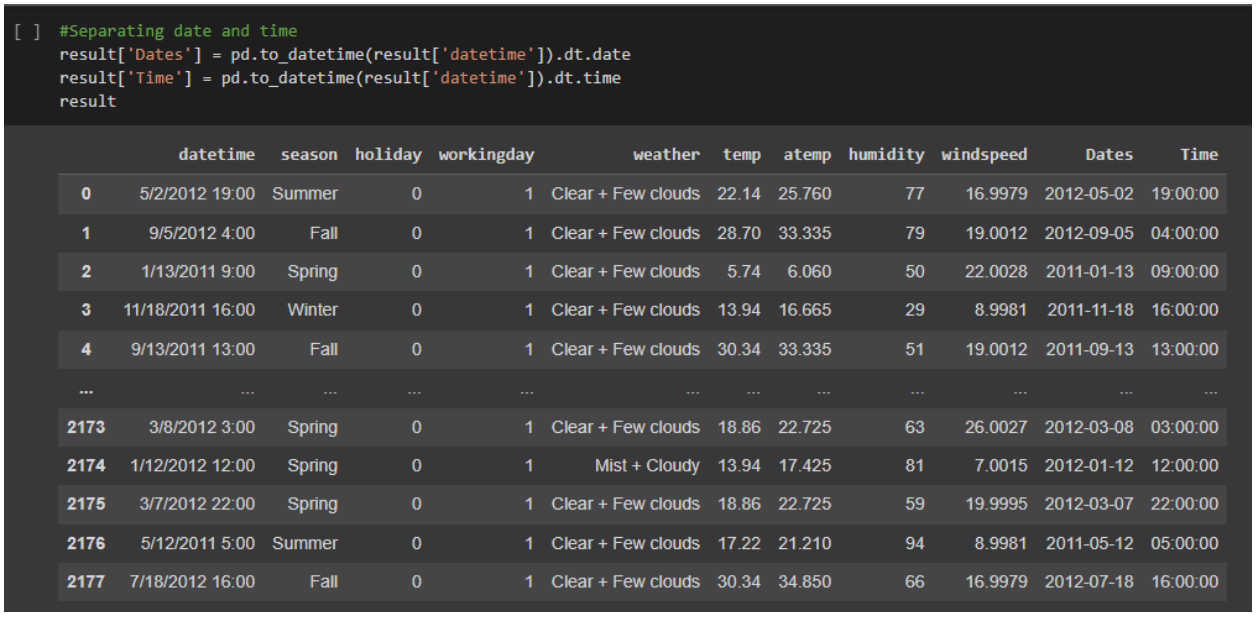




So, we see how Exploratory Data Analysis helps us identify underlying patterns in the data, let us draw out conclusions and this even serves as the basis of feature engineering before we start building our model.

Data Preprocessing

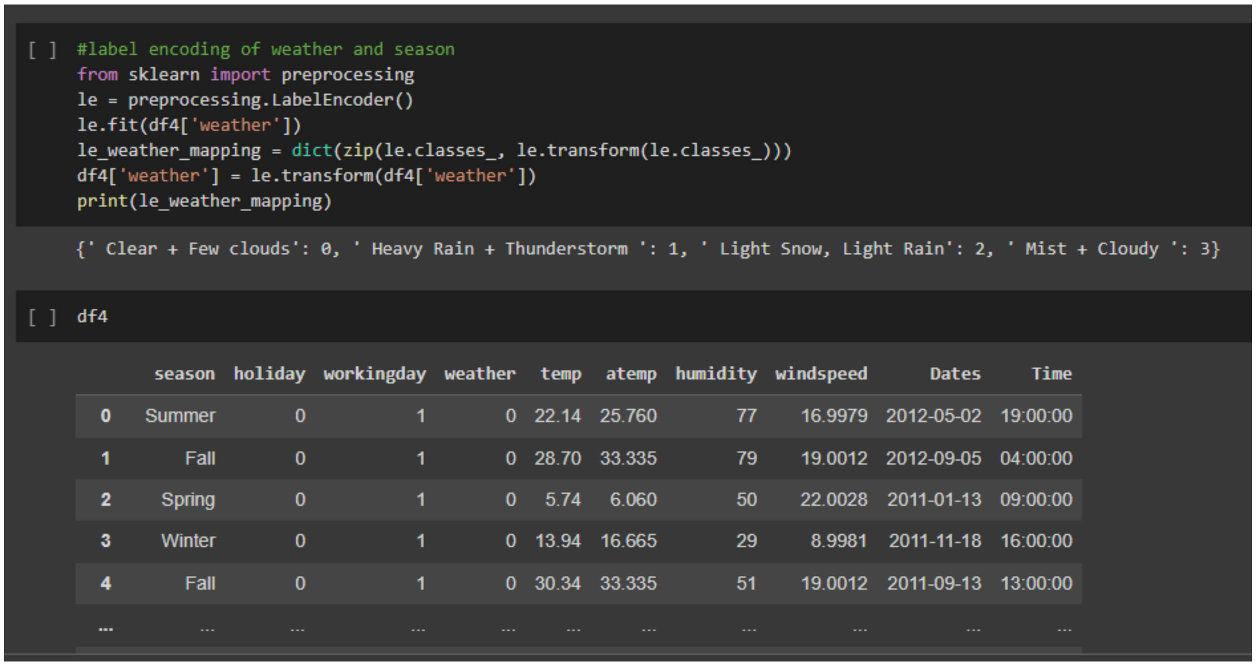
1. First step is to separate date and time from the column datetime.



```
[ ] #Separating date and time
    result['Dates'] = pd.to_datetime(result['datetime']).dt.date
    result['Time'] = pd.to_datetime(result['datetime']).dt.time
    result
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	Dates	Time
0	5/2/2012 19:00	Summer	0	1	Clear + Few clouds	22.14	25.760	77	16.9979	2012-05-02	19:00:00
1	9/5/2012 4:00	Fall	0	1	Clear + Few clouds	28.70	33.335	79	19.0012	2012-09-05	04:00:00
2	1/13/2011 9:00	Spring	0	1	Clear + Few clouds	5.74	6.060	50	22.0028	2011-01-13	09:00:00
3	11/18/2011 16:00	Winter	0	1	Clear + Few clouds	13.94	16.665	29	8.9981	2011-11-18	16:00:00
4	9/13/2011 13:00	Fall	0	1	Clear + Few clouds	30.34	33.335	51	19.0012	2011-09-13	13:00:00
...	...	...	...	...	...	...	...	...	...	...	...
2173	3/8/2012 3:00	Spring	0	1	Clear + Few clouds	18.86	22.725	63	26.0027	2012-03-08	03:00:00
2174	1/12/2012 12:00	Spring	0	1	Mist + Cloudy	13.94	17.425	81	7.0015	2012-01-12	12:00:00
2175	3/7/2012 22:00	Spring	0	1	Clear + Few clouds	18.86	22.725	59	19.9995	2012-03-07	22:00:00
2176	5/12/2011 5:00	Summer	0	1	Clear + Few clouds	17.22	21.210	94	8.9981	2011-05-12	05:00:00
2177	7/18/2012 16:00	Fall	0	1	Clear + Few clouds	30.34	34.850	66	16.9979	2012-07-18	16:00:00

2. Now dropping the original datetime column.
3. Next step is to do label encoding of the weather and season column.

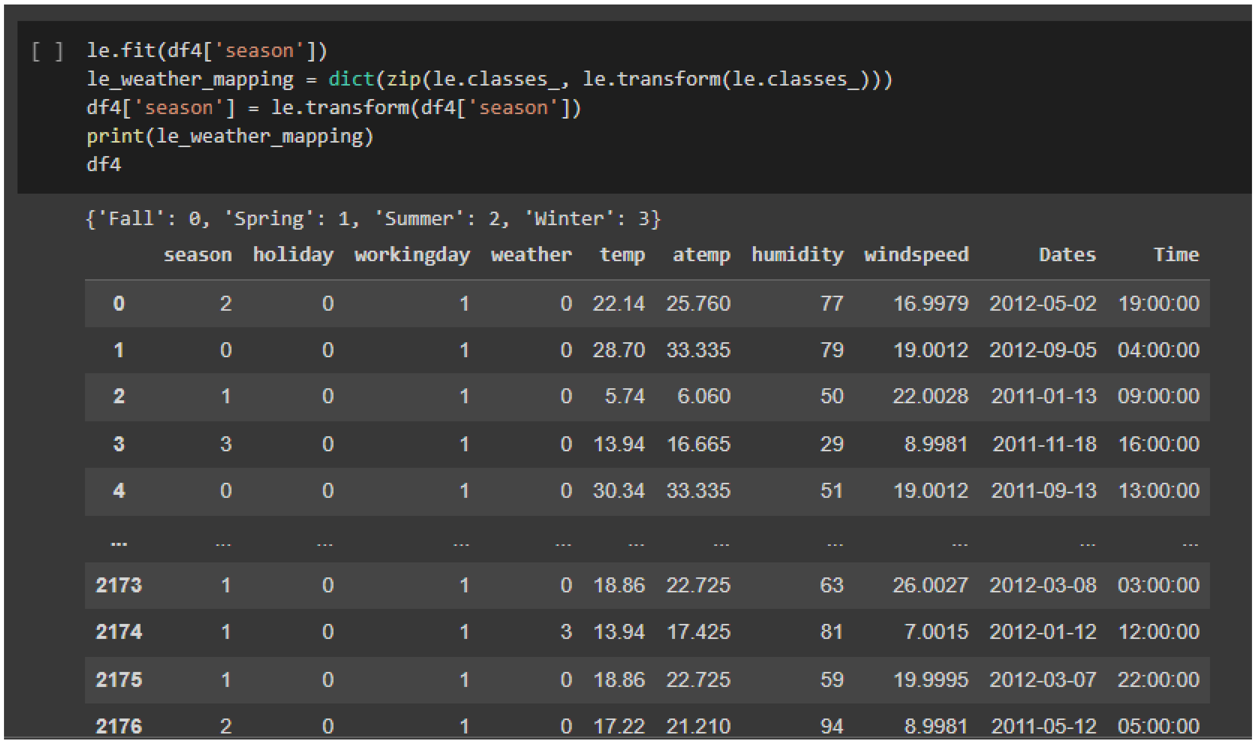


```
[ ] #label encoding of weather and season
    from sklearn import preprocessing
    le = preprocessing.LabelEncoder()
    le.fit(df4['weather'])
    le_weather_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    df4['weather'] = le.transform(df4['weather'])
    print(le_weather_mapping)
```

```
{' Clear + Few clouds': 0, ' Heavy Rain + Thunderstorm ': 1, ' Light Snow, Light Rain': 2, ' Mist + Cloudy ': 3}
```

```
[ ] df4
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	Dates	Time
0	Summer	0	1	0	22.14	25.760	77	16.9979	2012-05-02	19:00:00
1	Fall	0	1	0	28.70	33.335	79	19.0012	2012-09-05	04:00:00
2	Spring	0	1	0	5.74	6.060	50	22.0028	2011-01-13	09:00:00
3	Winter	0	1	0	13.94	16.665	29	8.9981	2011-11-18	16:00:00
4	Fall	0	1	0	30.34	33.335	51	19.0012	2011-09-13	13:00:00
...	...	...	...	...	...	...	...	...	...	...

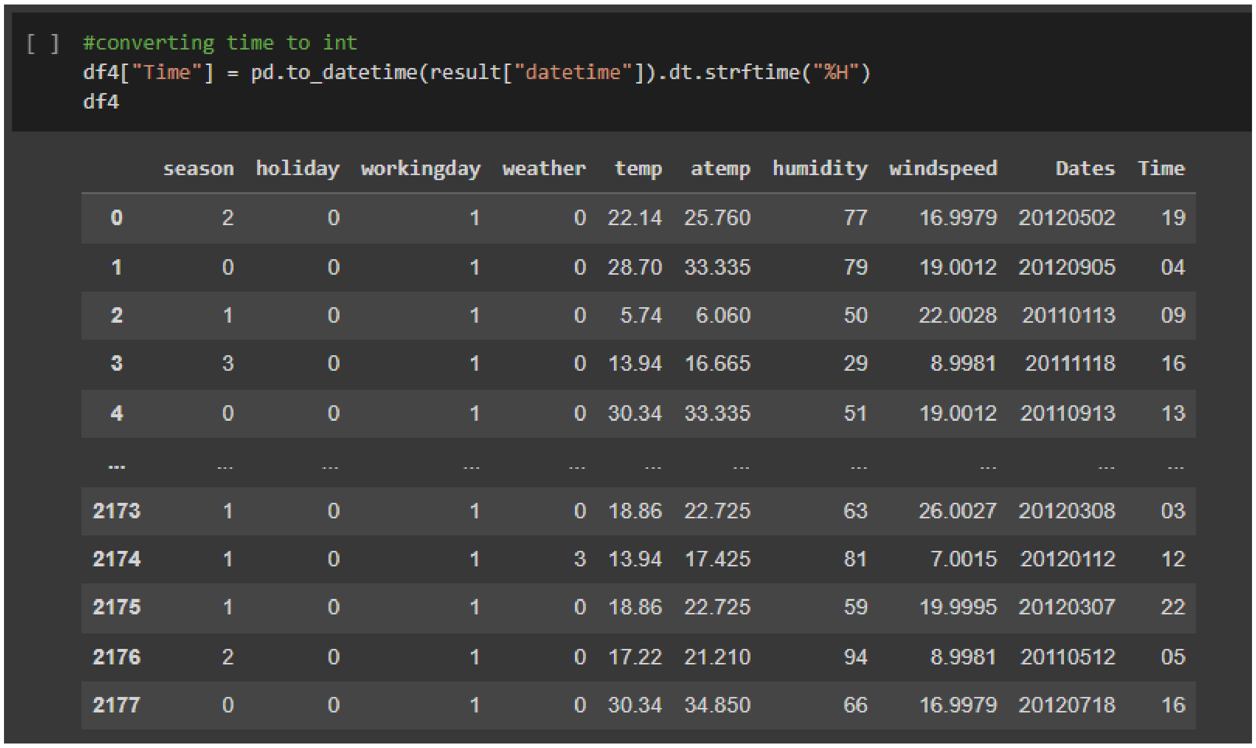


```
[ ] le.fit(df4['season'])
    le_weather_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    df4['season'] = le.transform(df4['season'])
    print(le_weather_mapping)
    df4
```

```
{'Fall': 0, 'Spring': 1, 'Summer': 2, 'Winter': 3}
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	Dates	Time
0	2	0	1	0	22.14	25.760	77	16.9979	2012-05-02	19:00:00
1	0	0	1	0	28.70	33.335	79	19.0012	2012-09-05	04:00:00
2	1	0	1	0	5.74	6.060	50	22.0028	2011-01-13	09:00:00
3	3	0	1	0	13.94	16.665	29	8.9981	2011-11-18	16:00:00
4	0	0	1	0	30.34	33.335	51	19.0012	2011-09-13	13:00:00
...	...	...	...	...	...	...	...	...	...	...
2173	1	0	1	0	18.86	22.725	63	26.0027	2012-03-08	03:00:00
2174	1	0	1	3	13.94	17.425	81	7.0015	2012-01-12	12:00:00
2175	1	0	1	0	18.86	22.725	59	19.9995	2012-03-07	22:00:00
2176	2	0	1	0	17.22	21.210	94	8.9981	2011-05-12	05:00:00

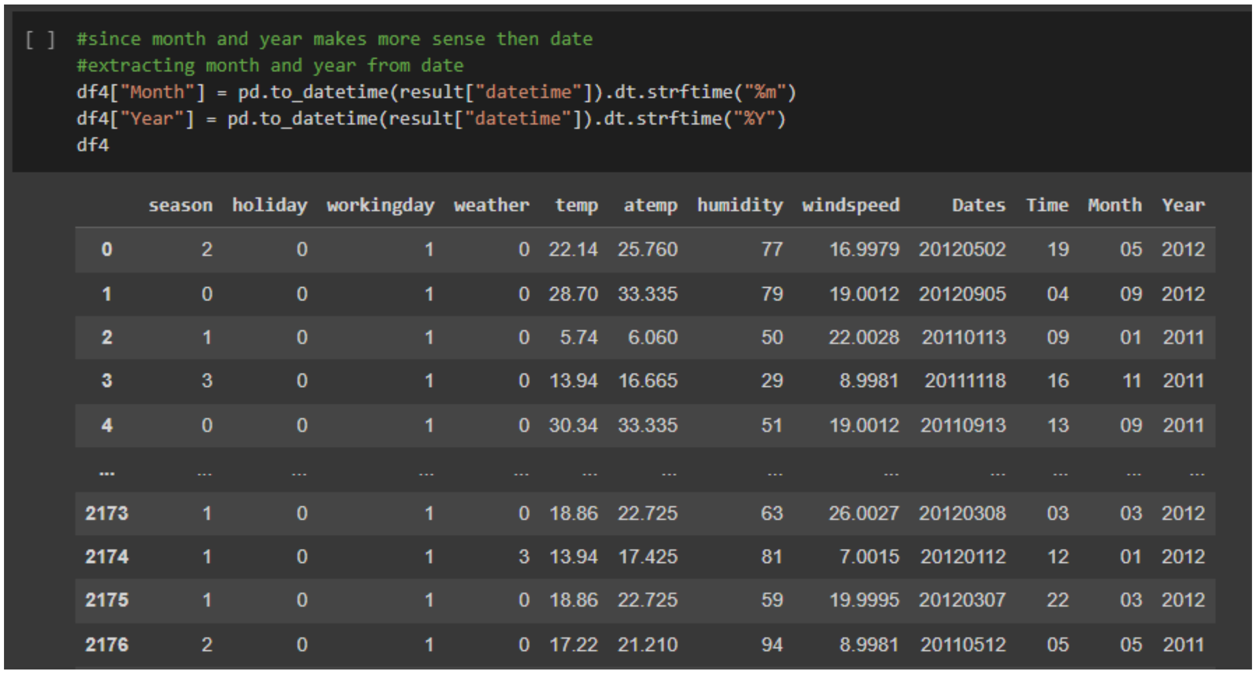
4. Hours is the most important value in the time column, so we will extract hours from the time column.



```
[ ]  #converting time to int
     df4["Time"] = pd.to_datetime(result["datetime"]).dt.strftime("%H")
     df4
```

```
      season  holiday  workingday  weather   temp   atemp  humidity  windspeed     Dates  Time
0          2        0           1        0  22.14  25.760        77    16.9979  20120502    19
1          0        0           1        0  28.70  33.335        79    19.0012  20120905    04
2          1        0           1        0   5.74   6.060        50    22.0028  20110113    09
3          3        0           1        0  13.94  16.665        29     8.9981  20111118    16
4          0        0           1        0  30.34  33.335        51    19.0012  20110913    13
...      ...      ...         ...      ...    ...     ...       ...        ...       ...   ...
2173       1        0           1        0  18.86  22.725        63    26.0027  20120308    03
2174       1        0           1        3  13.94  17.425        81     7.0015  20120112    12
2175       1        0           1        0  18.86  22.725        59    19.9995  20120307    22
2176       2        0           1        0  17.22  21.210        94     8.9981  20110512    05
2177       0        0           1        0  30.34  34.850        66    16.9979  20120718    16
```

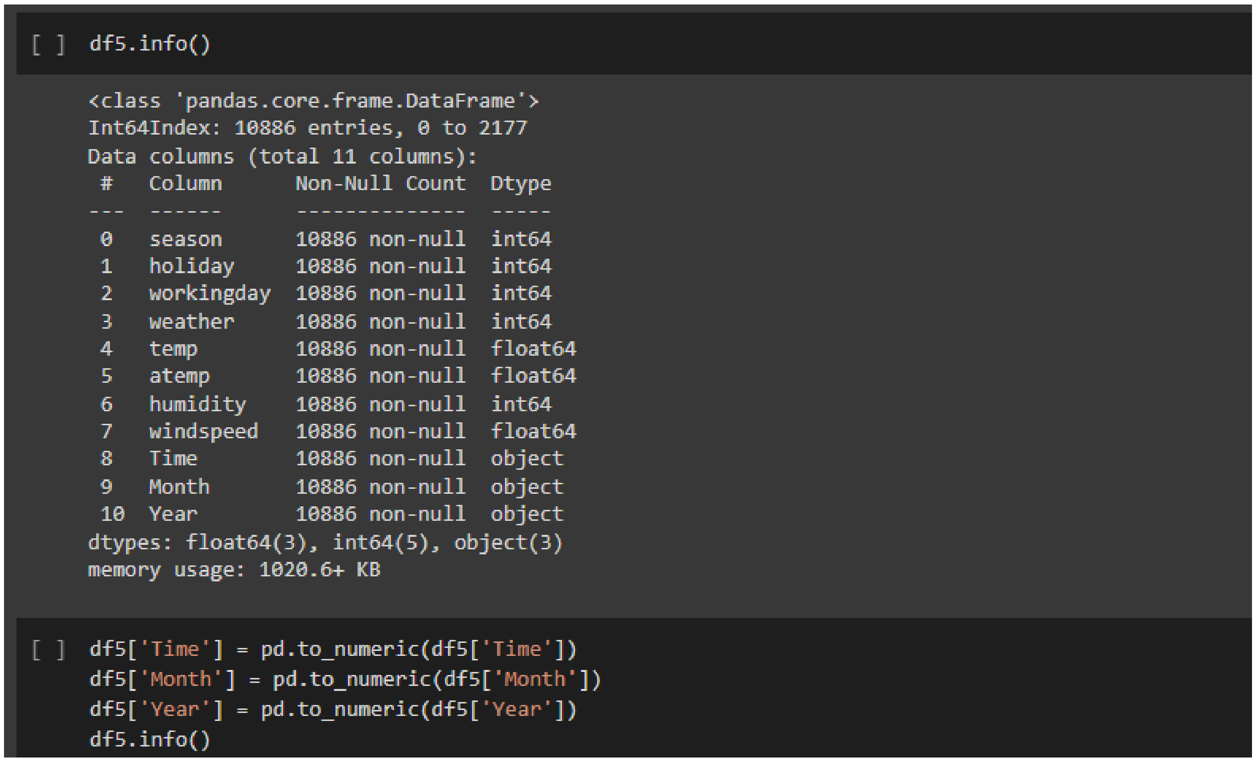
5. Similarly, year and month have more significance in the date column, so making two separate columns for month and year.



```
[ ]  #since month and year makes more sense then date
     #extracting month and year from date
     df4["Month"] = pd.to_datetime(result["datetime"]).dt.strftime("%m")
     df4["Year"] = pd.to_datetime(result["datetime"]).dt.strftime("%Y")
     df4
```

```
      season  holiday  workingday  weather   temp   atemp  humidity  windspeed     Dates  Time  Month  Year
0          2        0           1        0  22.14  25.760        77    16.9979  20120502    19     05  2012
1          0        0           1        0  28.70  33.335        79    19.0012  20120905    04     09  2012
2          1        0           1        0   5.74   6.060        50    22.0028  20110113    09     01  2011
3          3        0           1        0  13.94  16.665        29     8.9981  20111118    16     11  2011
4          0        0           1        0  30.34  33.335        51    19.0012  20110913    13     09  2011
...      ...      ...         ...      ...    ...     ...       ...        ...       ...   ...    ...   ...
2173       1        0           1        0  18.86  22.725        63    26.0027  20120308    03     03  2012
2174       1        0           1        3  13.94  17.425        81     7.0015  20120112    12     01  2012
2175       1        0           1        0  18.86  22.725        59    19.9995  20120307    22     03  2012
2176       2        0           1        0  17.22  21.210        94     8.9981  20110512    05     05  2011
```

6. Since Time, Months and Year are non numeric values converting them to float64.



```
[ ]  df5.info()

     <class 'pandas.core.frame.DataFrame'>
     Int64Index: 10886 entries, 0 to 2177
     Data columns (total 11 columns):
      #   Column      Non-Null Count  Dtype
     ---  ------      --------------  -----
      0   season      10886 non-null  int64
      1   holiday     10886 non-null  int64
      2   workingday  10886 non-null  int64
      3   weather     10886 non-null  int64
      4   temp        10886 non-null  float64
      5   atemp       10886 non-null  float64
      6   humidity    10886 non-null  int64
      7   windspeed   10886 non-null  float64
      8   Time        10886 non-null  object
      9   Month       10886 non-null  object
      10  Year        10886 non-null  object
     dtypes: float64(3), int64(5), object(3)
     memory usage: 1020.6+ KB

[ ]  df5['Time'] = pd.to_numeric(df5['Time'])
     df5['Month'] = pd.to_numeric(df5['Month'])
     df5['Year'] = pd.to_numeric(df5['Year'])
     df5.info()
```

Extracting Segments

K-Means Clustering

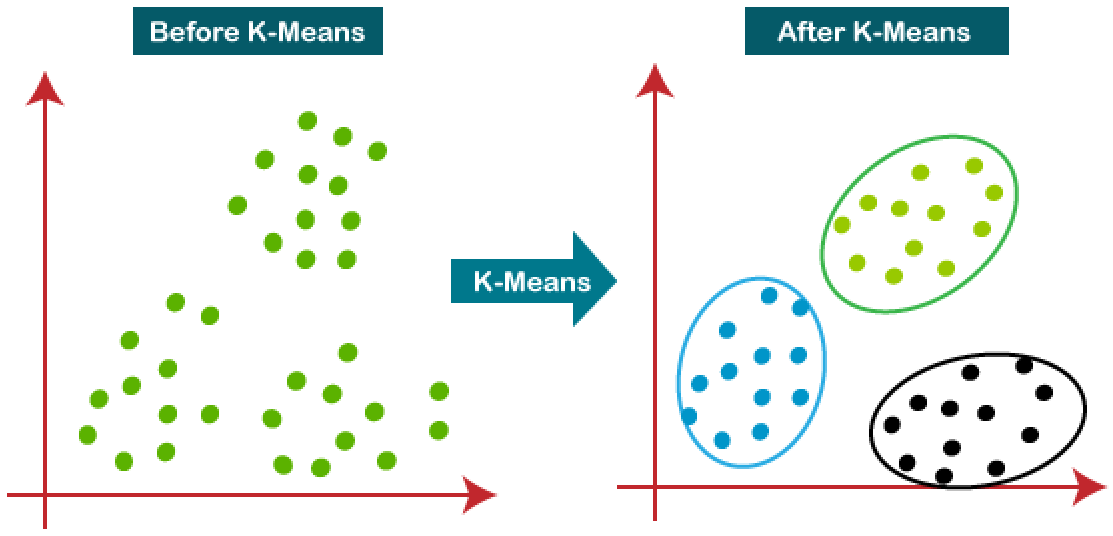
K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what K-means clustering algorithm is, how the algorithm works, along with the Python implementation of k-means clustering.

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

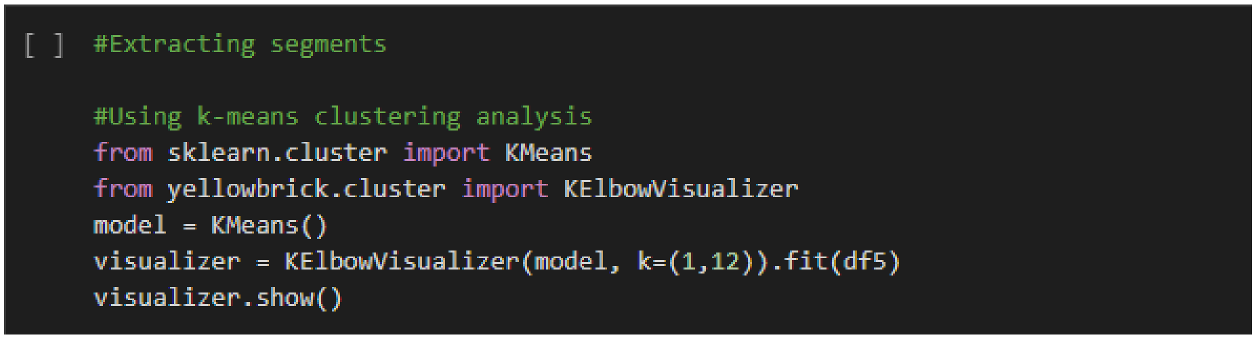
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

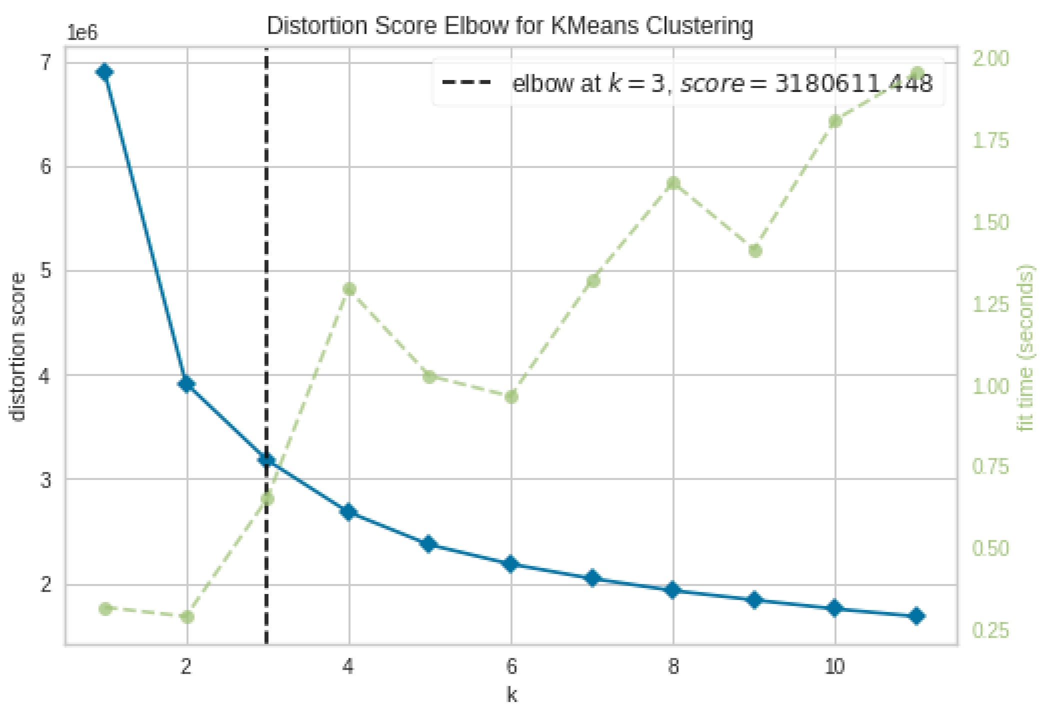


Now Applying K Means Algorithm to the dataset

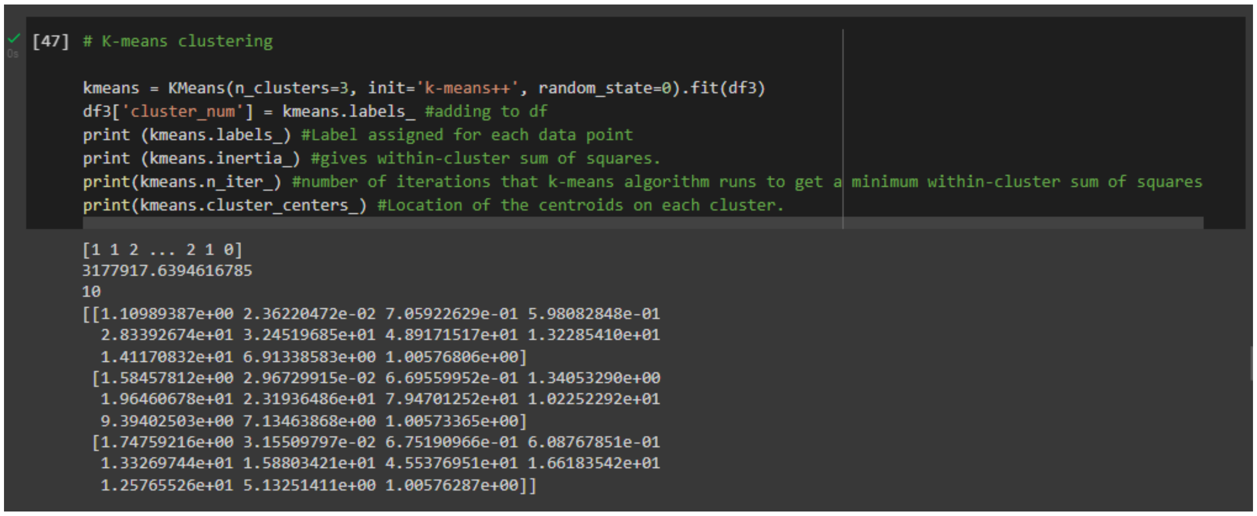


```
[ ]  #Extracting segments

     #Using k-means clustering analysis
     from sklearn.cluster import KMeans
     from yellowbrick.cluster import KElbowVisualizer
     model = KMeans()
     visualizer = KElbowVisualizer(model, k=(1,12)).fit(df5)
     visualizer.show()
```



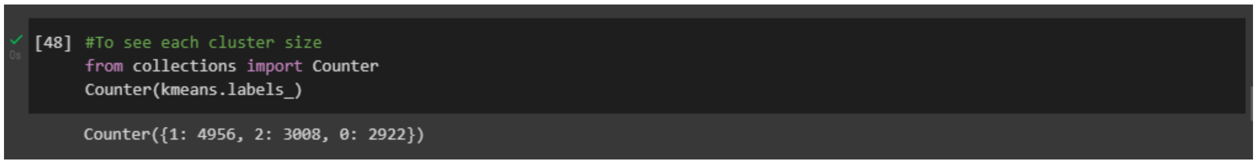
The elbow plot shows 3 is the ideal number of segments to be formed. So dividing the data in 3 clusters.



```
[47] # K-means clustering

     kmeans = KMeans(n_clusters=3, init='k-means++', random_state=0).fit(df3)
     df3['cluster_num'] = kmeans.labels_ #adding to df
     print (kmeans.labels_) #Label assigned for each data point
     print (kmeans.inertia_) #gives within-cluster sum of squares.
     print(kmeans.n_iter_) #number of iterations that k-means algorithm runs to get a minimum within-cluster sum of squares
     print(kmeans.cluster_centers_) #Location of the centroids on each cluster.
```

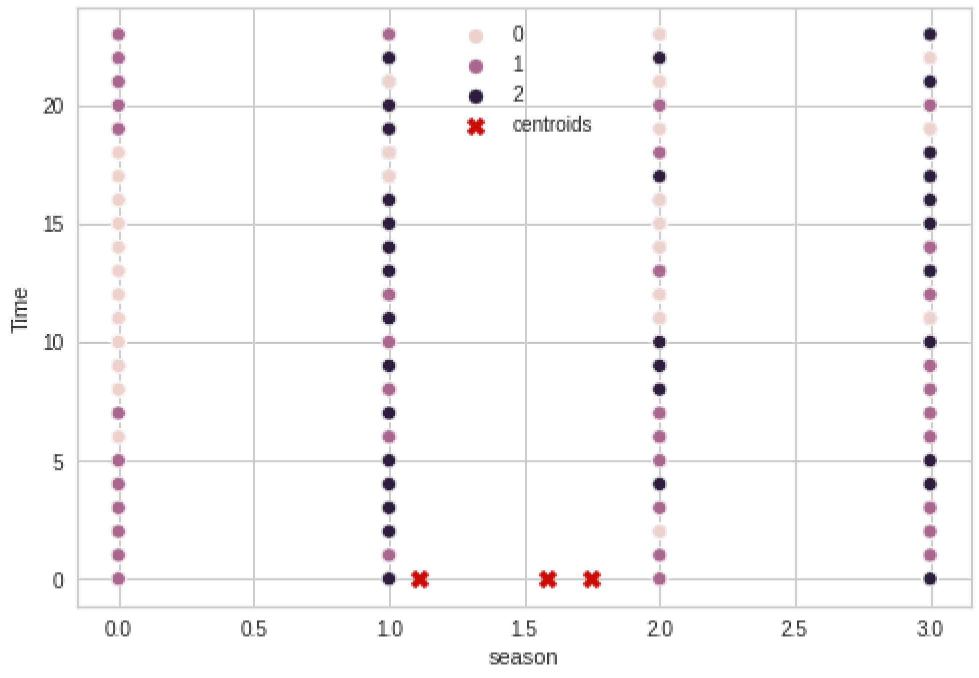
```
[1 1 2 ... 2 1 0]
3177917.6394616785
10
[[1.10989387e+00 2.36220472e-02 7.05922629e-01 5.98082848e-01
  2.83392674e+01 3.24519685e+01 4.89171517e+01 1.32285410e+01
  1.41170832e+01 6.91338583e+00 1.00576806e+00]
 [1.58457812e+00 2.96729915e-02 6.69559952e-01 1.34053290e+00
  1.96460678e+01 2.31936486e+01 7.94701252e+01 1.02252292e+01
  9.39402503e+00 7.13463868e+00 1.00573365e+00]
 [1.74759216e+00 3.15509797e-02 6.75190966e-01 6.08767851e-01
  1.33269744e+01 1.58803421e+01 4.55376951e+01 1.66183542e+01
  1.25765526e+01 5.13251411e+00 1.00576287e+00]]
```

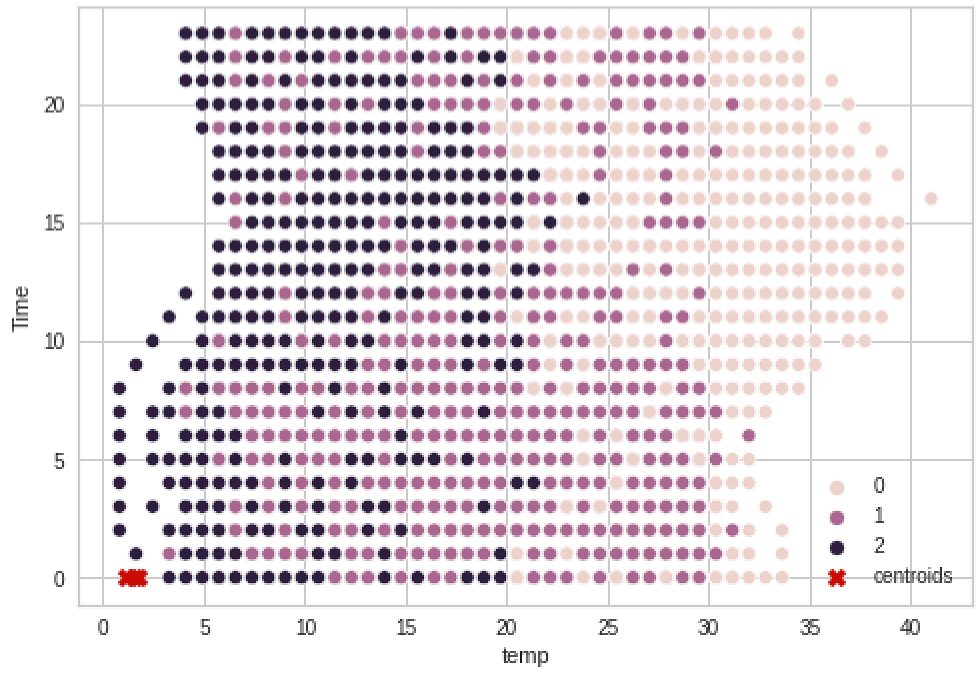


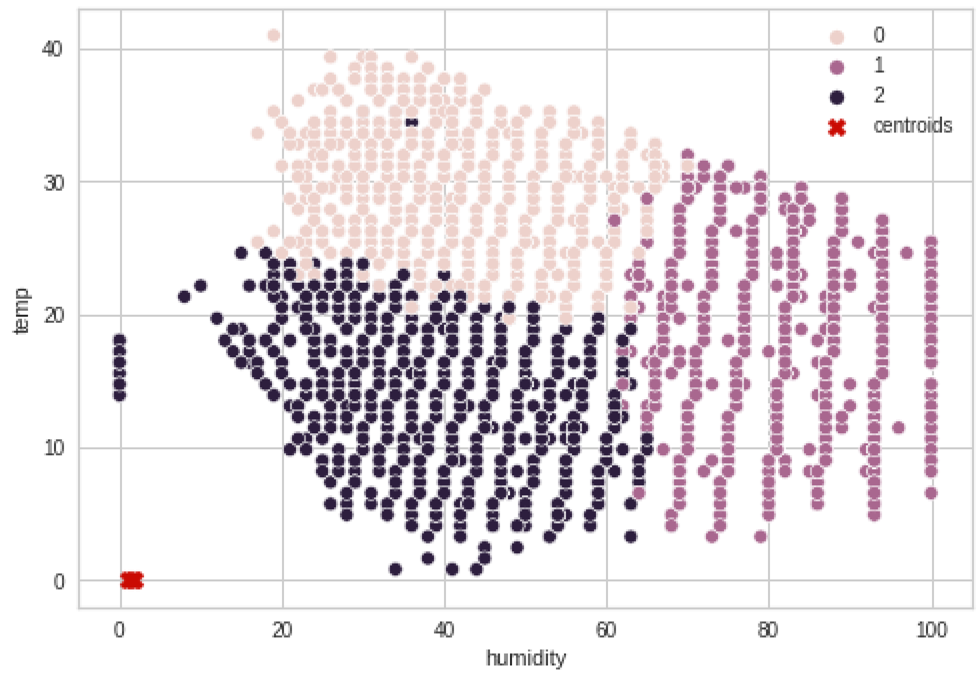
```
[48] #To see each cluster size
     from collections import Counter
     Counter(kmeans.labels_)
```

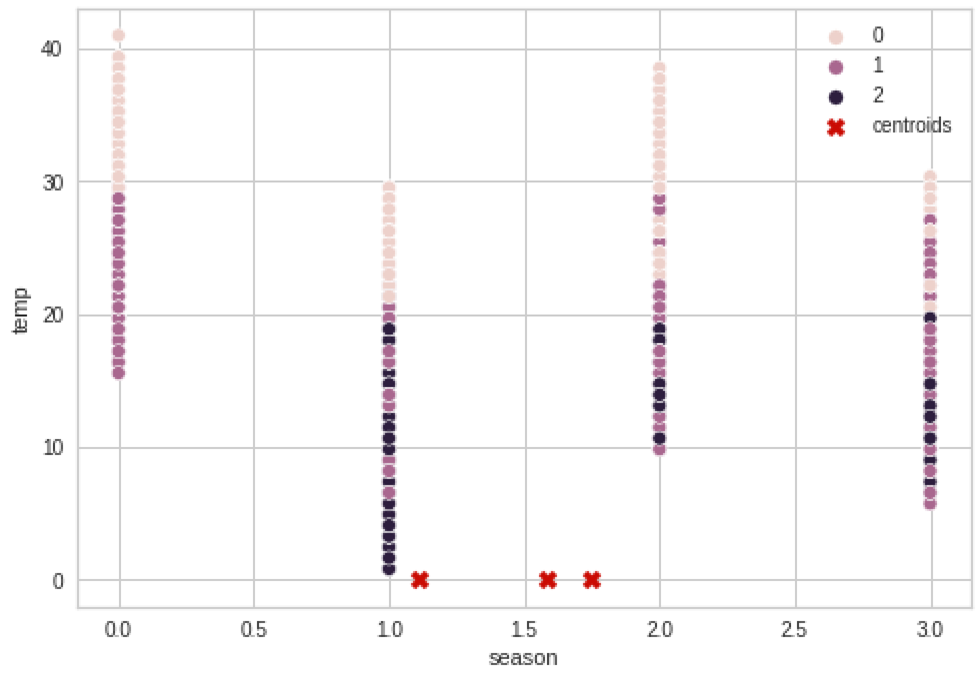
```
Counter({1: 4956, 2: 3008, 0: 2922})
```

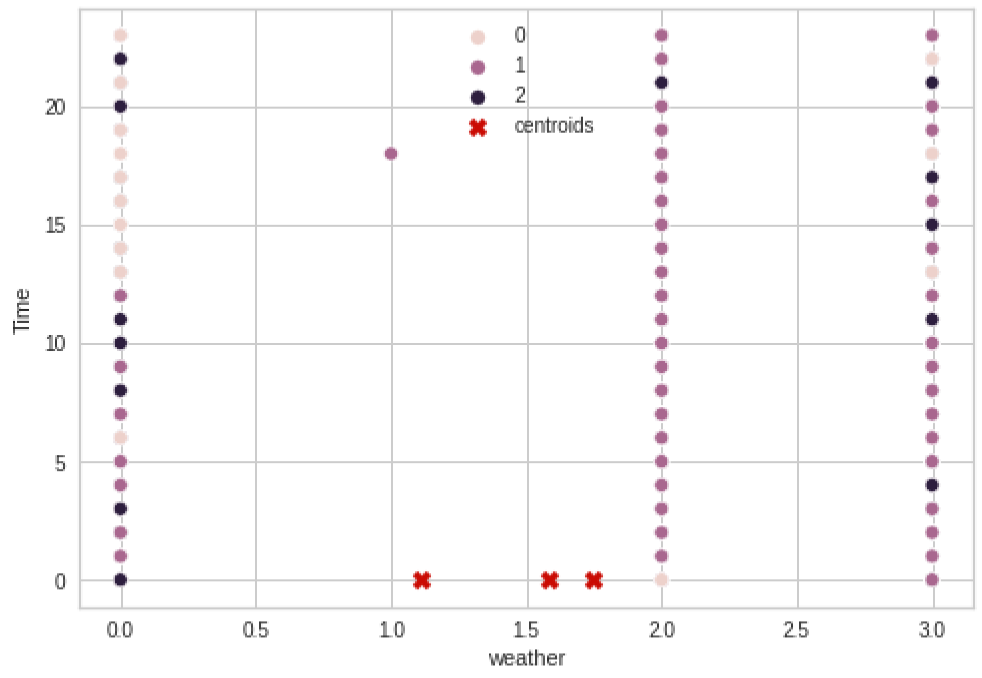
Now visualizing clusters to find ideal segments






0
1
2
centroids
temp
humidity
0
20
40
60
80
100
10
30

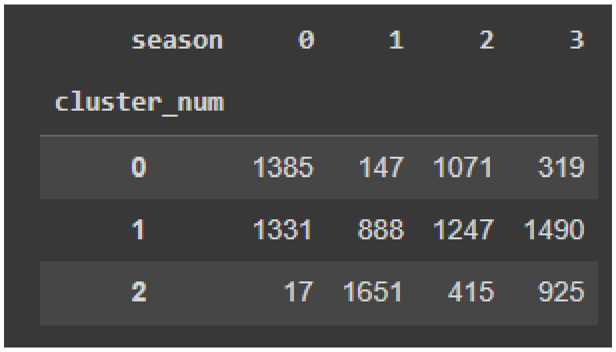

0
1
2
centroids
temp
season
0.0
0.5
1.0
1.5
2.0
2.5
3.0



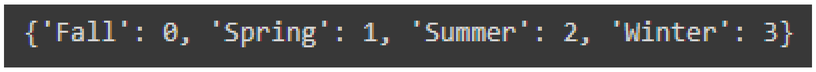
Describing Segments

Now the data is divided into 3 segments, each segment has some attributes. In order to understand the segments in more detail below are some crosstab and mosaic plots for different attributes.

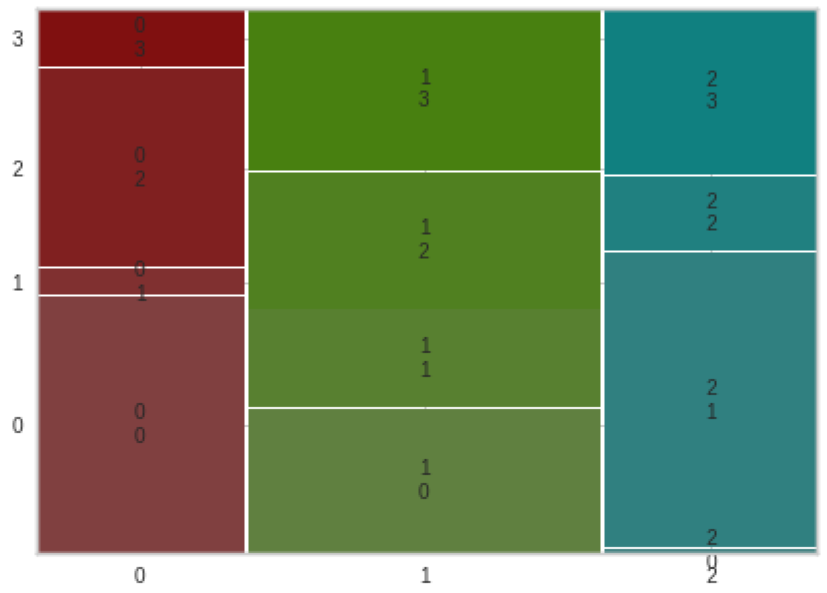
1. Season



season	0	1	2	3
cluster_num				
0	1385	147	1071	319
1	1331	888	1247	1490
2	17	1651	415	925

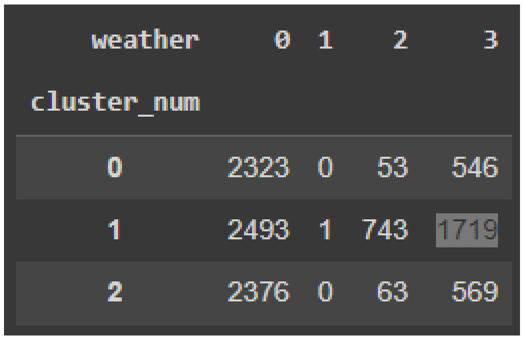


```
{'Fall': 0, 'Spring': 1, 'Summer': 2, 'Winter': 3}
```



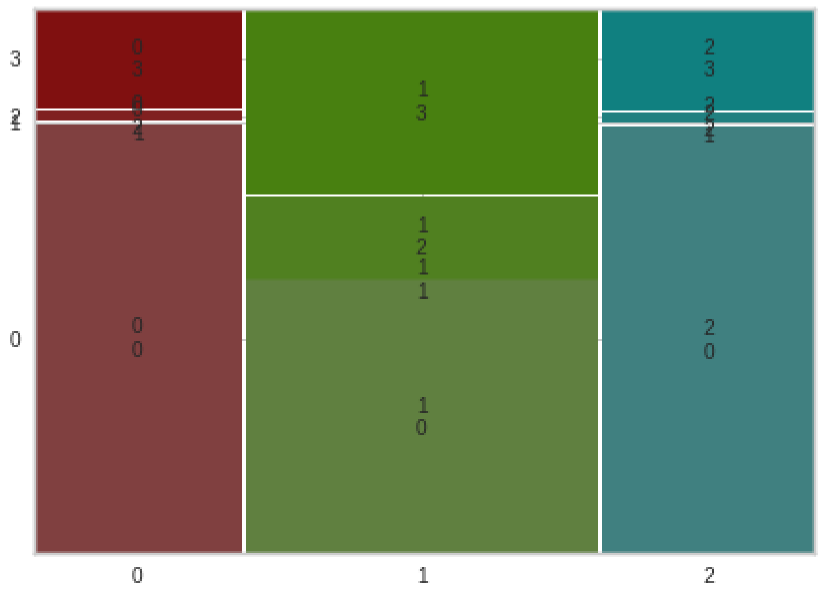
As you can see cluster 0 contains mostly Fall and summer seasons, while cluster 2 contains mostly spring season.

2. Weather



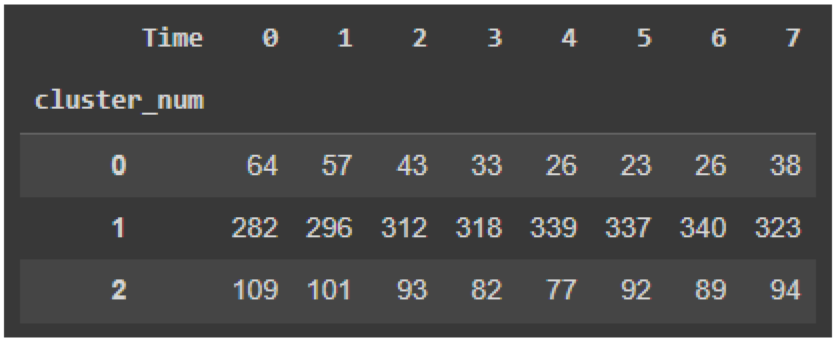
weather cluster_num	0	1	2	3
0	2323	0	53	546
1	2493	1	743	1719
2	2376	0	63	569

```
{' Clear + Few clouds': 0, ' Heavy Rain + Thunderstorm ': 1, ' Light Snow, Light Rain': 2, ' Mist + Cloudy ': 3}
```

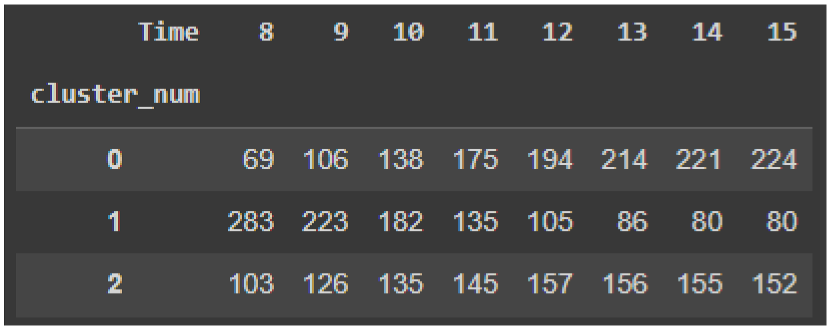


As you can see from the plots that segment 1 contains cases in which weather is bad, that is either light snow and rain or mist and cloudy.

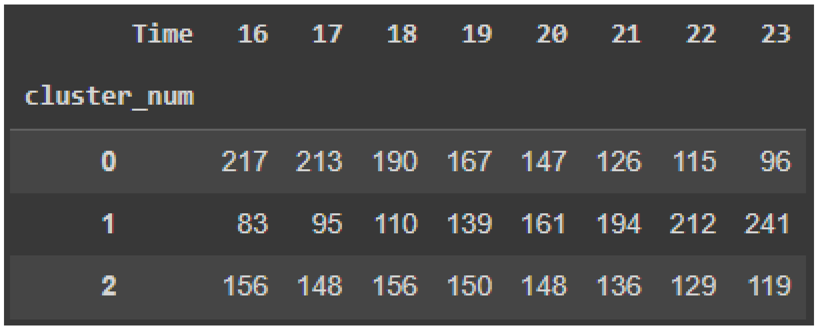
3. Time: Hours



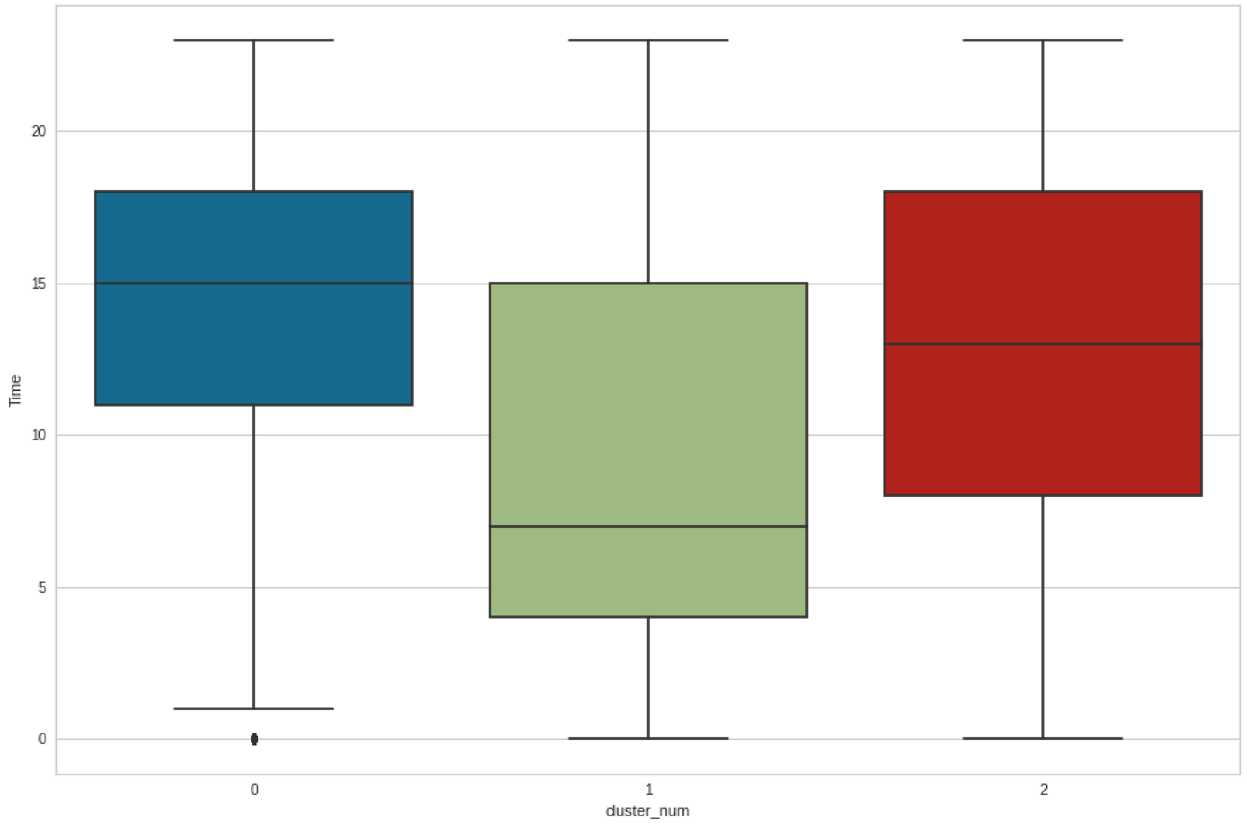
Time cluster_num	0	1	2	3	4	5	6	7
0	64	57	43	33	26	23	26	38
1	282	296	312	318	339	337	340	323
2	109	101	93	82	77	92	89	94



Time cluster_num	8	9	10	11	12	13	14	15
0	69	106	138	175	194	214	221	224
1	283	223	182	135	105	86	80	80
2	103	126	135	145	157	156	155	152



Time	16	17	18	19	20	21	22	23
cluster_num								
0	217	213	190	167	147	126	115	96
1	83	95	110	139	161	194	212	241
2	156	148	156	150	148	136	129	119



As we can see cluster 1 contains values during the night time from hours 22 in the night to 9 in the morning.

Target Segments

As seen from the above visualizations of weather against the clusters, segment 1 contains most values where weather is bad. It is also seen that during the night time from 10 pm to 9 am segment 1 contains most values.

So segment 1 could be a potential target segment, as most cab companies currently present in the market have very poor service in these two scenarios.

Customizing the Marketing Mix

The marketing mix refers to the set of actions, or tactics, that a company uses to promote its brand or product in the market. The 4Ps make up a typical marketing mix -

Price, Product, Promotion and Place.

a. Price: refers to the value that is put for a product. It depends on costs of production, segment targeted, ability of the market to pay, supply - demand and a host of other direct and indirect factors. There can be several types of pricing strategies, each tied in with an overall business plan
b. Product: refers to the item actually being sold. The product must deliver a minimum level of performance; otherwise even the best work on the other elements of the marketing mix won't do any good.
c. Place: refers to the point of sale. In every industry, catching the eye of the consumer and making it easy for her to buy it is the main aim of a good distribution or 'place' strategy. Retailers pay a premium for the right location. In fact, the mantra of a successful retail business is 'location, location, location’.
d. Promotion: this refers to all the activities undertaken to make the product or service known to the user and trade. This can include advertising, word of mouth, press reports, incentives, commissions and awards to the trade. It can also include consumer schemes, direct marketing, contests and prizes.

Potential Customer Base in the Early Markets

As per the market research we found out that segment 1 contains 4,956 out of 10,886 values present in the data set. Most of these customers are those who wanted to book a cab in a bad weather situation or during the night time. Let's assume even if half the customers in this segment belong to our target audience then it constitutes 25% of the whole data. Above we have discussed that monthly rides in India according to the data in 2018 are 50 million.

So 25% of 50 million is 12.5 million these are the rides which come in our market segment. Let's say we try to convert just 10% of these customers to our cab service then our potential target audience is 1.25 million rides a month.

Let's assume that each ride averages Rs. 100. So we can expect a monthly revenue of Rs. 125 million or 12.5 Crore. We know that it does include all the expenses.

Marketing Strategy

In order to convert these customers we need to adopt a proper marketing strategy. As we know our target audience are people in bad weather and night time. All cab services offer higher prices during these situations. We will provide discounted prices to the customers during the night and bad weather. This will lead to customers coming on our platform.

Github Link to the Project

https://github.com/skhan4784/Cab-Market-Segmentation