Car Sharing

June 6, 2020

1 Deutsche Bahn

1.1 Introduction

We are given 4 Datasets. The tables contain data on vehicles, rental stations, bookings, and tariff categories of **Car Sharing** business named **Deutsche Bahn** in Germany. The tables need to be cleaned, and data models have to be built to answer some questions and making predictions. Let's import packages we believe will help us in this task.

[138]:
```
import pandas as pd
from unidecode import unidecode
import itertools
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import matplotlib.ticker as ticker
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import jaccard_score
from sklearn import metrics
import statsmodels.api as sm
```

1.2 Loading, Cleaning and Displaying the Datasets

1.2.1 Booking

[2]:
```
#Since csv file columns are divided by a seperator, we must mention it.
Booking = pd.read_csv("OPENDATA_BOOKING_CARSHARING - Copy.csv", sep=';')

#Removing unwanted column.
Booking.drop('TECHNICAL_INCOME_CHANNEL', axis=1, inplace=True)

#Dropping NaN rows.
Booking.dropna(inplace=True)
```

```python
#Droppiing Duplicates.
Booking.drop_duplicates(inplace=True)

#Removing umlauts.
for col in Booking:
    if type(Booking[col][0])==str:
        Booking[col]=Booking[col].apply(unidecode)

#Converting DataFrames containing time from str to timestamps.
Booking['DATE_BOOKING'] = pd.to_datetime(Booking['DATE_BOOKING'])
Booking['DATE_UNTIL'] = pd.to_datetime(Booking['DATE_UNTIL'])
Booking['DATE_FROM'] = pd.to_datetime(Booking['DATE_FROM'])

#Displaying DataFrame.
Booking.head()
```

[2]:

```
   BOOKING_HAL_ID  CATEGORY_HAL_ID  VEHICLE_HAL_ID  \
0        17842196           100012          150359
1        18270895           100003          149335
2        19054992           100012          151333
3        19057626           100003          149540
4        19313282           100001          150574

                            CUSTOMER_HAL_ID        DATE_BOOKING  \
0  9680D41CFEFE292240253676FF6DD6C242B98EFD 2013-06-05 08:49:33
1  045B17DDFAA4DCE1751DF14B2DFC2C3106C5E788 2013-06-25 14:12:08
2  645B3B221397740C5DD3ACE9915B28D717697D1F 2013-08-01 07:20:47
3  00DF8A75463E3424010AF22F5292FB9499DBEFBD 2013-08-01 09:22:07
4  6551685BE2457EC2944877C65423089CDD6EA6C2 2013-08-13 10:28:38

            DATE_FROM          DATE_UNTIL COMPUTE_EXTRA_BOOKING_FEE  \
0 2014-01-12 13:00:00 2014-01-12 14:30:00                      Nein
1 2014-05-06 13:30:00 2014-05-06 19:00:00                      Nein
2 2014-06-14 14:00:00 2014-06-22 10:30:00                      Nein
3 2014-02-01 15:00:00 2014-02-08 15:00:00                      Nein
4 2014-05-16 14:45:00 2014-05-16 22:00:00                        Ja

  TRAVERSE_USE  DISTANCE     START_RENTAL_ZONE  START_RENTAL_ZONE_HAL_ID  \
0         Nein      14.0  Bernkasteler Strasse                    401768
1         Nein      84.0          ZOB Oldenburg                   400346
2         Nein    1036.0          Hbf Stralsund                    32961
3         Nein     681.0   Donnersbergerbrucke                    401104
4           Ja      60.0             Hbf Fulda                    404524

        END_RENTAL_ZONE  END_RENTAL_ZONE_HAL_ID RENTAL_ZONE_HAL_SRC  \
0  Bernkasteler Strasse                  401768             Station
```

```
1         ZOB Oldenburg                  400346            Station
2         Hbf Stralsund                   32961            Station
3  Donnersbergerbrucke                   401104            Station
4             Hbf Fulda                  404524            Station

   CITY_RENTAL_ZONE
0              Koln
1  Oldenburg (Oldb)
2         Stralsund
3           Munchen
4             Fulda
```

1.2.2 Vehicle

[3]:
```
#Since csv file columns are divided by a seperator, we must mention it.
Vehicle = pd.read_csv("OPENDATA_VEHICLE_CARSHARING - Copy.csv", sep = ';')

#Removing unneccasry columns.
Vehicle.
 ↪drop(['COMPANY','COMPANY_GROUP','ACCESS_CONTROL_COMPONENT_TYPE','SERIAL_NUMBER'],axis=1,inpla

#Dropping NaN rows.
Vehicle.dropna(inplace=True)

#Droppiing Duplicates.
Vehicle.drop_duplicates(inplace=True)

#Removing umlauts.
for col in Vehicle:
    if type(Vehicle[col][166])==str:
        Vehicle[col]=Vehicle[col].apply(unidecode)

#Displaying DataFrame.
Vehicle.head()
```

[3]:
```
   VEHICLE_HAL_ID VEHICLE_MODEL_TYPE VEHICLE_MANUFACTURER_NAME  \
0          143031               Auto                      Ford
2          147314               Auto                      Ford
3          147382               Auto                      Opel
6          147983               Auto                      Ford
7          147310               Auto                      Ford

  VEHICLE_MODEL_NAME                 VEHICLE_TYPE_NAME                VIN  \
0           Transit  2,2 Diesel 63kW !! kein Radio !!  WF0XXXBDFX8R74238
2             Focus             1,6 Diesel 80kW NAVI  WF0SXXGCDSAA82712
3             Astra             1,7 Diesel 81kW NAVI  W0L0AHL35B2057645
6            Fiesta            1,6 Diesel 70kW NAVI   WF0JXXGAJJBD83978
```

```
7              Focus              1,6 Diesel 80kW NAVI  WF0SXXGCDSAA82693

  REGISTRATION_PLATE  KW FUEL_TYPE_NAME OWNERSHIP_TYPE CAPACITY_AMOUNT
0           F-R 8018  63         Diesel  Langzeitmiete            60 l
2           F-R 8794  80         Diesel  Langzeitmiete            52 l
3           F-R 8829  81         Diesel  Langzeitmiete            52 l
6           F-R 8719  70         Diesel  Langzeitmiete            45 l
7           F-R 8758  80         Diesel  Langzeitmiete            52 l
```

1.2.3 Category

[4]:
```python
#Since csv file columns are divided by a seperator, we must mention it.
Category = pd.read_csv("OPENDATA_CATEGORY_CARSHARING - Copy.csv", sep = ';')

#Removing unneccasry columns.
Category.drop(['COMPANY','COMPANY_GROUP'],axis=1,inplace=True)

#Dropping NaN rows.
Category.dropna(inplace=True)

#Droppiing Duplicates.
Category.drop_duplicates(inplace=True)

#Removing umlauts.
for col in Category:
    if type(Category[col][0])==str:
        Category[col]=Category[col].apply(unidecode)

#Displaying DataFrame.
Category.head()
```

[4]:

	HAL_ID	CATEGORY
0	100000	Werbeklasse (mit Beklebung)
1	100001	Kleinklasse (teilweise ohne Navi)
2	100002	Mini (teilweise ohne Navi)
3	100003	Kompaktklasse
4	100004	Zubehor

1.2.4 Rental zone

[5]:
```python
#Since csv file columns are divided by a seperator, we must mention it.
Rental_zone = pd.read_csv("OPENDATA_RENTAL_ZONE_CARSHARING - Copy.csv", sep = ';
 ↪')

#Removing unneccasry columns.
Rental_zone.drop(['COMPANY','COMPANY_GROUP'],axis=1,inplace=True)
```

```
#Dropping NaN rows.
Rental_zone.dropna(inplace=True)

#Droppiing Duplicates.
Rental_zone.drop_duplicates(inplace=True)

#Removing umlauts.
for col in Rental_zone:
    if type(Rental_zone[col][0])==str:
        Rental_zone[col]=Rental_zone[col].apply(unidecode)

#Displaying DataFrame.
Rental_zone.head()
```

[5]:

```
   RENTAL_ZONE_HAL_ID RENTAL_ZONE_HAL_SRC               NAME  \
0                  38             Station    Paul-Lincke-Ufer
1                  79             Station          Ostbahnhof
2                 136             Station         Hbf Rostock
3                 138             Station        Hbf Schwerin
4                 171             Station  Hbf Aschaffenburg

               CODE          TYPE           CITY      COUNTRY  \
0               PLU   parkingarea         Berlin  Deutschland
1               OST  stationbased         Berlin  Deutschland
2       Hbf Rostock  stationbased        Rostock  Deutschland
3      Hbf Schwerin   parkingarea       Schwerin  Deutschland
4  Hbf Aschaffenburg  stationbased  Aschaffenburg  Deutschland

            LATITUDE           LONGITUDE POI_AIRPORT_X  \
0  52,491966685810670  13,437334746122360          Nein
1  52,509446616791216  13,433682918548584          Nein
2  54,077917752559770  12,132610380649567          Nein
3  53,633873801997470  11,406887769699097          Nein
4  49,981667892009890   9,144830703735351          Nein

  POI_LONG_DISTANCE_TRAINS_X POI_SUBURBAN_TRAINS_X POI_UNDERGROUND_X ACTIVE_X
0                       Nein                  Nein              Nein     Nein
1                         Ja                  Nein              Nein       Ja
2                         Ja                    Ja              Nein       Ja
3                         Ja                  Nein              Nein       Ja
4                         Ja                  Nein              Nein       Ja
```

Displaying Loss of Data:

[6]:

```
print('Loss of Data for Dataset Booking:', ((548073-547872)/548073)*100, '%')
print('Loss of Data for Dataset Vehicle:', ((1773-1678)/1773)*100, '%')
```

```
print('Loss of Data for Dataset Category:', 0, '%')
print('Loss of Data for Dataset Rental zone:', ((628-616)/628)*100, '%')
```

```
Loss of Data for Dataset Booking: 0.036673946718776516 %
Loss of Data for Dataset Vehicle: 5.3581500282007894 %
Loss of Data for Dataset Category: 0 %
Loss of Data for Dataset Rental zone: 1.910828025477707 %
```

1.3 Which factors influence the utilization of a given car?

To answer this question, We must define what utilization of car is. **We define utility of car to be the number of times it has been booked.** So a car that has been booked most number of times will have the highest utility. This value is our **Dependent variable**.

Now we must decide factors which our relevant to our question, or **Independent variables**. These variables will be selected from different datasets and put into one.

[7]:
```
#Creating a new dataframe.
car_df = pd.DataFrame()

#Adding Independent variables deemed important.
car_df['VEHICLE_HAL_ID'] = Vehicle['VEHICLE_HAL_ID']
car_df['CATEGORY_HAL_ID'] = Booking['CATEGORY_HAL_ID']
car_df['VEHICLE_MANUFACTURER_NAME'] = Vehicle['VEHICLE_MANUFACTURER_NAME']
car_df['VEHICLE_MODEL_NAME'] = Vehicle['VEHICLE_MODEL_NAME']
car_df['KW'] = Vehicle['KW']
car_df['FUEL_TYPE_NAME'] = Vehicle['FUEL_TYPE_NAME']
car_df['OWNERSHIP_TYPE'] = Vehicle['OWNERSHIP_TYPE']

#Counting the number of times a vehicle is booked.
counts_vehicle = pd.DataFrame(Booking['VEHICLE_HAL_ID'].value_counts().
 ↪reset_index())

#Calculating the total distance a vehicle has traveled.
distance=Booking.groupby('VEHICLE_HAL_ID',as_index=False)['DISTANCE'].sum()
counts_vehicle.columns = ['VEHICLE_HAL_ID', 'NUMBER_OF_TIMES_UTILIZED']

#Adding Number of times a vehicle is utilized and total distance traveled to␣
 ↪DataFrame.
merged_car = pd.merge(car_df, distance, on='VEHICLE_HAL_ID')
merged_car = pd.merge(merged_car, counts_vehicle, on='VEHICLE_HAL_ID')

#Displaying DataFrame.
merged_car.head()
```

[7]:

	VEHICLE_HAL_ID	CATEGORY_HAL_ID	VEHICLE_MANUFACTURER_NAME	\
0	143031	100012	Ford	
1	147314	100012	Ford	

```
2          147382          100003                          Opel
3          147983          100007                          Ford
4          147310          100006                          Ford

  VEHICLE_MODEL_NAME  KW FUEL_TYPE_NAME OWNERSHIP_TYPE  DISTANCE  \
0            Transit  63         Diesel  Langzeitmiete     112.0
1              Focus  80         Diesel  Langzeitmiete    3701.0
2              Astra  81         Diesel  Langzeitmiete    4108.0
3             Fiesta  70         Diesel  Langzeitmiete    2217.0
4              Focus  80         Diesel  Langzeitmiete    5074.0

   NUMBER_OF_TIMES_UTILIZED
0                         5
1                        53
2                        37
3                        39
4                        32
```

Our assessment shows there are some tarrifs in our column **CATEGORY_HAL_ID** for which no information is provided in dataframe **Category**. These tarriffs will need to be removed. First, Let's confirm our assessment

[89]:
```
#Making copies of our dataframes, They can be manipulated if needed without␣
↪affecting the original.
booking = Booking.copy()
vehicle = Vehicle.copy()
category = Category.copy()
rental_zone = Rental_zone.copy()

#Confirming if there are any values which are not present in DataFrame Category.
for item in booking['CATEGORY_HAL_ID'].unique():
    if item not in category['HAL_ID'].values:
        # printing values which have no information.
        print(item)
```

```
1000
150010
43
56
1005
1002
44
1001
34501
24
150003
151062
27
```

```
2199
45
558
559
50
34500
536
1557
215
47
538
539
569
1003
400000
560
550027
568
330002
1004
801003
589
554709
588
205500
1587
205200
207
520001
```

Let's now remove these values from our Dataframe.

[9]:
```python
for item in range(len(merged_car['CATEGORY_HAL_ID'])):
    if merged_car['CATEGORY_HAL_ID'][item] not in category['HAL_ID'].values:
        merged_car.drop(item,inplace=True)

#Displaying remaning number of rows.
len(merged_car['CATEGORY_HAL_ID'])
```

[9]:
```
1642
```

1.3.1 Multiple Linear Regression

To capture our dependent variable, we can use Multiple Linear Regression. Multiple Linear Regression is very similar to Simple Linear Regression, but this method is used to explain the relationship between one continuous response (dependent) variable and two or more predictor (independent) variables.

$$\begin{aligned}
Y &: Response\ Variable \\
X_1 &: Predictor\ Variable\ 1 \\
X_2 &: Predictor\ Variable\ 2 \\
X_3 &: Predictor\ Variable\ 3 \\
X_4 &: Predictor\ Variable\ 4
\end{aligned}$$

$$\begin{aligned}
a &: intercept \\
b_1 &: coefficients\ of\ Variable\ 1 \\
b_2 &: coefficients\ of\ Variable\ 2 \\
b_3 &: coefficients\ of\ Variable\ 3 \\
b_4 &: coefficients\ of\ Variable\ 4
\end{aligned}$$

The equation is given by:

$$Yhat = a + b_1X_1 + b_2X_2 + b_3X_3 + b_4X_4$$

Since we have made a dataframe consisting of data we find relevant, let's prepare it for regression. In particular, we need to make **dummy variables** out of categorical columns.

[65]:
```
#Creating a copy of our dataframe
merged_car_1 = merged_car.copy()

#Changing column CATEGORY_HAL_ID to str as it is categorical data.
merged_car_1['CATEGORY_HAL_ID'] =  merged_car_1['CATEGORY_HAL_ID'].astype(float).
 ↪astype(str)

#Getting Dummy variables for Categorical Data.
Dummy_Car_Hal = pd.get_dummies(merged_car['CATEGORY_HAL_ID'])
Dummy_Manu = pd.get_dummies(merged_car['VEHICLE_MANUFACTURER_NAME'])
Dummy_Vehicle_Model = pd.get_dummies(merged_car['VEHICLE_MODEL_NAME'])
Dummy_Fuel = pd.get_dummies(merged_car['FUEL_TYPE_NAME'])
Dummy_Ownership = pd.get_dummies(merged_car['OWNERSHIP_TYPE'])

#Dependent Variable.
Dep_variable = merged_car_1['NUMBER_OF_TIMES_UTILIZED']

#Removing unnceccary columns.
merged_car_1.drop(['VEHICLE_HAL_ID','CATEGORY_HAL_ID',\
                   'VEHICLE_MANUFACTURER_NAME','VEHICLE_MODEL_NAME',\
```

```
↪'FUEL_TYPE_NAME','OWNERSHIP_TYPE','NUMBER_OF_TIMES_UTILIZED'], axis=1,␣
↪inplace=True)

#Joining Dummy Dataframes with merged_car_1.
merged_car_1=merged_car_1.join(Dummy_Car_Hal)
merged_car_1=merged_car_1.join(Dummy_Manu)
merged_car_1=merged_car_1.join(Dummy_Vehicle_Model)
merged_car_1=merged_car_1.join(Dummy_Fuel)
merged_car_1=merged_car_1.join(Dummy_Ownership)

#Displaying final result.
merged_car_1.head()
```

[65]:

```
   KW  DISTANCE  100001  100002  100003  100005  100006  100007  100009  \
0  63     112.0       0       0       0       0       0       0       0
1  80    3701.0       0       0       0       0       0       0       0
2  81    4108.0       0       0       1       0       0       0       0
3  70    2217.0       0       0       0       0       0       1       0
4  80    5074.0       0       0       0       0       1       0       0

   100010  ...  Transit  Transit Custom  Vito  Vivaro  Diesel  \
0       0  ...        1               0     0       0       1
1       0  ...        0               0     0       0       1
2       0  ...        0               0     0       0       1
3       0  ...        0               0     0       0       1
4       0  ...        0               0     0       0       1

   Plug In (Strom, Super)  Super (Benzin)  Super E10  Kauf  Langzeitmiete
0                       0               0          0     0              1
1                       0               0          0     0              1
2                       0               0          0     0              1
3                       0               0          0     0              1
4                       0               0          0     0              1

[5 rows x 50 columns]
```

Now that we have created the dataframe, Let's split our data into training and test datasets. It will help us check our model.

[66]:

```
msk = np.random.rand(len(merged_car_1)) < 0.8
train_x = merged_car_1[msk]
test_x= merged_car_1[~msk]
train_y = Dep_variable[msk]
test_y = Dep_variable[~msk]
```

Applying Regression:

[67]:
```
lm = LinearRegression()
lm.fit(train_x, train_y)
```

[67]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Let's have a look at our coefficients:

[68]:
```
list(zip(lm.coef_, merged_car_1.columns))
```

[68]:
```
[(-0.5183348851283659, 'KW'),
 (0.009450836122494796, 'DISTANCE'),
 (-3.320751488605964, 100001),
 (1.7367212329226476, 100002),
 (9.586142736126643, 100003),
 (-39.43000712659267, 100005),
 (77.72087965952147, 100006),
 (-67.38101655128075, 100007),
 (0.14659845641890001, 100009),
 (6.284933882904216, 100010),
 (6.798432969872751, 100012),
 (30.98214868339187, 100013),
 (-23.124082454680092, 100014),
 (36.37409637145536, 'Citroen'),
 (-48.69446405378241, 'Fiat'),
 (-68.88727578286137, 'Ford'),
 (-86.6777135334124, 'Mercedes'),
 (-27.716111360199484, 'Opel'),
 (30.047967998714427, 'Peugeot'),
 (0.0, 'Renault'),
 (132.48837249987469, 'Toyota'),
 (33.06512786021116, 'VW'),
 (-121.8998247779714, '107'),
 (-48.69446405378259, '500'),
 (-17.82516694795913, 'Ampera'),
 (-152.1467000263844, 'Astra'),
 (132.48837249987463, 'Aygo'),
 (151.94779277668664, 'Boxer'),
 (-132.9427332489726, 'C-Klasse'),
 (-91.90147373006393, 'Caddy'),
 (-52.582669216300076, 'Corsa'),
 (36.37409637145516, 'DS3'),
 (5.421953916540744, 'Fiesta'),
 (-159.71750453386585, 'Focus'),
 (-92.96472957890849, 'Insignia'),
 (41.59564584524881, 'Ka'),
 (0.0, 'Megane'),
 (-91.37658325416577, 'Mondeo'),
```

```
(-15.24136825982498, 'Sprinter'),
(124.96660159027552, 'T5'),
(-93.62502682826491, 'Transit'),
(228.8142390716451, 'Transit Custom'),
(61.50638797538524, 'Vito'),
(287.8031544093534, 'Vivaro'),
(41.747402069557296, 'Diesel'),
(-17.82516694795908, 'Plug In (Strom, Super)'),
(-14.957010630801289, 'Super (Benzin)'),
(-8.965224490796537, 'Super E10'),
(64.56553164002484, 'Kauf'),
(-64.5655316400248, 'Langzeitmiete')]
```

Let's check our model's accuracy:

[69]:
```python
#Predicting Dependent variable based on test set.
Yhat = lm.predict(test_x)

#Checking Errors.
print('Mean Absolute Error:', metrics.mean_absolute_error(test_y, Yhat))
print('Mean Squared Error:', metrics.mean_squared_error(test_y, Yhat))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(test_y,␣
↪Yhat)))
```

```
Mean Absolute Error: 78.04534027468846
Mean Squared Error: 11250.761073158053
Root Mean Squared Error: 106.06960485057938
```

Conclustion:

We believe this is a decent model. Looking at the coefficients, among tarrifs, IDs **100006** and **100007** are particulary interesting. Having former results in a higher utility in a significant way and vice versa with later. Among Vehicle manufacturers and vehicle model names, **Agyo**, **Citroen**, **Boxer**, **DS3** and **Vivaro** results in a higher utility. Having a vehicle which uses **Diesel** results in higher uitlity. Last, but not the least, **Kauf**, which means purchase in English, results in a higher utility and vice versa with **Langzeitmiete**, which is Long term rent.

1.3.2 Which types of cars have the best/worst utilization?

Let's observe Vehicles with best utilization:

[15]:
```python
best = merged_car[['VEHICLE_HAL_ID','VEHICLE_MANUFACTURER_NAME',␣
↪'VEHICLE_MODEL_NAME','NUMBER_OF_TIMES_UTILIZED']]
best.sort_values(by=['NUMBER_OF_TIMES_UTILIZED'], ascending = False).head()
```

[15]:
```
     VEHICLE_HAL_ID VEHICLE_MANUFACTURER_NAME VEHICLE_MODEL_NAME  \
599          156602                    Toyota               Aygo
947          157331                   Citroen                DS3
788          157330                   Citroen                DS3
```

```
890           156777                   Citroen              DS3
1077          160280                  Mercedes             Vito

      NUMBER_OF_TIMES_UTILIZED
599                       1483
947                       1423
788                       1361
890                       1337
1077                      1330
```

Let's observe Vehicles with worst utilization:

[16]:
```
best.sort_values(by=['NUMBER_OF_TIMES_UTILIZED']).head()
```

[16]:
```
      VEHICLE_HAL_ID VEHICLE_MANUFACTURER_NAME VEHICLE_MODEL_NAME  \
24            147966                      Ford             Fiesta
1225          181564                      Opel              Astra
78            147963                      Ford             Fiesta
1039          171766                      Ford             Fiesta
933           159901                      Opel              Corsa

      NUMBER_OF_TIMES_UTILIZED
24                           1
1225                         2
78                           3
1039                         4
933                          4
```

1.3.3 How could this information be used to improve the overall utilization?

Much of factors deemed important in our analysis are left up to the choice of consumers, meaning the type of car they need. However several steps may be taken that might increase overall utilization of vehicles. we will list them below:

1. Having Vehicles availabe for purchase and not long term rent might increase utilization.
2. Having more Vehicles which use Diesel might increase utilization.
3. Having more Vehicles bearing names like **DS3** or **Agyo** might increase overall utilization.

1.4 Is it possible to categorize / cluster the different stations?

We must decide how we are going to categorize rental stations. One way is to categorize them according to the number of times a booking was started at a rental station. Let's assess this information.

[17]:
```
Booking['START_RENTAL_ZONE_HAL_ID'].value_counts().describe()
```

[17]:
```
count       424.000000
mean       1292.150943
```

```
std        1602.482590
min           1.000000
25%         342.750000
50%         864.000000
75%        1693.750000
max       12824.000000
Name: START_RENTAL_ZONE_HAL_ID, dtype: float64
```

It can be seen that a rental station selected at random will have **1292** bookings approximately. However, There is a lot of variablity in this assumption as evident by a very high standard deviation. We will use percentiles to divide counts of bookings at a rental station into **5** categories:

1. Rental stations bookings below or equal to **25th** percentile will be categorized as **Very Low**.
2. Rental stations bookings above **25th** percentile but equal to or below **50th** percentile will be categorized as **Low**.
3. Rental stations bookings above **50th** percentile but equal to or below **75th** percentile will be categorized as **Medium**.
4. Rental stations bookings above **75th** percentile but equal to or below **90th** percentile will be categorized as **High**.
5. Rental stations bookings above **90th** percentile will be categorized as **Very High**.

We will also include data we think is important in differentiating these categories.

[90]:
```
#Dropping unnecessary column from copy of Rental zone we made earlier.
rental_zone.
 ↪drop(['COUNTRY','RENTAL_ZONE_HAL_SRC','LATITUDE','LONGITUDE','CODE','NAME'],␣
 ↪axis=1, inplace=True)

#Counting the number of times a vehicle is booked.
counts_booking = pd.DataFrame(Booking['START_RENTAL_ZONE_HAL_ID'].value_counts().
 ↪reset_index())
counts_booking.columns = ['RENTAL_ZONE_HAL_ID', 'NUMBER_OF_BOOKINGS']

#Merging Datasets.
merged_rental_zone = pd.merge(rental_zone, counts_booking,␣
 ↪on='RENTAL_ZONE_HAL_ID')

#Setting boundaries for different categories.
Very_low = np.percentile(merged_rental_zone['NUMBER_OF_BOOKINGS'], 25)
Medium = np.percentile(merged_rental_zone['NUMBER_OF_BOOKINGS'], 50)
High = np.percentile(merged_rental_zone['NUMBER_OF_BOOKINGS'], 75)
Very_high = np.percentile(merged_rental_zone['NUMBER_OF_BOOKINGS'], 90)

# Adding an empty column Category.
merged_rental_zone['CATEGORY'] = np.nan

# Adding values to column Category.
for item in range(len(merged_rental_zone['RENTAL_ZONE_HAL_ID'])):
```

```
    if merged_rental_zone['NUMBER_OF_BOOKINGS'][item] <= Very_low:
        merged_rental_zone.loc[[item],'CATEGORY'] = 'Very Low'
    elif (merged_rental_zone['NUMBER_OF_BOOKINGS'][item] > Very_low) and\
    (merged_rental_zone['NUMBER_OF_BOOKINGS'][item] <= Medium) :
        merged_rental_zone.loc[[item],'CATEGORY'] = 'Low'
    elif (merged_rental_zone['NUMBER_OF_BOOKINGS'][item] > Medium) and\
    (merged_rental_zone['NUMBER_OF_BOOKINGS'][item] <= High) :
        merged_rental_zone.loc[[item],'CATEGORY'] = 'Medium'
    elif (merged_rental_zone['NUMBER_OF_BOOKINGS'][item] > High) and\
    (merged_rental_zone['NUMBER_OF_BOOKINGS'][item] <= Very_high) :
        merged_rental_zone.loc[[item],'CATEGORY'] = 'High'
    else:
        merged_rental_zone.loc[[item],'CATEGORY'] = 'Very High'

#Removing column Number of Bookings.
merged_rental_zone.drop('NUMBER_OF_BOOKINGS',axis=1, inplace=True)

#Displaying Rental zone station IDs with respective categories.
merged_rental_zone[['RENTAL_ZONE_HAL_ID', 'CATEGORY']]
```

[90]:

```
     RENTAL_ZONE_HAL_ID   CATEGORY
0                    38     Medium
1                    79  Very High
2                   136     Medium
3                   138       High
4                   171  Very High
..                  ...        ...
390              406229   Very Low
391              406256   Very Low
392              406277   Very Low
393              406327   Very Low
394              406429   Very Low

[395 rows x 2 columns]
```

1.4.1 Which are the differentiating factors?

We already have factors we think are important in determining our Rental sation's category. Let's give a numeric values to our categories have a look:

[91]:

```
#Giving numeric values to categories.
number = LabelEncoder()
merged_rental_zone['CATEGORY'] = number.
 ↪fit_transform(merged_rental_zone['CATEGORY'])

merged_rental_zone.head()
```

[91]:
```
   RENTAL_ZONE_HAL_ID         TYPE          CITY POI_AIRPORT_X  \
0                  38   parkingarea        Berlin          Nein
1                  79  stationbased        Berlin          Nein
2                 136  stationbased       Rostock          Nein
3                 138   parkingarea      Schwerin          Nein
4                 171  stationbased  Aschaffenburg          Nein

  POI_LONG_DISTANCE_TRAINS_X POI_SUBURBAN_TRAINS_X POI_UNDERGROUND_X ACTIVE_X  \
0                       Nein                  Nein              Nein     Nein
1                         Ja                  Nein              Nein       Ja
2                         Ja                    Ja              Nein       Ja
3                         Ja                  Nein              Nein       Ja
4                         Ja                  Nein              Nein       Ja

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

[92]:
```
merged_rental_copy = merged_rental_zone.copy()

#Dummy_Name = pd.get_dummies(merged_rental_zone['NAME'])
Dummy_Type = pd.get_dummies(merged_rental_zone['TYPE'])
Dummy_City = pd.get_dummies(merged_rental_zone['CITY'])
Dummy_Near_Airport = pd.get_dummies(merged_rental_zone['POI_AIRPORT_X'])
Dummy_Near_Train = pd.
 ↪get_dummies(merged_rental_zone['POI_LONG_DISTANCE_TRAINS_X'])
Dummy_Near_S_Train = pd.get_dummies(merged_rental_zone['POI_SUBURBAN_TRAINS_X'])
Dummy_Underground = pd.get_dummies(merged_rental_zone['POI_UNDERGROUND_X'])
Dummy_Active = pd.get_dummies(merged_rental_zone['ACTIVE_X'])

Dep_variable = merged_rental_zone['CATEGORY']

#Removing unnceccary columns.
merged_rental_copy.drop(['RENTAL_ZONE_HAL_ID','TYPE','CITY',\
                       ␣
 ↪'POI_AIRPORT_X','POI_LONG_DISTANCE_TRAINS_X','POI_SUBURBAN_TRAINS_X',\
                        'POI_UNDERGROUND_X', 'CATEGORY','ACTIVE_X'], axis=1,␣
 ↪inplace=True)

#merged_rental_copy=merged_rental_copy.join(Dummy_Name)
merged_rental_copy=merged_rental_copy.join(Dummy_Type)
merged_rental_copy=merged_rental_copy.join(Dummy_City, rsuffix = "_")
merged_rental_copy=merged_rental_copy.join(Dummy_Near_Airport, rsuffix = "_")
merged_rental_copy=merged_rental_copy.join(Dummy_Near_Train, rsuffix = "-")
```

```
merged_rental_copy=merged_rental_copy.join(Dummy_Near_S_Train, rsuffix = "&")
merged_rental_copy=merged_rental_copy.join(Dummy_Underground, rsuffix = "*")
merged_rental_copy=merged_rental_copy.join(Dummy_Active, rsuffix = "^")

merged_rental_copy
```

[92]:

```
     freefloating  parkingarea  stationbased  Aachen  Aschaffenburg  Bayreuth  \
0               0            1             0       0              0         0
1               0            0             1       0              0         0
2               0            0             1       0              0         0
3               0            1             0       0              0         0
4               0            0             1       0              1         0
..            ...          ...           ...     ...            ...       ...
390             0            1             0       0              0         0
391             0            1             0       0              0         0
392             0            0             1       0              0         0
393             0            1             0       0              0         0
394             0            0             1       0              0         0

     Berlin  Bielefeld  Bietigheim-Bissingen  Cottbus  ...  Ja  Nein  Ja-  \
0         1          0                     0        0  ...   0     1    0
1         1          0                     0        0  ...   0     1    1
2         0          0                     0        0  ...   0     1    1
3         0          0                     0        0  ...   0     1    1
4         0          0                     0        0  ...   0     1    1
..      ...        ...                   ...      ...  ...  ..   ...  ...
390       1          0                     0        0  ...   0     1    0
391       1          0                     0        0  ...   0     1    0
392       0          0                     0        0  ...   0     1    0
393       1          0                     0        0  ...   0     1    0
394       0          0                     0        0  ...   0     1    0

     Nein-  Ja&  Nein&  Ja*  Nein*  Ja^  Nein^
0        1    0      1    0      1    0      1
1        0    0      1    0      1    1      0
2        0    1      0    0      1    1      0
3        0    0      1    0      1    1      0
4        0    0      1    0      1    1      0
..     ...  ...    ...  ...    ...  ...    ...
390      1    0      1    0      1    1      0
391      1    0      1    0      1    1      0
392      1    0      1    0      1    1      0
393      1    0      1    0      1    1      0
394      1    0      1    0      1    1      0

[395 rows x 85 columns]
```

Let's normalize our Independent variables.

[144]:
```
#merged_rental_display = merged_rental_copy.copy()
X = preprocessing.StandardScaler().fit(merged_rental_copy).
 ↪transform(merged_rental_copy)
X[0]
```

[144]:
```
array([-0.05037927,  1.90972742, -1.89552717, -0.05037927, -0.05037927,
       -0.05037927,  1.58958665, -0.05037927, -0.05037927, -0.05037927,
       -0.05037927, -0.05037927, -0.05037927, -0.07133764, -0.05037927,
       -0.05037927, -0.07133764, -0.05037927, -0.05037927, -0.16116459,
       -0.05037927, -0.13431767, -0.05037927, -0.12419406, -0.05037927,
       -0.1132277 , -0.05037927, -0.05037927, -0.07133764, -0.05037927,
       -0.07133764, -0.05037927, -0.05037927, -0.07133764, -0.05037927,
       -0.05037927, -0.05037927, -0.08748178, -0.05037927, -0.47213369,
       -0.07133764, -0.05037927, -0.07133764, -0.10114435, -0.05037927,
       -0.05037927, -0.08748178, -0.05037927, -0.05037927, -0.05037927,
       -0.05037927, -0.4148576 , -0.05037927, -0.05037927, -0.05037927,
       -0.05037927, -0.05037927, -0.05037927, -0.05037927, -0.05037927,
       -0.05037927, -0.05037927, -0.15269598, -0.07133764, -0.05037927,
       -0.05037927, -0.07133764, -0.07133764, -0.05037927, -0.31666789,
       -0.05037927, -0.05037927, -0.05037927, -0.05037927, -0.05037927,
       -0.05037927,  0.05037927, -0.37192544,  0.37192544, -0.32625539,
        0.32625539, -0.28669109,  0.28669109, -1.15640741,  1.15640741])
```

Train/Test dataset

Okay, we split our dataset into train and test set:

[132]:
```
msk = np.random.rand(len(merged_rental_copy)) < 0.8
train_x = X[msk]
test_x= X[~msk]
train_y = Dep_variable[msk]
test_y = Dep_variable[~msk]
```

Modeling and displaying coefficients

[145]:
```
#Logistic regression Model.
LR = LogisticRegression(multi_class='multinomial', solver='newton-cg').
 ↪fit(train_x,train_y)

#Displaying coefficients.
list(zip(LR.coef_[1], merged_rental_copy.columns))
```

[145]:
```
[(0.28482295659585277, 'freefloating'),
 (-0.08114154787053299, 'parkingarea'),
 (0.046117599743733556, 'stationbased'),
 (-0.03077486532595244, 'Aachen'),
```

```
(2.0223822712452932e-07, 'Aschaffenburg'),
(-0.06503159862797753, 'Bayreuth'),
(-0.02044722327390027, 'Berlin'),
(-0.03077486532595244, 'Bielefeld'),
(0.23193123410675365, 'Bietigheim-Bissingen'),
(0.2747027700322114, 'Cottbus'),
(-0.10070846341731561, 'Deggenhausertal'),
(-0.09654445683213114, 'Dillingen'),
(-0.029381691651599373, 'Duisburg'),
(-0.07226826191218544, 'Dusseldorf'),
(2.0223822712452932e-07, 'Eisenach'),
(0.17288759183252767, 'Erfurt'),
(-0.13640742198760736, 'Eriskirch'),
(-0.029381691651599394, 'Essen'),
(-0.04208008688605347, 'Flugh. Berlin'),
(-0.026060745614853442, 'Frankfurt am Main'),
(-0.029381691651599394, 'Freiburg'),
(-0.21410864439583174, 'Friedrichshafen'),
(-0.06036944806937729, 'Fulda'),
(-0.13856644882277766, 'Garmisch-Partenkirchen'),
(2.0223822712452932e-07, 'Gelsenkirchen'),
(-0.20034981228709997, 'Halle'),
(2.0223822712452932e-07, 'Hamburg'),
(-0.03801506486746206, 'Hamm'),
(-0.07664834628083424, 'Hannover'),
(2.0223822712452932e-07, 'Heidelberg'),
(0.23789763970187958, 'Heilbronn'),
(-0.09654445683213116, 'Heusweiler'),
(0.23193123410674774, 'Hildesheim'),
(-0.09188272007486098, 'Homburg / Saar'),
(-0.03077486532595242, 'Ingolstadt'),
(2.0223822712452932e-07, 'Jena'),
(-0.03801506486746182, 'Kaiserslautern'),
(-0.06518749704954038, 'Karlsruhe'),
(-0.09654445683213107, 'Kleinblittersdorf'),
(-0.21949342107668218, 'Koln'),
(0.2942833756147962, 'Krefeld'),
(-0.09654445683213118, 'Losheim am See'),
(0.17830392164848566, 'Ludwigsburg'),
(-0.2176118309405411, 'Mannheim'),
(2.0223822712452932e-07, 'Markdorf'),
(2.0223822712452932e-07, 'Meckenbeuren'),
(-0.11242756623636582, 'Meschede'),
(-0.0965444568321312, 'Mettlach'),
(0.23193123410674735, 'Minden'),
(0.2319312341067533, 'Monchengladbach'),
(2.0223822712452932e-07, 'Mulheim / Ruhr'),
```

```
(0.4754851705881098, 'Munchen'),
(-0.029381691651599446, 'Munster'),
(-0.02174804807119077, 'Oberhausen'),
(-0.030774865325952423, 'Offenburg'),
(-0.03077486532595242, 'Osnabruck'),
(-0.09654445683213105, 'Ottweiler'),
(0.18013831978501274, 'Panketal'),
(-0.01010223299659515, 'Potsdam'),
(0.2747027700322113, 'Ravensburg'),
(0.27470277003221105, 'Rosenheim'),
(-0.021748048071190695, 'Rostock'),
(-0.26339261442538964, 'Saarbrucken'),
(-0.13640742198760747, 'Saarlouis'),
(-0.09654445683213117, 'Salem'),
(2.0223822712452932e-07, 'Schwerin'),
(-0.10625748530378801, 'Siegburg'),
(-0.09396926160021674, 'St. Ingbert'),
(2.0223822712452932e-07, 'St. Wendel'),
(0.15195352939241505, 'Stuttgart'),
(-0.030774865325952267, 'Trier'),
(-0.05738532690340136, 'Troisdorf'),
(-0.02938169165159947, 'Ulm'),
(2.0223822712452932e-07, 'Weimar'),
(-0.038015064867461756, 'Wuppertal'),
(-0.04208008688605347, 'Ja'),
(0.042080086886053435, 'Nein'),
(-0.10223461353000336, 'Ja-'),
(0.10223461353000338, 'Nein-'),
(-0.09010728763379679, 'Ja&'),
(0.09010728763379765, 'Nein&'),
(0.09042798934479206, 'Ja*'),
(-0.09042798934479221, 'Nein*'),
(-0.4795825252210854, 'Ja^'),
(0.4795825252210806, 'Nein^')]
```

Let's make a prediction using our test set.

```
[134]: Yhat = LR.predict(test_x)
       Yhat
```

```
[134]: array([3, 0, 3, 3, 3, 3, 0, 3, 3, 3, 1, 1, 3, 1, 1, 1, 2, 0, 1, 4, 2, 2,
              2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 3, 0, 2, 0, 1, 1,
              2, 2, 2, 2, 2, 2, 2, 1, 0, 1, 4, 1, 4, 1, 1, 1, 1, 1, 1, 2, 4, 4,
              2, 4, 4, 4, 2, 2, 2, 2, 1, 4, 0, 0, 2, 2, 2, 2, 1, 4, 4, 0])
```

Evaluation

Lets try **accuracy score** for accuracy evaluation. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

[135]:
```
metrics.accuracy_score(test_y, Yhat)
```

[135]:
```
0.5
```

It seems our model is able make an accurate prediction **50%** of the time. There could reasons why we couldn't get a better score like confounding variables or outliers.

Judging from coefficients, it seems whether a station is **active** or not is the most important factor.The **type** of station is the second most important factor. Whether a station is near a **train station** appears to be next most important factor. These 3 factors have combined coefficients of **0.9** so they are most relevent when making our categories.

1.4.2 How could this information be used in Flinkster's operations?

Looking at our 3 most important factors, we can take points:

1. Whether a station is active or not is most important, maybe keeping it active gives it a higher chance of being categorized as **Very High.**
2. When purcashing or buliding a new Rental station, maybe keeping the type **free floating** gives it a higher chance of being categorized as **Very High.**
3. When purcashing or buliding a new Rental station, maybe keeping it near a **train station** higher chance of being categorized as **Very High.**