DURATIONS PREDICTION FOR SLZ AT VIC (SPAIN)

Research Questions

1. What are the significant factors that policy makers should consider when defining parking durations?
2. How do operational and location factors explain durations prediction?
3. How can LZ be effectly managed based on parking durations understanding and forecast?
4. From a modelling perspective, how should supply and demand perspectives complement to a better use of the curbside?

Libraries Installation

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import wilcoxon
%matplotlib inline
!python --version
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
import statsmodels.formula.api as sm
import os
import warnings
warnings.filterwarnings('ignore')
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
from patsy import dmatrices
import statsmodels.api as sm
!pip install fitter
from fitter import Fitter, get_common_distributions, get_distributions

print('Pandas', pd.__version__)
print('NumPy', np.__version__)
print('Matplotlib', matplotlib.__version__)
from pprint import pprint

!pip install hillmaker

import hillmaker as hm

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import decomposition, datasets
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
```

In [2]:

```
conda list
```

```
# packages in environment at /Users/juanpabl/opt/anaconda3:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0                    py38_0
_py-xgboost-mutex         2.0                       cpu_0    conda-forge
alabaster                 0.7.12             pyhd3eb1b0_0
altair                    4.2.0                    pypi_0    pypi
anaconda                  2021.05                  py38_0
anaconda-client           1.7.2                    py38_0
anaconda-navigator        2.0.3                    py38_0
anaconda-project          0.9.1              pyhd3eb1b0_1
ansi2html                 1.7.0                    pypi_0    pypi
anyio                     2.2.0            py38hecd8cb5_1
appdirs                   1.4.4                      py_0
applaunchservices         0.2.1                      py_0
appnope                   0.1.2           py38hecd8cb5_1001
appscript                 1.1.2            py38h9ed2024_0
argh                      0.26.2                   py38_0
argon2-cffi               20.1.0           py38h9ed2024_1
```

In [3]:

```
operations_2018 = pd.read_excel('operations_2018.xlsx')
operations_2019 = pd.read_excel('operations_2019.xlsx')
operations = pd.concat([operations_2018,operations_2019],ignore_index=True)
```

In [4]:

```
occ_gen = pd.read_csv('Occ_dur_ZL_pf_occupancy_dow_binofday.csv')
occ_LZ_PA = pd.read_csv('Occ_dur_ZL_pf_occupancy_ZoneId_ProfessionalActivity_dow_binofday.csv')
occ_VT = pd.read_csv('Occ_dur_vt_occupancy_VehicleType_dow_binofday.csv')
occ_dur = pd.read_csv('Occ_dur_full_arrivals_Duration_dow_binofday.csv')
vic_weather = pd.read_csv('vic_weather.csv')
```

In [5]:

```
occ_LZ_PA_full = pd.read_csv('Occ_dur_ZL_pf_bydatetime_ZoneId_ProfessionalActivity_datetime.csv')
occ_VT_full = pd.read_csv('Occ_dur_vt_bydatetime_VehicleType_datetime.csv')
occ_dur_full = pd.read_csv('Occ_dur_full_bydatetime_Duration_datetime.csv')
occupation = pd.read_csv('Occ_dur_ZL_pf_bydatetime_datetime.csv')
```

Databases descriptive analysis

In [6]:

```
operations.shape
```

Out[6]:

```
(103967, 16)
```

In [7]:

```
operations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103967 entries, 0 to 103966
Data columns (total 16 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   OperationId           103967 non-null  object
 1   VehicleId             103967 non-null  object
 2   TimeStart             103967 non-null  object
 3   TimeFinish            103967 non-null  object
 4   TimeLimit             103967 non-null  object
 5   VehicleType           103967 non-null  int64
 6   Emission              103953 non-null  float64
 7   PMR                   103967 non-null  int64
 8   Usage                 100323 non-null  float64
 9   VehicleAuthorized     103967 non-null  int64
 10  ProfessionalActivity  99127 non-null   float64
 11  Model                 103967 non-null  object
 12  ZoneId                103967 non-null  object
 13  ClosedBy              103967 non-null  object
 14  AreaFoundBy           103967 non-null  object
 15  DriverId              103967 non-null  object
dtypes: float64(3), int64(3), object(10)
memory usage: 12.7+ MB
```

In [8]:

```
operations.describe()
```

Out[8]:

	VehicleType	Emission	PMR	Usage	VehicleAuthorized	ProfessionalActivity
count	103967.000000	103953.000000	103967.000000	100323.000000	103967.000000	99127.000000
mean	1.919215	1.194857	0.035050	0.011921	0.012610	2.585774
std	1.379350	0.710804	0.183906	0.108533	0.111584	2.257321
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000
50%	1.000000	1.000000	0.000000	0.000000	0.000000	2.000000
75%	4.000000	2.000000	0.000000	0.000000	0.000000	5.000000
max	4.000000	4.000000	1.000000	1.000000	1.000000	7.000000

In [9]:

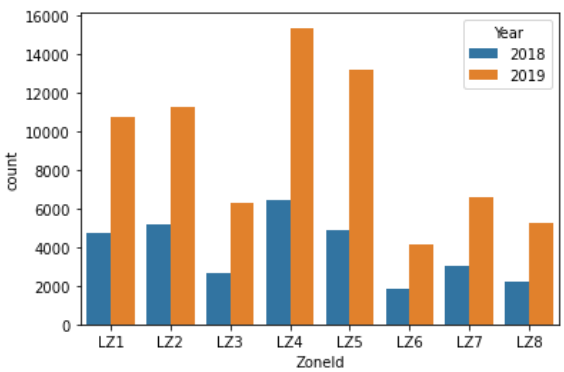
```
slz_names = {'3BEAFB2A-4678-4785-A397-90CC4C95829E':'LZ1','7908475C-E4D2-45F4-9CE9-DB6F28129C48':'LZ2',
               '051AA64E-9530-471F-AA58-B52D3BABE969':'LZ3', '7FEBCAEB-976F-4B1B-A31B-6F15EDE76D17':'LZ4',
               '326FB7A3-3A9F-472D-9FF9-348B0DD06206':'LZ5', '0F698316-47D1-470B-803A-478F6A4F529A':'LZ6',
               'CC2E43E1-F9C4-4269-AA13-AC20AE02140E':'LZ7', '598218CC-0AF2-4FD2-92C3-9160D83E5AD1':'LZ8'}
professional_activity = { 0:'Unspecified',1:'Install & Maintenance',2:'Transport and parcels',
                          3:'Construction',4:'Local Commerce',5: 'Commercial Agent',6:'Food',7:'Automotive',8:'Others'
emissions = {0: 'High Emissions',1: 'Medium Emissions',2: 'Low Emissions',3: 'Hybrid',4: 'eVehicle'}
vehicle_type = {0:'Private',1: 'LV',2:'HV',3:'Truck',4: 'Van'}

#Categorical variables labeling
operations['ZoneId'] = operations['ZoneId'].map(slz_names)
operations['ProfessionalActivity'] = operations['ProfessionalActivity'].map(professional_activity)
operations['Emission'] = operations['Emission'].map(emissions)
operations['VehicleType'] = operations['VehicleType'].map(vehicle_type)

#Dates to datetime format
operations['TimeFinish'] = pd.to_datetime(operations.loc[:,'TimeFinish'])
operations['TimeStart'] = pd.to_datetime(operations.loc[:,'TimeStart'])
operations['TimeLimit'] = pd.to_datetime(operations.loc[:,'TimeLimit'])

operations['Year'] = operations['TimeStart'].dt.year

#Duration variable inclusion
operations['Duration'] = operations['TimeFinish'] - operations['TimeStart']
operations['Duration'] = operations.loc[:,'Duration'].dt.total_seconds()/60
```

In [10]:

```
sns.countplot(data=operations,x='ZoneId',order=['LZ1','LZ2','LZ3','LZ4','LZ5','LZ6','LZ7','LZ8'],hue='Year')
```

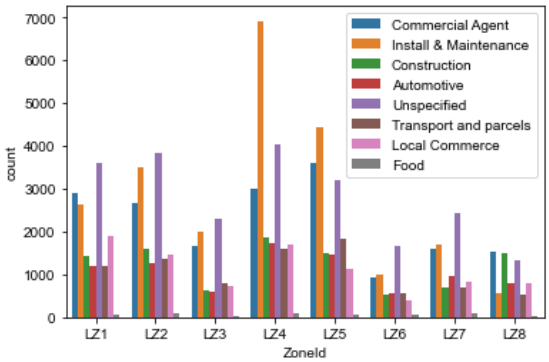
Out[10]:

```
<AxesSubplot:xlabel='ZoneId', ylabel='count'>
```



In [11]:

```
sns.countplot(data=operations,x='ZoneId',hue='ProfessionalActivity',order=['LZ1','LZ2','LZ3','LZ4','LZ5','LZ6','LZ7','
plt.legend()
sns.set(rc={'figure.figsize':(20,20)})
```



In [12]:

```
outlyers = operations[operations['Duration']>150]
outlyers
```

Out[12]:

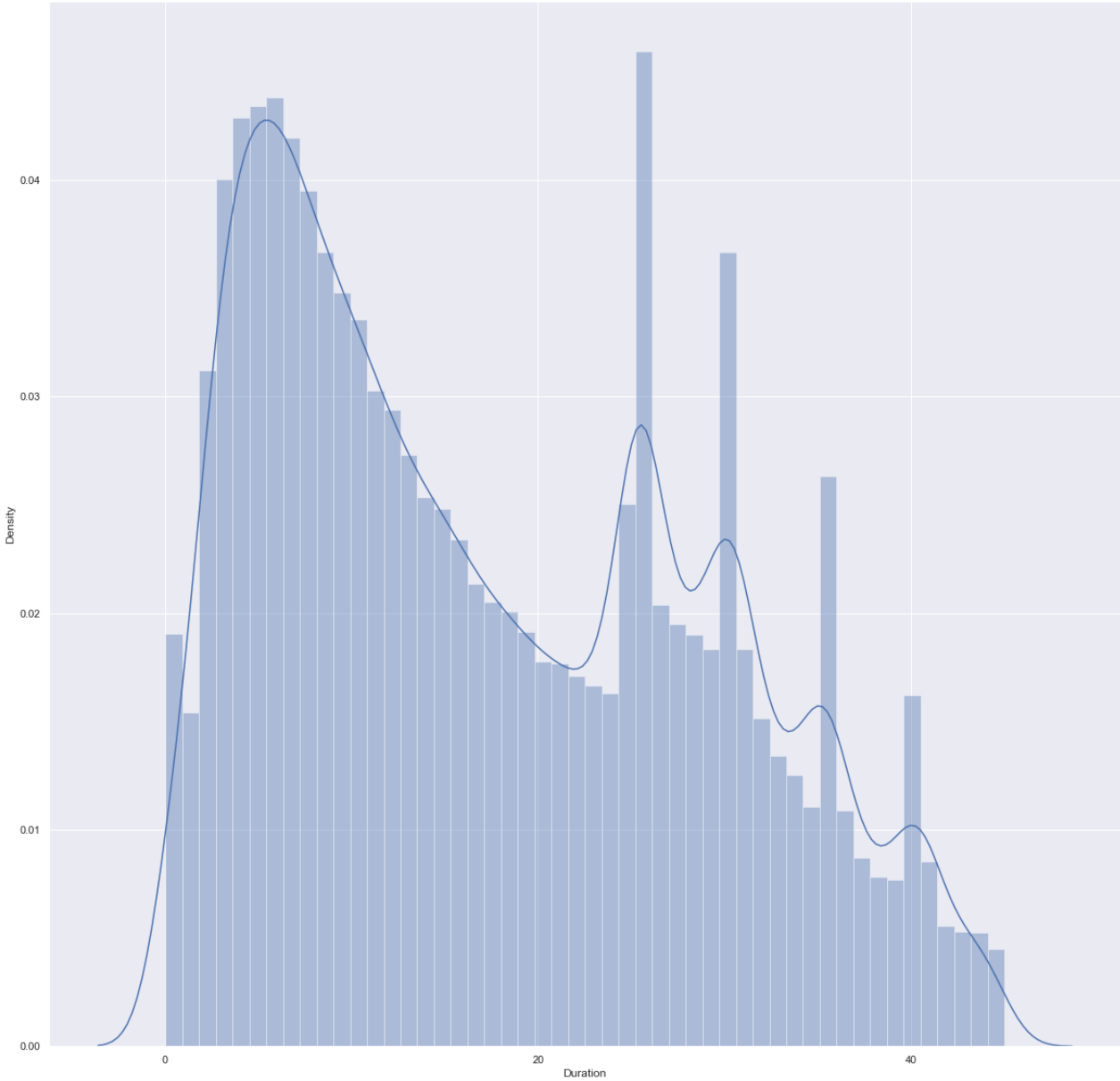
	OperationId	VehicleId	TimeStart	TimeFinish	TimeLimit	VehicleType	Emission	PMR	Usage	VehicleAuthorized	Profess
13320	21AFDF84-8BD0-4BF2-8F21-6A0055E08DFD	26ABAB48-9095-41DB-B37C-D1FA5396E642	2018-10-03 08:06:41.160	2018-10-03 17:03:37.650	2018-10-03 08:36:41.160	LV	High Emissions	0	0.0	0	
35672	98D7258E-1C0D-429A-B6CA-BC073EC2D143	125312EA-5D1E-489B-9BD9-C898E9F141A7	2019-01-23 10:52:44.337	2019-01-23 15:02:51.367	2019-01-23 11:22:44.337	LV	High Emissions	0	0.0	0	
60521	FD38372E-89B5-44DD-8337-84DF54340238	10C942F1-AE68-4F09-8300-910C7527FF4F	2019-05-21 09:29:42.030	2019-05-21 14:51:49.527	2019-05-21 09:59:42.030	LV	Medium Emissions	0	0.0	0	
70705	722DD874-F5F8-4FA8-933C-96DCEA082302	560F069C-1D3E-4BA7-B860-FACDE2BDA4E4	2019-07-10 08:13:13.157	2019-07-10 12:32:46.847	2019-07-10 08:23:13.157	Private	Low Emissions	0	1.0	1	
79439	6DB311D0-1129-4A1E-BE5C-43035D35FFC4	32088F5C-C5D1-4080-85EE-7897734884DE	2019-08-28 09:16:14.777	2019-08-28 15:02:24.697	2019-08-28 09:46:14.777	LV	High Emissions	0	0.0	0	
81771	79AAE8A2-AC63-49C1-B2FE-5AE77FC74A0A	738BBD35-0DA5-406A-B390-7DDE41C3118A	2019-09-09 08:45:45.727	2019-09-09 12:53:32.900	2019-09-09 09:15:45.727	LV	High Emissions	0	0.0	0	
91849	100BB85B-FF1C-4D54-B85D-FD577448781B	CCD61520-525E-4CF5-963D-604542BAE01E	2019-10-30 16:26:46.823	2019-10-31 09:58:47.897	2019-10-30 16:56:46.823	LV	High Emissions	0	0.0	0	Comr
91850	FF39E878-B854-413F-8B97-3A6E18970726	AE9A479D-9D37-424E-B7A6-8C5E6BEE7AB1	2019-10-30 16:26:58.303	2019-10-31 09:58:47.897	2019-10-30 16:56:58.303	LV	Low Emissions	0	0.0	0	Comr
91867	C791114D-3D16-45E3-9083-8E6D069898F5	67445FBD-CF56-4D28-B67D-70CC0AE1AB8C	2019-10-30 17:01:08.233	2019-10-31 09:40:35.700	2019-10-30 17:31:08.233	Van	Low Emissions	0	0.0	0	
91872	60BDF9F7-E7E8-464D-A395-EAA2792C8C9B	07CC1EB5-BE29-452C-B949-82E31E008C1B	2019-10-30 17:17:05.343	2019-10-31 09:58:47.897	2019-10-30 17:47:05.343	LV	Medium Emissions	0	0.0	0	
91884	FAF432F4-1B14-4A4E-BB2D-2D179BB064AA	2162690E-6576-4D0D-8808-524F4F4B28C7	2019-10-30 18:00:03.613	2019-10-31 09:58:47.897	2019-10-30 18:10:03.613	Private	Medium Emissions	0	1.0	1	
91897	936E9467-18DA-4DDD-9B44-4C5AA40552F0	9469E8D9-BFE3-4199-9F41-09C20F024BB4	2019-10-30 18:39:22.213	2019-10-31 09:58:47.897	2019-10-30 19:00:59.000	Private	Low Emissions	1	NaN	0	

In [13]:

```
durations = operations[(operations['Duration']<45)&(operations['ClosedBy']=='User')]
```

In [14]:

```
sns.distplot(durations['Duration'])
sns.set(style="whitegrid", font_scale=6)
```



In [15]:

```
operations['Duration'].count()
```

Out[15]:

```
103967
```

In [16]:

```
operations['Duration'].describe()
```

Out[16]:

```
count    103967.000000
mean         22.778443
std          16.983574
min           0.029767
25%           8.792417
50%          20.934833
75%          35.451475
max        1052.017900
Name: Duration, dtype: float64
```

In [17]:

```
durations['Duration'].count()
```

Out[17]:

```
83490
```

In [18]:

```
durations['Duration'].describe()
```

Out[18]:

```
count    83490.000000
mean        17.491083
std         11.630924
min          0.029767
25%          7.228350
50%         15.174167
75%         26.685388
max         44.999833
Name: Duration, dtype: float64
```

In [19]:

```
durations.info()
```

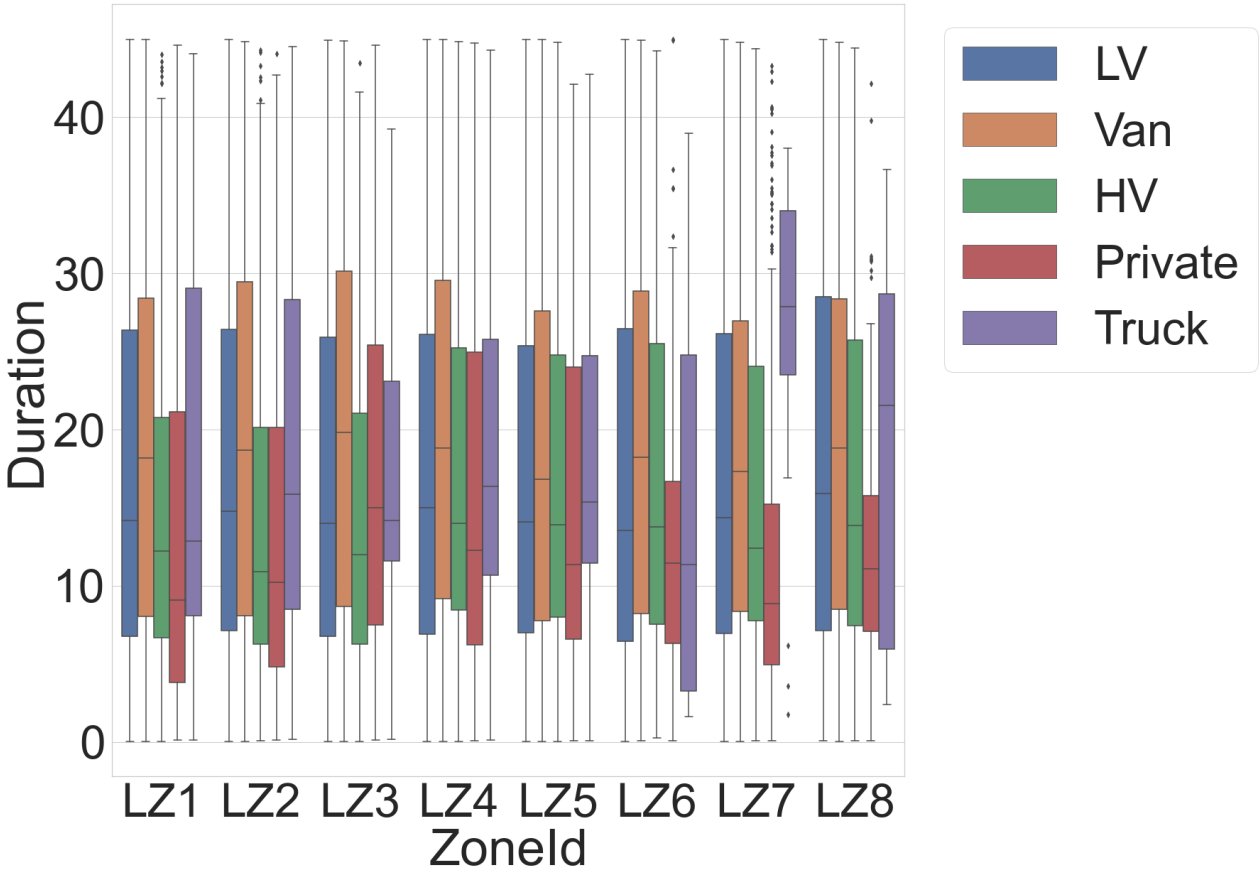
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 83490 entries, 0 to 103965
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   OperationId           83490 non-null  object
 1   VehicleId             83490 non-null  object
 2   TimeStart             83490 non-null  datetime64[ns]
 3   TimeFinish            83490 non-null  datetime64[ns]
 4   TimeLimit             83490 non-null  datetime64[ns]
 5   VehicleType           83490 non-null  object
 6   Emission              83478 non-null  object
 7   PMR                   83490 non-null  int64
 8   Usage                 80426 non-null  float64
 9   VehicleAuthorized     83490 non-null  int64
 10  ProfessionalActivity  79594 non-null  object
 11  Model                 83490 non-null  object
 12  ZoneId                83484 non-null  object
 13  ClosedBy              83490 non-null  object
 14  AreaFoundBy           83490 non-null  object
 15  DriverId              83490 non-null  object
 16  Year                  83490 non-null  int64
 17  Duration              83490 non-null  float64
dtypes: datetime64[ns](3), float64(2), int64(3), object(10)
memory usage: 12.1+ MB
```

In [20]:

```
sns.boxplot(x='ZoneId',y='Duration',data=durations,hue='VehicleType',order=['LZ1','LZ2','LZ3','LZ4','LZ5','LZ6','LZ7',
plt.legend(loc='upper left',bbox_to_anchor=(1.02, 1))
```

Out[20]:

```
<matplotlib.legend.Legend at 0x7fc9867547f0>
```

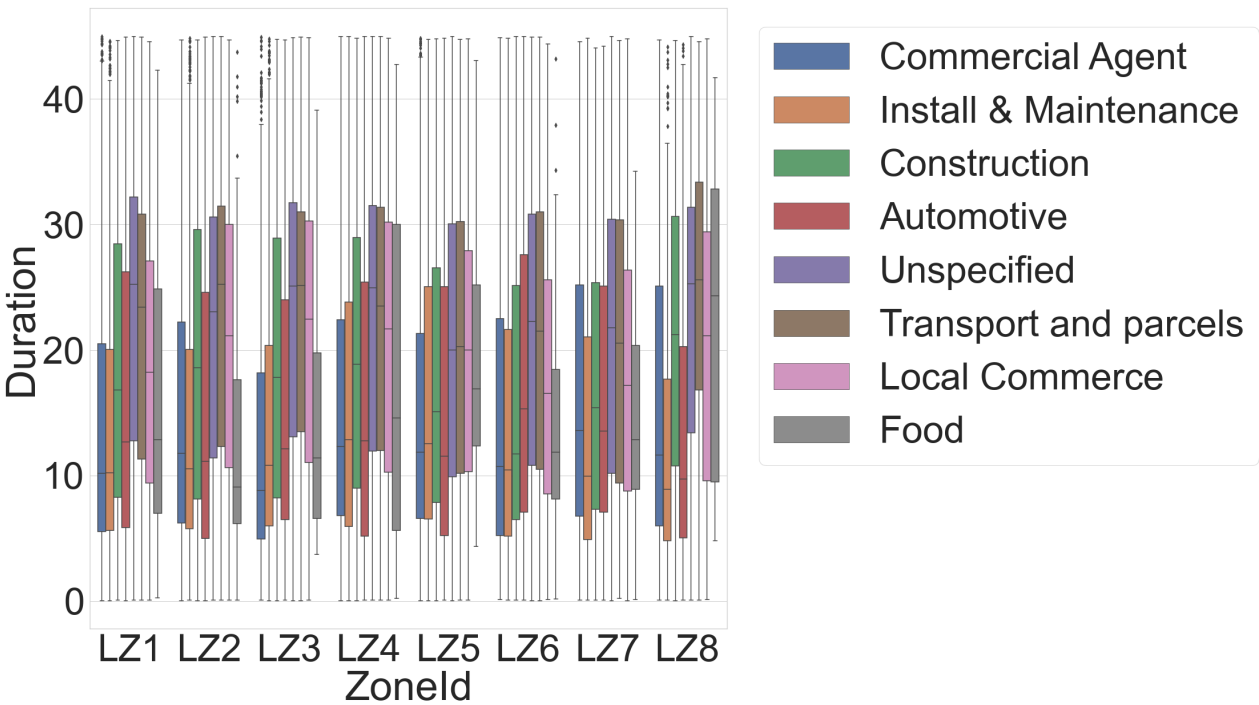


In [21]:

```
sns.boxplot(x='ZoneId',y='Duration',data=durations,hue='ProfessionalActivity',order=['LZ1','LZ2','LZ3','LZ4','LZ5','LZ
plt.legend(loc='upper left',bbox_to_anchor=(1.02, 1))
```

Out[21]:

```
<matplotlib.legend.Legend at 0x7fc985ca1b80>
```



DB Cleaning

In [22]:

```
durations.drop(labels=['AreaFoundBy','PMR','OperationId','VehicleId','TimeLimit',
                       'Usage','VehicleAuthorized','ClosedBy','DriverId','Model'],axis=1,inplace=True)
durations.head()
```

Out[22]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration
0	2018-07-02 04:51:15.903	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.53300
1	2018-07-02 06:20:19.950	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.35495
2	2018-07-02 06:25:32.173	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.17790
3	2018-07-02 06:34:09.463	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.79950
4	2018-07-02 06:48:58.327	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.42210

In [23]:

```
durations.tail()
```

Out[23]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration
103959	2019-12-31 16:45:14.473	2019-12-31 16:49:38.977	Private	Medium Emissions	NaN	LZ5	2019	4.408400
103960	2019-12-31 16:59:11.010	2019-12-31 17:15:41.267	Van	Low Emissions	Unspecified	LZ2	2019	16.504283
103961	2019-12-31 17:02:59.563	2019-12-31 17:34:03.320	LV	Low Emissions	Install & Maintenance	LZ4	2019	31.062617
103963	2019-12-31 17:07:32.757	2019-12-31 17:24:06.327	Van	Medium Emissions	Construction	LZ2	2019	16.559500
103965	2019-12-31 17:31:15.193	2019-12-31 17:38:30.293	LV	Low Emissions	Construction	LZ5	2019	7.251667

In [24]:

```
durations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 83490 entries, 0 to 103965
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             83490 non-null  datetime64[ns]
 1   TimeFinish            83490 non-null  datetime64[ns]
 2   VehicleType           83490 non-null  object
 3   Emission              83478 non-null  object
 4   ProfessionalActivity  79594 non-null  object
 5   ZoneId                83484 non-null  object
 6   Year                  83490 non-null  int64
 7   Duration              83490 non-null  float64
dtypes: datetime64[ns](2), float64(1), int64(1), object(4)
memory usage: 7.7+ MB
```

Occupation analysis

In [25]:

```
help(hm.make_hills)
```

```
Help on function make_hills in module hillmaker.hills:

make_hills(scenario_name, stops_df, infield, outfield, start_analysis, end_analysis, catfield=None, bin_s
ize_minutes=60, percentiles=(0.25, 0.5, 0.75, 0.95, 0.99), cat_to_exclude=None, occ_weight_field=None, to
tals=1, nonstationary_stats=True, stationary_stats=True, export_bydatetime_csv=True, export_summaries_csv
=True, export_path='.', edge_bins=1, verbose=0)
    Compute occupancy, arrival, and departure statistics by time bin of day and day of week.

    Main function that first calls `bydatetime.make_bydatetime` to calculate occupancy, arrival
    and departure values by date by time bin and then calls `summarize.summarize`
    to compute the summary statistics.

    Parameters
    ----------

    scenario_name : string
        Used in output filenames
    stops_df : DataFrame
        Base data containing one row per visit
    infield : string
        Column name corresponding to the arrival times
    outfield : string
        Column name corresponding to the departure times
    start_analysis : datetime-like, str
        Starting datetime for the analysis (must be convertible to pandas Timestamp)
    end_analysis : datetime-like, str
        Ending datetime for the analysis (must be convertible to pandas Timestamp)
    catfield : string or List of strings, optional
        Column name(s) corresponding to the categories. If none is specified, then
        only overall occupancy is analyzed.
        Default is None
    bin_size_minutes : int, optional
        Number of minutes in each time bin of the day, default is 60
    percentiles : list or tuple of floats (e.g. [0.5, 0.75, 0.95]), optional
        Which percentiles to compute. Default is (0.25, 0.5, 0.75, 0.95, 0.99)
    cat_to_exclude : list, optional
        Categories to ignore, default is None
    occ_weight_field : string, optional
        Column name corresponding to the weights to use for occupancy incrementing.
    edge_bins: int, default 1
        Occupancy contribution method for arrival and departure bins. 1=fractional, 2=whole bin
    totals: int, default 1
        0=no totals, 1=totals by datetime, 2=totals bydatetime as well as totals for each field in the
        catfields (only relevant for > 1 category field)
    nonstationary_stats : bool, optional
       If True, datetime bin stats are computed. Else, they aren't computed. Default is True
    stationary_stats : bool, optional
       If True, overall, non time bin dependent, stats are computed. Else, they aren't computed. Default
is True
    export_bydatetime_csv : bool, optional
       If True, bydatetime DataFrames are exported to csv files. Default is True.
    export_summaries_csv : bool, optional
       If True, summary DataFrames are exported to csv files. Default is True.
    export_path : string, optional
        Destination path for exported csv files, default is current directory
    verbose : int, optional
        The verbosity level. The default, zero, means silent mode. Higher numbers mean more output messag
es.

    Returns
    -------
    dict of DataFrames
       The bydatetime DataFrames and all summary DataFrames.
```

In [ ]:

```
# Required inputs
# ---------------

# scenario name
#scenario = 'Occ_dur_LZ'

# Column name in trip_df corresponding to the bike rental time
#in_fld_name = 'TimeStart'

# Column name in trip_df corresponding to the bike return time
#out_fld_name = 'TimeFinish'

# Column name in trip_df corresponding to some categorical field for grouping
#cat_fld_name = ['ZoneId']

# Start and end times for the analysis. We'll just use data from 2015.
#start = '07/02/2018 00:00:00'
#end = '12/31/2019 22:00:00'

# Optional inputs
# ---------------

# Verbosity level of hillmaker messages
#verbose = 1

# Path to destination location for csv output files produced by hillmaker
#output = './content2'
```

In [111]:

```
#hills_5 = hm.make_hills(scenario, duraciones, in_fld_name, out_fld_name, start, end, cat_fld_name,
 #             bin_size_minutes=1,verbose=verbose)
```

```
min of intime: 2018-07-02 04:51:15.903000
max of intime: 2019-12-31 17:31:15.193000
min of outtime: 2018-07-02 04:58:47.883000
max of outtime: 2019-12-31 17:38:30.293000
85023 stop records processed.
Datetime DataFrame created (seconds): 312.9828
Created nonstationary summaries - ['ZoneId']
Created nonstationary summaries - []
Created stationary summaries - ['ZoneId']
Created stationary summaries - []
Summaries by datetime created (seconds): 2070.1409
By datetime exported to csv (seconds): 540.7605
Summaries exported to csv (seconds): 6.8982
Total time (seconds): 2930.7850
```

In [26]:

```
df_occ_LZ = pd.read_csv('Occ_dur_LZ_bydatetime_ZoneId_datetime.csv')
```

In [27]:

```
df_occ_LZ
```

Out[27]:

	ZoneId	datetime	arrivals	departures	occupancy	day_of_week	dow_name	bin_of_day	bin_of_week
0	VIC-001	2018-07-02 00:00:00	0.0	0.0	0.0	0	Monday	0	0
1	VIC-001	2018-07-02 00:01:00	0.0	0.0	0.0	0	Monday	1	1
2	VIC-001	2018-07-02 00:02:00	0.0	0.0	0.0	0	Monday	2	2
3	VIC-001	2018-07-02 00:03:00	0.0	0.0	0.0	0	Monday	3	3
4	VIC-001	2018-07-02 00:04:00	0.0	0.0	0.0	0	Monday	4	4
...	...	...	...	...	...	...	...	...	...
7101004	NaN	2019-12-31 21:56:00	0.0	0.0	0.0	1	Tuesday	1316	2756
7101005	NaN	2019-12-31 21:57:00	0.0	0.0	0.0	1	Tuesday	1317	2757
7101006	NaN	2019-12-31 21:58:00	0.0	0.0	0.0	1	Tuesday	1318	2758
7101007	NaN	2019-12-31 21:59:00	0.0	0.0	0.0	1	Tuesday	1319	2759
7101008	NaN	2019-12-31 22:00:00	0.0	0.0	0.0	1	Tuesday	1320	2760

7101009 rows × 9 columns

In [28]:

```
df_occ_LZ['ZoneId'].isnull().value_counts()
```

Out[28]:

```
False    6312008
True      789001
Name: ZoneId, dtype: int64
```

In [29]:

```
df_occ_LZ.dropna(axis=0,how='any',inplace=True)
```

In [30]:

```
df_occ_LZ
```

Out[30]:

	ZoneId	datetime	arrivals	departures	occupancy	day_of_week	dow_name	bin_of_day	bin_of_week
0	VIC-001	2018-07-02 00:00:00	0.0	0.0	0.0	0	Monday	0	0
1	VIC-001	2018-07-02 00:01:00	0.0	0.0	0.0	0	Monday	1	1
2	VIC-001	2018-07-02 00:02:00	0.0	0.0	0.0	0	Monday	2	2
3	VIC-001	2018-07-02 00:03:00	0.0	0.0	0.0	0	Monday	3	3
4	VIC-001	2018-07-02 00:04:00	0.0	0.0	0.0	0	Monday	4	4
...	...	...	...	...	...	...	...	...	...
6312003	VIC-008	2019-12-31 21:56:00	0.0	0.0	0.0	1	Tuesday	1316	2756
6312004	VIC-008	2019-12-31 21:57:00	0.0	0.0	0.0	1	Tuesday	1317	2757
6312005	VIC-008	2019-12-31 21:58:00	0.0	0.0	0.0	1	Tuesday	1318	2758
6312006	VIC-008	2019-12-31 21:59:00	0.0	0.0	0.0	1	Tuesday	1319	2759
6312007	VIC-008	2019-12-31 22:00:00	0.0	0.0	0.0	1	Tuesday	1320	2760

6125924 rows × 9 columns

In [31]:

```
days_of_week = ['Monday','Tuesday','Wednesday','Thursday','Friday']
df_occ_hour = df_occ_LZ[(df_occ_LZ['dow_name'].isin(days_of_week)) & (df_occ_LZ['bin_of_day']>359) & (df_occ_LZ['bin_o
df_occ_hour['datetime'] = pd.to_datetime(df_occ_hour.loc[:,'datetime'])
df_occ_hour
```

Out[31]:

	ZoneId	datetime	arrivals	departures	occupancy	day_of_week	dow_name	bin_of_day	bin_of_week
360	VIC-001	2018-07-02 06:00:00	0.0	0.0	0.0	0	Monday	360	360
361	VIC-001	2018-07-02 06:01:00	0.0	0.0	0.0	0	Monday	361	361
362	VIC-001	2018-07-02 06:02:00	0.0	0.0	0.0	0	Monday	362	362
363	VIC-001	2018-07-02 06:03:00	0.0	0.0	0.0	0	Monday	363	363
364	VIC-001	2018-07-02 06:04:00	0.0	0.0	0.0	0	Monday	364	364
...	...	...	...	...	...	...	...	...	...
6311763	VIC-008	2019-12-31 17:56:00	0.0	0.0	0.0	1	Tuesday	1076	2516
6311764	VIC-008	2019-12-31 17:57:00	0.0	0.0	0.0	1	Tuesday	1077	2517
6311765	VIC-008	2019-12-31 17:58:00	0.0	0.0	0.0	1	Tuesday	1078	2518
6311766	VIC-008	2019-12-31 17:59:00	0.0	0.0	0.0	1	Tuesday	1079	2519
6311767	VIC-008	2019-12-31 18:00:00	0.0	0.0	0.0	1	Tuesday	1080	2520

2096628 rows × 9 columns

In [32]:

```
df_occ_hour.groupby(by='ZoneId').agg({'occupancy':'max'})
```

Out[32]:

ZoneId	occupancy
VIC-001	7.800000
VIC-002	8.000000
VIC-003	5.000000
VIC-004	9.000000
VIC-005	8.000000
VIC-006	4.000000
VIC-007	6.000000
VIC-008	5.583333

In [33]:

```
df_occ_1 = df_occ_LZ[df_occ_LZ['arrivals']>0]
df_occ_1
```

Out[33]:

	ZoneId	datetime	arrivals	departures	occupancy	day_of_week	dow_name	bin_of_day	bin_of_week
291	VIC-001	2018-07-02 04:51:00	1.0	0.0	0.750000	0	Monday	291	291
408	VIC-001	2018-07-02 06:48:00	1.0	0.0	0.033333	0	Monday	408	408
411	VIC-001	2018-07-02 06:51:00	1.0	0.0	1.683333	0	Monday	411	411
473	VIC-001	2018-07-02 07:53:00	1.0	0.0	0.233333	0	Monday	473	473
474	VIC-001	2018-07-02 07:54:00	1.0	1.0	0.866667	0	Monday	474	474
...	...	...	...	...	...	...	...	...	...
6171904	VIC-008	2019-09-25 14:57:00	1.0	0.0	0.300000	2	Wednesday	897	3777
6171925	VIC-008	2019-09-25 15:18:00	1.0	0.0	1.966667	2	Wednesday	918	3798
6171926	VIC-008	2019-09-25 15:19:00	1.0	0.0	2.800000	2	Wednesday	919	3799
6171935	VIC-008	2019-09-25 15:28:00	1.0	1.0	3.000000	2	Wednesday	928	3808
6171952	VIC-008	2019-09-25 15:45:00	1.0	0.0	1.083333	2	Wednesday	945	3825

68430 rows × 9 columns

In [34]:

```
df_occ_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68430 entries, 291 to 6171952
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   ZoneId       68430 non-null  object 
 1   datetime     68430 non-null  object 
 2   arrivals     68430 non-null  float64
 3   departures   68430 non-null  float64
 4   occupancy    68430 non-null  float64
 5   day_of_week  68430 non-null  int64  
 6   dow_name     68430 non-null  object 
 7   bin_of_day   68430 non-null  int64  
 8   bin_of_week  68430 non-null  int64  
dtypes: float64(3), int64(3), object(3)
memory usage: 5.2+ MB
```

In [35]:

```
df_occ_1['datetime'] = pd.to_datetime(df_occ_1.loc[:,'datetime'])
```

In [36]:

```
df_occ_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68430 entries, 291 to 6171952
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ZoneId       68430 non-null  object
 1   datetime     68430 non-null  datetime64[ns]
 2   arrivals     68430 non-null  float64
 3   departures   68430 non-null  float64
 4   occupancy    68430 non-null  float64
 5   day_of_week  68430 non-null  int64
 6   dow_name     68430 non-null  object
 7   bin_of_day   68430 non-null  int64
 8   bin_of_week  68430 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(3), object(2)
memory usage: 5.2+ MB
```

In [37]:

```
LZ_names = {'VIC-001':'LZ1','VIC-002':'LZ2',
               'VIC-003':'LZ3','VIC-004':'LZ4',
               'VIC-005':'LZ5','VIC-006':'LZ6',
               'VIC-007':'LZ7','VIC-008':'LZ8'}


df_occ_1['ZoneId'] = df_occ_1['ZoneId'].map(LZ_names)
```

In [38]:

```
df_occ_1
```

Out[38]:

	ZoneId	datetime	arrivals	departures	occupancy	day_of_week	dow_name	bin_of_day	bin_of_week
291	LZ1	2018-07-02 04:51:00	1.0	0.0	0.750000	0	Monday	291	291
408	LZ1	2018-07-02 06:48:00	1.0	0.0	0.033333	0	Monday	408	408
411	LZ1	2018-07-02 06:51:00	1.0	0.0	1.683333	0	Monday	411	411
473	LZ1	2018-07-02 07:53:00	1.0	0.0	0.233333	0	Monday	473	473
474	LZ1	2018-07-02 07:54:00	1.0	1.0	0.866667	0	Monday	474	474
...	...	...	...	...	...	...	...	...	...
6171904	LZ8	2019-09-25 14:57:00	1.0	0.0	0.300000	2	Wednesday	897	3777
6171925	LZ8	2019-09-25 15:18:00	1.0	0.0	1.966667	2	Wednesday	918	3798
6171926	LZ8	2019-09-25 15:19:00	1.0	0.0	2.800000	2	Wednesday	919	3799
6171935	LZ8	2019-09-25 15:28:00	1.0	1.0	3.000000	2	Wednesday	928	3808
6171952	LZ8	2019-09-25 15:45:00	1.0	0.0	1.083333	2	Wednesday	945	3825

68430 rows × 9 columns

Association Analysis - GLZM

Design matrix configuration

In [39]:

```
durations
```

Out[39]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration
0	2018-07-02 04:51:15.903	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.533000
1	2018-07-02 06:20:19.950	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.354950
2	2018-07-02 06:25:32.173	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.177900
3	2018-07-02 06:34:09.463	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.799500
4	2018-07-02 06:48:58.327	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.422100
...	...	...	...	...	...	...	...	...
103959	2019-12-31 16:45:14.473	2019-12-31 16:49:38.977	Private	Medium Emissions	NaN	LZ5	2019	4.408400
103960	2019-12-31 16:59:11.010	2019-12-31 17:15:41.267	Van	Low Emissions	Unspecified	LZ2	2019	16.504283
103961	2019-12-31 17:02:59.563	2019-12-31 17:34:03.320	LV	Low Emissions	Install & Maintenance	LZ4	2019	31.062617
103963	2019-12-31 17:07:32.757	2019-12-31 17:24:06.327	Van	Medium Emissions	Construction	LZ2	2019	16.559500
103965	2019-12-31 17:31:15.193	2019-12-31 17:38:30.293	LV	Low Emissions	Construction	LZ5	2019	7.251667

83490 rows × 8 columns

In [40]:

```
df_validation=durations[durations['TimeStart']>'2019-09-25']
```

In [41]:

```
df_validation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14540 entries, 84776 to 103965
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             14540 non-null  datetime64[ns]
 1   TimeFinish            14540 non-null  datetime64[ns]
 2   VehicleType           14540 non-null  object
 3   Emission              14539 non-null  object
 4   ProfessionalActivity  13823 non-null  object
 5   ZoneId                14540 non-null  object
 6   Year                  14540 non-null  int64
 7   Duration              14540 non-null  float64
dtypes: datetime64[ns](2), float64(1), int64(1), object(4)
memory usage: 1022.3+ KB
```

In [42]:

```
df_validation.dropna(inplace=True)
```

In [43]:

```
df_validation
```

Out[43]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration
84776	2019-09-25 05:48:36.093	2019-09-25 06:18:46.273	LV	Medium Emissions	Unspecified	LZ4	2019	30.169667
84777	2019-09-25 06:01:58.570	2019-09-25 06:32:16.897	LV	Low Emissions	Commercial Agent	LZ5	2019	30.305450
84779	2019-09-25 06:12:51.333	2019-09-25 06:51:03.673	Van	Medium Emissions	Transport and parcels	LZ5	2019	38.205667
84780	2019-09-25 06:23:00.200	2019-09-25 06:52:39.777	LV	Medium Emissions	Unspecified	LZ4	2019	29.659617
84781	2019-09-25 06:25:23.463	2019-09-25 07:05:43.897	LV	Low Emissions	Automotive	LZ4	2019	40.340567
...	...	...	...	...	...	...	...	...
103958	2019-12-31 16:11:19.307	2019-12-31 16:38:39.057	Van	Medium Emissions	Unspecified	LZ3	2019	27.329167
103960	2019-12-31 16:59:11.010	2019-12-31 17:15:41.267	Van	Low Emissions	Unspecified	LZ2	2019	16.504283
103961	2019-12-31 17:02:59.563	2019-12-31 17:34:03.320	LV	Low Emissions	Install & Maintenance	LZ4	2019	31.062617
103963	2019-12-31 17:07:32.757	2019-12-31 17:24:06.327	Van	Medium Emissions	Construction	LZ2	2019	16.559500

In [44]:

```
vic_weather.head()
```

Out[44]:

	date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	moon_illumination	moonrise	moonset	sunrise	...	WindGustKmph	clou
0	2018-07-02 00:00:00	29	19	0.0	13.6	6	73	10:45 PM	08:20 AM	04:29 AM	...	14	
1	2018-07-02 01:00:00	29	19	0.0	13.6	6	73	10:45 PM	08:20 AM	04:29 AM	...	14	
2	2018-07-02 02:00:00	29	19	0.0	13.6	6	73	10:45 PM	08:20 AM	04:29 AM	...	13	
3	2018-07-02 03:00:00	29	19	0.0	13.6	6	73	10:45 PM	08:20 AM	04:29 AM	...	12	
4	2018-07-02 04:00:00	29	19	0.0	13.6	6	73	10:45 PM	08:20 AM	04:29 AM	...	10	

5 rows × 25 columns

In [45]:

```
durations['TimeWeather'] = durations['TimeStart'].dt.floor('H')
```

In [46]:

```
durations.head()
```

Out[46]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration	TimeWeather
0	2018-07-02 04:51:15.903	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.53300	2018-07-02 04:00:00
1	2018-07-02 06:20:19.950	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.35495	2018-07-02 06:00:00
2	2018-07-02 06:25:32.173	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.17790	2018-07-02 06:00:00
3	2018-07-02 06:34:09.463	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.79950	2018-07-02 06:00:00
4	2018-07-02 06:48:58.327	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.42210	2018-07-02 06:00:00

In [47]:

```
durations['TimeStart'] = durations['TimeStart'].dt.floor('Min')
```

In [48]:

```
durations
```

Out[48]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration	TimeWeather
0	2018-07-02 04:51:00	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.533000	2018-07-02 04:00:00
1	2018-07-02 06:20:00	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.354950	2018-07-02 06:00:00
2	2018-07-02 06:25:00	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.177900	2018-07-02 06:00:00
3	2018-07-02 06:34:00	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.799500	2018-07-02 06:00:00
4	2018-07-02 06:48:00	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.422100	2018-07-02 06:00:00
...	...	...	...	...	...	...	...	...	...
103959	2019-12-31 16:45:00	2019-12-31 16:49:38.977	Private	Medium Emissions	NaN	LZ5	2019	4.408400	2019-12-31 16:00:00
103960	2019-12-31 16:59:00	2019-12-31 17:15:41.267	Van	Low Emissions	Unspecified	LZ2	2019	16.504283	2019-12-31 16:00:00
103961	2019-12-31 17:02:00	2019-12-31 17:34:03.320	LV	Low Emissions	Install & Maintenance	LZ4	2019	31.062617	2019-12-31 17:00:00
103963	2019-12-31 17:07:00	2019-12-31 17:24:06.327	Van	Medium Emissions	Construction	LZ2	2019	16.559500	2019-12-31 17:00:00
103965	2019-12-31 17:31:00	2019-12-31 17:38:30.293	LV	Low Emissions	Construction	LZ5	2019	7.251667	2019-12-31 17:00:00

83490 rows × 9 columns

In [49]:

```
df = pd.merge(durations, df_occ_1,how='inner',left_on=['TimeStart','ZoneId'],right_on=['datetime','ZoneId'])
```

In [50]:

```
df
```

Out[50]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration	TimeWeather	datetime	arrivals	departure
0	2018-07-02 04:51:00	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.533000	2018-07-02 04:00:00	2018-07-02 04:51:00	1.0	0.
1	2018-07-02 06:20:00	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.354950	2018-07-02 06:00:00	2018-07-02 06:20:00	1.0	0.
2	2018-07-02 06:25:00	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.177900	2018-07-02 06:00:00	2018-07-02 06:25:00	1.0	0.
3	2018-07-02 06:34:00	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.799500	2018-07-02 06:00:00	2018-07-02 06:34:00	1.0	0.
4	2018-07-02 06:48:00	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.422100	2018-07-02 06:00:00	2018-07-02 06:48:00	1.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	.
69133	2019-09-25 15:51:00	2019-09-25 16:08:18.523	LV	Low Emissions	Unspecified	LZ1	2019	16.892667	2019-09-25 15:00:00	2019-09-25 15:51:00	1.0	0.
69134	2019-09-25 15:57:00	2019-09-25 15:58:23.747	LV	High Emissions	Unspecified	LZ5	2019	0.957667	2019-09-25 15:00:00	2019-09-25 15:57:00	1.0	0.
69135	2019-09-25 15:59:00	2019-09-25 16:43:36.883	LV	Medium Emissions	Unspecified	LZ4	2019	44.120100	2019-09-25 15:00:00	2019-09-25 15:59:00	1.0	0.
69136	2019-09-25 16:02:00	2019-09-25 16:08:04.370	Van	Medium Emissions	Local Commerce	LZ5	2019	5.280450	2019-09-25 16:00:00	2019-09-25 16:02:00	1.0	0.
69137	2019-09-25 16:08:00	2019-09-25 16:24:52.283	LV	Low Emissions	Commercial Agent	LZ4	2019	15.966667	2019-09-25 16:00:00	2019-09-25 16:08:00	1.0	0.

69138 rows × 17 columns

In [51]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69138 entries, 0 to 69137
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             69138 non-null  datetime64[ns]
 1   TimeFinish            69138 non-null  datetime64[ns]
 2   VehicleType           69138 non-null  object
 3   Emission              69128 non-null  object
 4   ProfessionalActivity  65952 non-null  object
 5   ZoneId                69138 non-null  object
 6   Year                  69138 non-null  int64
 7   Duration              69138 non-null  float64
 8   TimeWeather           69138 non-null  datetime64[ns]
 9   datetime              69138 non-null  datetime64[ns]
 10  arrivals              69138 non-null  float64
 11  departures            69138 non-null  float64
 12  occupancy             69138 non-null  float64
 13  day_of_week           69138 non-null  int64
 14  dow_name              69138 non-null  object
 15  bin_of_day            69138 non-null  int64
 16  bin_of_week           69138 non-null  int64
dtypes: datetime64[ns](4), float64(4), int64(4), object(5)
memory usage: 9.5+ MB
```

In [52]:

```
df['ZoneId'].isnull().value_counts()
```

Out[52]:

```
False    69138
Name: ZoneId, dtype: int64
```

In [53]:

```
null = {'Emission':'Medium Emissions','ProfessionalActivity':'Unspecified'}
df.fillna(value=null,inplace=True)
```

In [54]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69138 entries, 0 to 69137
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             69138 non-null  datetime64[ns]
 1   TimeFinish            69138 non-null  datetime64[ns]
 2   VehicleType           69138 non-null  object
 3   Emission              69138 non-null  object
 4   ProfessionalActivity  69138 non-null  object
 5   ZoneId                69138 non-null  object
 6   Year                  69138 non-null  int64
 7   Duration              69138 non-null  float64
 8   TimeWeather           69138 non-null  datetime64[ns]
 9   datetime              69138 non-null  datetime64[ns]
 10  arrivals              69138 non-null  float64
 11  departures            69138 non-null  float64
 12  occupancy             69138 non-null  float64
 13  day_of_week           69138 non-null  int64
 14  dow_name              69138 non-null  object
 15  bin_of_day            69138 non-null  int64
 16  bin_of_week           69138 non-null  int64
dtypes: datetime64[ns](4), float64(4), int64(4), object(5)
memory usage: 9.5+ MB
```

In [55]:

```
#Adding time variables
df['Month'] = df['TimeStart'].dt.month
df['QuarterStart'] = df['TimeStart'].dt.is_month_start
df['QuarterEnd'] = df['TimeStart'].dt.is_month_end

month_labels = {True:1,False:0}

df['QuarterStart'] = df['QuarterStart'].map(month_labels)
df['QuarterEnd'] = df['QuarterEnd'].map(month_labels)

df.rename(columns={'QuarterStart':'MonthStart','QuarterEnd':'MonthEnd'},inplace=True)

#Adding occupancy variables
LZ_capacity = {'LZ1':4,'LZ2':6,'LZ3':2,'LZ4':8,'LZ5':8,'LZ6':4,'LZ7':7,'LZ8':8}
df['Occupancy_Rate'] = df['ZoneId'].map(LZ_capacity)
df['Occupancy_Rate'] = df.loc[:,'occupancy']/df.loc[:,'Occupancy_Rate']

#Possible response variable transformation
df['logarithm_Dur'] = np.log10(df['Duration'])
```

In [56]:

```
df
```

Out[56]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration	TimeWeather	datetime	...	occupancy	da
0	2018-07-02 04:51:00	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.533000	2018-07-02 04:00:00	2018-07-02 04:51:00	...	0.750000	
1	2018-07-02 06:20:00	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.354950	2018-07-02 06:00:00	2018-07-02 06:20:00	...	0.683333	
2	2018-07-02 06:25:00	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.177900	2018-07-02 06:00:00	2018-07-02 06:25:00	...	0.466667	
3	2018-07-02 06:34:00	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.799500	2018-07-02 06:00:00	2018-07-02 06:34:00	...	0.850000	
4	2018-07-02 06:48:00	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.422100	2018-07-02 06:00:00	2018-07-02 06:48:00	...	0.033333	
...	...	...	...	...	...	...	...	...	...	...	...	...	
69133	2019-09-25 15:51:00	2019-09-25 16:08:18.523	LV	Low Emissions	Unspecified	LZ1	2019	16.892667	2019-09-25 15:00:00	2019-09-25 15:51:00	...	2.600000	
69134	2019-09-25 15:57:00	2019-09-25 15:58:23.747	LV	High Emissions	Unspecified	LZ5	2019	0.957667	2019-09-25 15:00:00	2019-09-25 15:57:00	...	1.566667	
69135	2019-09-25 15:59:00	2019-09-25 16:43:36.883	LV	Medium Emissions	Unspecified	LZ4	2019	44.120100	2019-09-25 15:00:00	2019-09-25 15:59:00	...	1.516667	
69136	2019-09-25 16:02:00	2019-09-25 16:08:04.370	Van	Medium Emissions	Local Commerce	LZ5	2019	5.280450	2019-09-25 16:00:00	2019-09-25 16:02:00	...	0.216667	
69137	2019-09-25 16:08:00	2019-09-25 16:24:52.283	LV	Low Emissions	Commercial Agent	LZ4	2019	15.966667	2019-09-25 16:00:00	2019-09-25 16:08:00	...	2.100000	

69138 rows × 22 columns

In [57]:

```
df.corr()
```

Out[57]:

	Year	Duration	arrivals	departures	occupancy	day_of_week	bin_of_day	bin_of_week	Month	MonthStart	MonthE
Year	1.000000	0.019540	0.013371	0.004959	0.055401	0.013440	-0.017156	0.011914	-0.736718	0.030153	-0.0117
Duration	0.019540	1.000000	0.004216	-0.068227	-0.004875	-0.011933	0.034233	-0.008889	-0.002069	-0.009385	-0.0038
arrivals	0.013371	0.004216	1.000000	0.046358	0.148486	-0.018330	-0.041864	-0.022053	-0.007130	0.006860	-0.0026
departures	0.004959	-0.068227	0.046358	1.000000	0.129029	-0.002821	-0.028437	-0.005350	-0.000968	-0.005004	0.0034
occupancy	0.055401	-0.004875	0.148486	0.129029	1.000000	-0.048046	-0.121973	-0.058895	-0.043684	-0.007979	0.0050
day_of_week	0.013440	-0.011933	-0.018330	-0.002821	-0.048046	1.000000	-0.044885	0.996046	-0.023106	0.020039	0.0449
bin_of_day	-0.017156	0.034233	-0.041864	-0.028437	-0.121973	-0.044885	1.000000	0.044044	-0.019233	-0.012902	0.0032
bin_of_week	0.011914	-0.008889	-0.022053	-0.005350	-0.058895	0.996046	0.044044	1.000000	-0.024818	0.018893	0.0452
Month	-0.736718	-0.002069	-0.007130	-0.000968	-0.043684	-0.023106	-0.019233	-0.024818	1.000000	-0.031550	-0.0210
MonthStart	0.030153	-0.009385	0.006860	-0.005004	-0.007979	0.020039	-0.012902	0.018893	-0.031550	1.000000	-0.0288
MonthEnd	-0.011780	-0.003803	-0.002699	0.003420	0.005062	0.044932	0.003284	0.045226	-0.021013	-0.028893	1.0000
Occupancy_Rate	0.040231	-0.004896	0.099699	0.088855	0.754465	-0.038786	-0.107348	-0.048334	-0.034481	-0.008209	0.0023
logarithm_Dur	0.026383	0.868344	-0.004616	-0.193101	0.014793	-0.011434	0.018998	-0.009745	-0.006978	-0.004480	-0.0033

In [58]:

```
vic_weather.columns
```

Out[58]:

```
Index(['date_time', 'maxtempC', 'mintempC', 'totalSnow_cm', 'sunHour',
       'uvIndex', 'moon_illumination', 'moonrise', 'moonset', 'sunrise',
       'sunset', 'DewPointC', 'FeelsLikeC', 'HeatIndexC', 'WindChillC',
       'WindGustKmph', 'cloudcover', 'humidity', 'precipMM', 'pressure',
       'tempC', 'visibility', 'winddirDegree', 'windspeedKmph', 'location'],
      dtype='object')
```

In [59]:

```
vic_weather.rename(columns={'date_time':'TimeWeather'},inplace=True)
```

In [60]:

```
vic_weather.describe()
```

Out[60]:

	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex	moon_illumination	DewPointC	FeelsLikeC	HeatIndexC	
count	13152.000000	13152.000000	13152.000000	13152.000000	13152.000000	13152.000000	13152.000000	13152.000000	13152.000000	13
mean	20.574818	11.158759	0.003467	10.501825	4.715328	49.456204	8.186816	15.475137	15.910052	
std	8.667106	6.651249	0.058001	3.302414	1.791976	28.866060	7.434933	9.290731	8.974319	
min	3.000000	-1.000000	0.000000	3.300000	1.000000	0.000000	-14.000000	-4.000000	-1.000000	
25%	13.000000	5.000000	0.000000	8.700000	3.000000	24.750000	3.000000	8.000000	9.000000	
50%	20.000000	11.000000	0.000000	10.150000	5.000000	49.000000	9.000000	15.000000	15.000000	
75%	29.000000	17.000000	0.000000	13.825000	6.000000	74.000000	14.000000	22.000000	23.000000	
max	39.000000	26.000000	1.100000	14.500000	9.000000	100.000000	25.000000	44.000000	44.000000	

In [61]:

```
vic_weather.drop(labels=['totalSnow_cm','uvIndex','moon_illumination','DewPointC','HeatIndexC',
                         'WindChillC','WindGustKmph','pressure','visibility','winddirDegree',
                         'moonrise','moonset','sunrise','sunset','tempC','location'],axis=1,inplace=True)
vic_weather.head()
```

Out[61]:

	TimeWeather	maxtempC	mintempC	sunHour	FeelsLikeC	cloudcover	humidity	precipMM	windspeedKmph
0	2018-07-02 00:00:00	29	19	13.6	20	72	67	0.0	10
1	2018-07-02 01:00:00	29	19	13.6	20	75	67	0.0	9
2	2018-07-02 02:00:00	29	19	13.6	20	78	68	0.0	9
3	2018-07-02 03:00:00	29	19	13.6	19	82	68	0.0	8
4	2018-07-02 04:00:00	29	19	13.6	20	79	67	0.0	7

In [62]:

```
vic_weather['TimeWeather'] = pd.to_datetime(vic_weather.loc[:,'TimeWeather'])

vic_weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13152 entries, 0 to 13151
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   TimeWeather    13152 non-null  datetime64[ns]
 1   maxtempC       13152 non-null  int64
 2   mintempC       13152 non-null  int64
 3   sunHour        13152 non-null  float64
 4   FeelsLikeC     13152 non-null  int64
 5   cloudcover     13152 non-null  int64
 6   humidity       13152 non-null  int64
 7   precipMM       13152 non-null  float64
 8   windspeedKmph  13152 non-null  int64
dtypes: datetime64[ns](1), float64(2), int64(6)
memory usage: 924.9 KB
```

In [63]:

```
vic_weather.set_index('TimeWeather')
```

Out[63]:

TimeWeather	maxtempC	mintempC	sunHour	FeelsLikeC	cloudcover	humidity	precipMM	windspeedKmph
2018-07-02 00:00:00	29	19	13.6	20	72	67	0.0	10
2018-07-02 01:00:00	29	19	13.6	20	75	67	0.0	9
2018-07-02 02:00:00	29	19	13.6	20	78	68	0.0	9
2018-07-02 03:00:00	29	19	13.6	19	82	68	0.0	8
2018-07-02 04:00:00	29	19	13.6	20	79	67	0.0	7
...	...	...	...	...	...	...	...	...
2019-12-31 19:00:00	10	1	8.7	4	11	65	0.0	2
2019-12-31 20:00:00	10	1	8.7	4	8	65	0.0	1
2019-12-31 21:00:00	10	1	8.7	4	5	65	0.0	1
2019-12-31 22:00:00	10	1	8.7	4	7	66	0.0	1
2019-12-31 23:00:00	10	1	8.7	5	9	66	0.0	1

13152 rows × 8 columns

In [64]:

```
vic_weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13152 entries, 0 to 13151
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   TimeWeather    13152 non-null  datetime64[ns]
 1   maxtempC       13152 non-null  int64
 2   mintempC       13152 non-null  int64
 3   sunHour        13152 non-null  float64
 4   FeelsLikeC     13152 non-null  int64
 5   cloudcover     13152 non-null  int64
 6   humidity       13152 non-null  int64
 7   precipMM       13152 non-null  float64
 8   windspeedKmph  13152 non-null  int64
dtypes: datetime64[ns](1), float64(2), int64(6)
memory usage: 924.9 KB
```

In [117]:

```
df_w = pd.merge(df,vic_weather, left_on='TimeWeather',right_on='TimeWeather',how='left',sort=False)
```

In [118]:

```
df_w['hour']= df_w['TimeStart'].dt.hour
df_w.hour = df_w.hour.astype('object')
```

In [119]:

```
pd.set_option("display.max_columns", None)
df_w.head()
```

Out[119]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration	TimeWeather	datetime	arrivals	departures	oc
0	2018-07-02 04:51:00	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.53300	2018-07-02 04:00:00	2018-07-02 04:51:00	1.0	0.0	(
1	2018-07-02 06:20:00	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.35495	2018-07-02 06:00:00	2018-07-02 06:20:00	1.0	0.0	(
2	2018-07-02 06:25:00	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.17790	2018-07-02 06:00:00	2018-07-02 06:25:00	1.0	0.0	(
3	2018-07-02 06:34:00	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.79950	2018-07-02 06:00:00	2018-07-02 06:34:00	1.0	0.0	(
4	2018-07-02 06:48:00	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.42210	2018-07-02 06:00:00	2018-07-02 06:48:00	1.0	0.0	(

In [120]:

```
df_w.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69138 entries, 0 to 69137
Data columns (total 31 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             69138 non-null  datetime64[ns]
 1   TimeFinish            69138 non-null  datetime64[ns]
 2   VehicleType           69138 non-null  object
 3   Emission              69138 non-null  object
 4   ProfessionalActivity  69138 non-null  object
 5   ZoneId                69138 non-null  object
 6   Year                  69138 non-null  int64
 7   Duration              69138 non-null  float64
 8   TimeWeather           69138 non-null  datetime64[ns]
 9   datetime              69138 non-null  datetime64[ns]
 10  arrivals              69138 non-null  float64
 11  departures            69138 non-null  float64
 12  occupancy             69138 non-null  float64
 13  day_of_week           69138 non-null  int64
 14  dow_name              69138 non-null  object
 15  bin_of_day            69138 non-null  int64
 16  bin_of_week           69138 non-null  int64
 17  Month                 69138 non-null  int64
 18  MonthStart            69138 non-null  int64
 19  MonthEnd              69138 non-null  int64
 20  Occupancy_Rate        69138 non-null  float64
 21  logarithm_Dur         69138 non-null  float64
 22  maxtempC              69138 non-null  int64
 23  mintempC              69138 non-null  int64
 24  sunHour               69138 non-null  float64
 25  FeelsLikeC            69138 non-null  int64
 26  cloudcover            69138 non-null  int64
 27  humidity              69138 non-null  int64
 28  precipMM              69138 non-null  float64
 29  windspeedKmph         69138 non-null  int64
 30  hour                  69138 non-null  object
dtypes: datetime64[ns](4), float64(8), int64(13), object(6)
memory usage: 16.9+ MB
```

In [121]:

```
df_w.columns
```

Out[121]:

```
Index(['TimeStart', 'TimeFinish', 'VehicleType', 'Emission',
       'ProfessionalActivity', 'ZoneId', 'Year', 'Duration', 'TimeWeather',
       'datetime', 'arrivals', 'departures', 'occupancy', 'day_of_week',
       'dow_name', 'bin_of_day', 'bin_of_week', 'Month', 'MonthStart',
       'MonthEnd', 'Occupancy_Rate', 'logarithm_Dur', 'maxtempC', 'mintempC',
       'sunHour', 'FeelsLikeC', 'cloudcover', 'humidity', 'precipMM',
       'windspeedKmph', 'hour'],
      dtype='object')
```

In [ ]:

```
#https://dius.com.au/2017/08/03/using-statsmodels-glms-to-model-beverage-consumption/
#https://statmath.wu.ac.at/courses/heather_turner/index.html

model_full3 = 'Duration ~ C(hour) + C(ProfessionalActivity, Treatment(reference="Unspecified")) + C(ZoneId) + C(Vehicl
response2, predictors2 = dmatrices(model_full3, df_w, return_type='dataframe')
lmfull3    = sm.GLM(response2,predictors2, family=sm.families.Gamma(link=sm.genmod.families.links.log)).fit()
print(lmfull3.summary())
```

In [ ]:

```
lmfull3.aic
```

In [93]:

```
lmfull3.pearson_chi2 / lmfull3.df_resid
```

Out[93]:

```
0.4399430092670387
```

Queueing Modelling

In [73]:

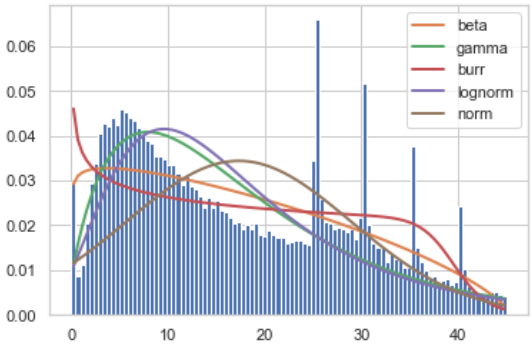
```
sns.set(style="whitegrid", font_scale=1)
```

In [74]:

```
data = df['Duration'].values

f = Fitter(data,
           distributions=['gamma',
                          'lognorm',
                          "beta",
                          "burr",
                          "norm"])
f.fit()
f.summary()
```

Out[74]:

	sumsquare_error	aic	bic	kl_div
beta	0.007771	794.278337	-1.106247e+06	0.063309
gamma	0.008926	822.623165	-1.096681e+06	0.073442
burr	0.010179	797.223877	-1.087588e+06	0.092178
lognorm	0.010941	828.790536	-1.082604e+06	0.090081
norm	0.016046	823.057776	-1.056139e+06	0.134015



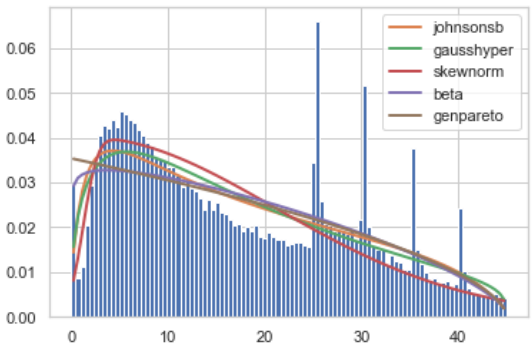
In [75]:

```
#https://fitter.readthedocs.io/en/latest/
f = Fitter(data)
f.fit()
# may take some time since by default, all distributions are tried
# but you call manually provide a smaller set of distributions
f.summary()
```

```
WARNING:root:SKIPPED burr distribution (taking more than 30 seconds)
WARNING:root:SKIPPED genexpon distribution (taking more than 30 seconds)
WARNING:root:SKIPPED genhalflogistic distribution (taking more than 30 seconds)
WARNING:root:SKIPPED geninvgauss distribution (taking more than 30 seconds)
WARNING:root:SKIPPED genlogistic distribution (taking more than 30 seconds)
WARNING:root:SKIPPED kstwo distribution (taking more than 30 seconds)
WARNING:root:SKIPPED johnsonsu distribution (taking more than 30 seconds)
WARNING:root:SKIPPED kappa4 distribution (taking more than 30 seconds)
WARNING:root:SKIPPED ksone distribution (taking more than 30 seconds)
WARNING:root:SKIPPED levy_stable distribution (taking more than 30 seconds)
WARNING:root:SKIPPED loggamma distribution (taking more than 30 seconds)
WARNING:root:SKIPPED lognorm distribution (taking more than 30 seconds)
WARNING:root:SKIPPED loguniform distribution (taking more than 30 seconds)
WARNING:root:SKIPPED lomax distribution (taking more than 30 seconds)
WARNING:root:SKIPPED mielke distribution (taking more than 30 seconds)
WARNING:root:SKIPPED ncf distribution (taking more than 30 seconds)
WARNING:root:SKIPPED nct distribution (taking more than 30 seconds)
WARNING:root:SKIPPED ncx2 distribution (taking more than 30 seconds)
WARNING:root:SKIPPED norminvgauss distribution (taking more than 30 seconds)
WARNING:root:SKIPPED pareto distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rv_continuous distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rv_histogram distribution (taking more than 30 seconds)
WARNING:root:SKIPPED pearson3 distribution (taking more than 30 seconds)
WARNING:root:SKIPPED powerlognorm distribution (taking more than 30 seconds)
WARNING:root:SKIPPED powernorm distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rdist distribution (taking more than 30 seconds)
WARNING:root:SKIPPED recipinvgauss distribution (taking more than 30 seconds)
WARNING:root:SKIPPED reciprocal distribution (taking more than 30 seconds)
WARNING:root:SKIPPED t distribution (taking more than 30 seconds)
WARNING:root:SKIPPED trapezoid distribution (taking more than 30 seconds)
WARNING:root:SKIPPED trapz distribution (taking more than 30 seconds)
WARNING:root:SKIPPED triang distribution (taking more than 30 seconds)
WARNING:root:SKIPPED truncexpon distribution (taking more than 30 seconds)
WARNING:root:SKIPPED truncnorm distribution (taking more than 30 seconds)
WARNING:root:SKIPPED tukeylambda distribution (taking more than 30 seconds)
WARNING:root:SKIPPED vonmises distribution (taking more than 30 seconds)
WARNING:root:SKIPPED vonmises_line distribution (taking more than 30 seconds)
WARNING:root:SKIPPED weibull_max distribution (taking more than 30 seconds)
WARNING:root:SKIPPED weibull_min distribution (taking more than 30 seconds)
```

Out[75]:

	sumsquare_error	aic	bic	kl_div
johnsonsb	0.006569	794.946046	-1.117869e+06	0.052837
gausshyper	0.006800	796.918419	-1.115451e+06	0.053427
skewnorm	0.007612	815.478370	-1.107689e+06	0.061448
beta	0.007771	794.278337	-1.106247e+06	0.063309
genpareto	0.008001	791.461959	-1.104241e+06	0.066338



In [76]:

```
f.fitted_param['johnsonsb']
```

Out[76]:

```
(0.46791764007186143,
 0.7200210716889017,
 -0.34296542076690706,
 46.23160397601005)
```

X,y formating for Clustering

In [77]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69138 entries, 0 to 69137
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             69138 non-null  datetime64[ns]
 1   TimeFinish            69138 non-null  datetime64[ns]
 2   VehicleType           69138 non-null  object
 3   Emission              69138 non-null  object
 4   ProfessionalActivity  69138 non-null  object
 5   ZoneId                69138 non-null  object
 6   Year                  69138 non-null  int64
 7   Duration              69138 non-null  float64
 8   datetime              69138 non-null  datetime64[ns]
 9   arrivals              69138 non-null  float64
 10  departures            69138 non-null  float64
 11  occupancy             69138 non-null  float64
 12  day_of_week           69138 non-null  int64
 13  dow_name              69138 non-null  object
 14  bin_of_day            69138 non-null  int64
 15  bin_of_week           69138 non-null  int64
 16  Month                 69138 non-null  int64
 17  MonthStart            69138 non-null  int64
 18  MonthEnd              69138 non-null  int64
 19  Occupancy_Rate        69138 non-null  float64
 20  logarithm_Dur         69138 non-null  float64
dtypes: datetime64[ns](3), float64(6), int64(7), object(5)
memory usage: 11.6+ MB
```

In [78]:

```
df.head()
```

Out[78]:

	TimeStart	TimeFinish	VehicleType	Emission	ProfessionalActivity	ZoneId	Year	Duration	datetime	arrivals	...	occupancy	day_of_week
0	2018-07-02 04:51:00	2018-07-02 04:58:47.883	LV	Medium Emissions	Commercial Agent	LZ1	2018	7.53300	2018-07-02 04:51:00	1.0	...	0.750000	0
1	2018-07-02 06:20:00	2018-07-02 06:23:41.247	LV	Low Emissions	Install & Maintenance	LZ4	2018	3.35495	2018-07-02 06:20:00	1.0	...	0.683333	0
2	2018-07-02 06:25:00	2018-07-02 06:29:42.847	LV	Low Emissions	Install & Maintenance	LZ7	2018	4.17790	2018-07-02 06:25:00	1.0	...	0.466667	0
3	2018-07-02 06:34:00	2018-07-02 06:45:57.433	LV	Low Emissions	Commercial Agent	LZ3	2018	11.79950	2018-07-02 06:34:00	1.0	...	0.850000	0
4	2018-07-02 06:48:00	2018-07-02 06:52:23.653	LV	Medium Emissions	Commercial Agent	LZ1	2018	3.42210	2018-07-02 06:48:00	1.0	...	0.033333	0

5 rows × 21 columns

In [79]:

```
X_q_categoric = df[['VehicleType','Emission','ProfessionalActivity','ZoneId','day_of_week','Month',]].values
X_q_numeric = df[['bin_of_day','Occupancy_Rate']].values
y_q = df['Duration'].values
y_q_round=np.rint(y_q)

print('Matrix shape - X categoric: ',X_q_categoric.shape)
print('Matrix shape - X numeric: ',X_q_numeric.shape)
print('Array shape - y: ',y_q.shape)
print('Array shape - y_round: ',y_q_round.shape)
```

```
Matrix shape - X categoric:  (69138, 6)
Matrix shape - X numeric:  (69138, 2)
Array shape - y:  (69138,)
Array shape - y_round:  (69138,)
```

In [81]:

```
for i in ['VehicleType','Emission','ProfessionalActivity','ZoneId','day_of_week','Month']:
  print(i,'\n',df[i].value_counts())
```

```
VehicleType
 LV         42496
Van        19826
HV          4293
Private     1901
Truck        622
Name: VehicleType, dtype: int64
Emission
 Medium Emissions    35991
Low Emissions       21314
High Emissions      11170
Hybrid                465
eVehicle              198
Name: Emission, dtype: int64
ProfessionalActivity
 Install & Maintenance    16399
Unspecified              16039
Commercial Agent         12976
Construction              6277
Local Commerce            5950
Automotive                5945
Transport and parcels     5202
Food                       350
Name: ProfessionalActivity, dtype: int64
ZoneId
 LZ4    14514
LZ5    12071
LZ2    11246
LZ1    10601
LZ7     6243
LZ3     6027
LZ8     4627
LZ6     3809
Name: ZoneId, dtype: int64
day_of_week
 1    14982
2    13536
4    12959
3    12479
0    11644
5     3538
Name: day_of_week, dtype: int64
Month
 9     8182
7     8083
8     7332
10    5589
1     5439
3     5419
11    5369
5     5337
2     5033
4     4653
6     4494
12    4208
Name: Month, dtype: int64
```

In [83]:

```
enc = OneHotEncoder(sparse=False)
X_q_categoric_onehot = enc.fit_transform(X_q_categoric)
print(X_q_categoric_onehot.shape)
print(type(X_q_categoric_onehot))
```

```
(69138, 44)
<class 'numpy.ndarray'>
```

In [91]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
X_q_numeric_minmax = scaler.fit_transform(X_q_numeric)
```

In [92]:

```
X_q_full = np.concatenate((X_q_numeric_minmax, X_q_categoric_onehot),
                          axis=1)
print(X_q_full.shape)
```

```
(69138, 46)
```

Dimensionality Reduction

In [93]:

```
pca = PCA()
transf = pca.fit_transform(X_q_full)

varianza_expl = pca.explained_variance_ratio_

print(varianza_expl)
```

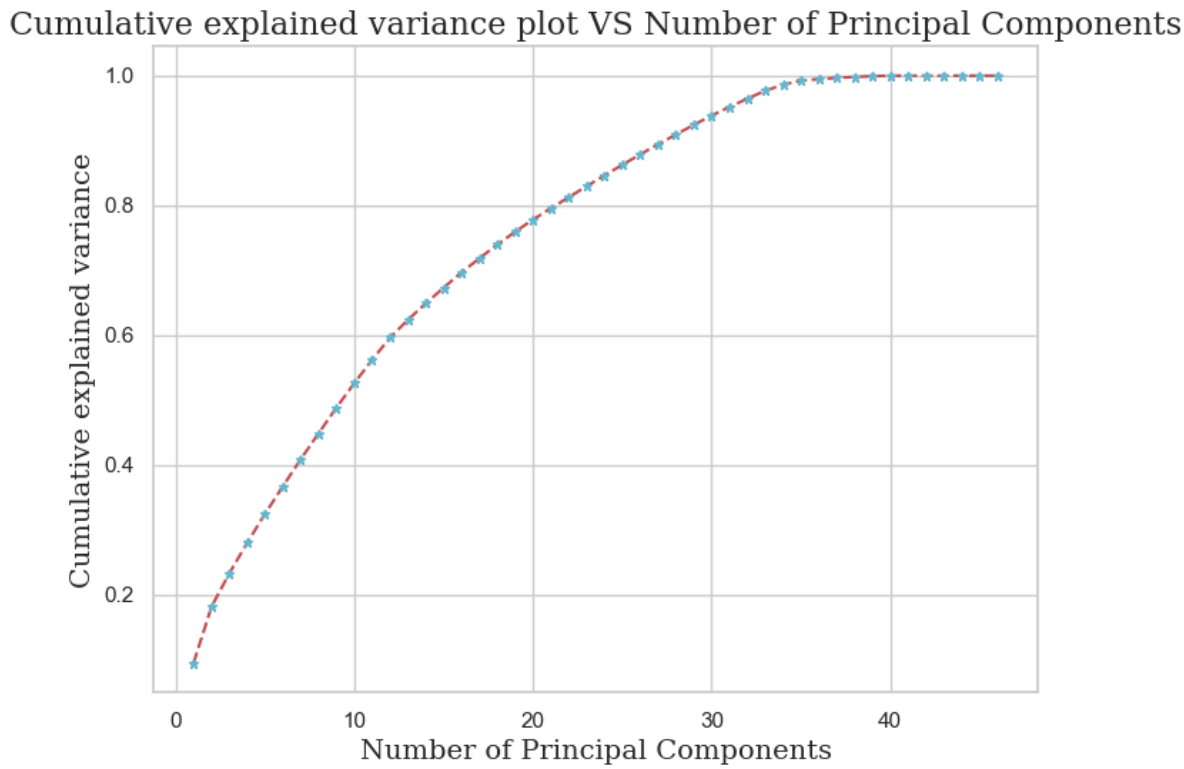
```
[9.44132704e-02 8.66616838e-02 5.15928104e-02 4.69922482e-02
 4.49254607e-02 4.23676925e-02 4.18160642e-02 3.98677679e-02
 3.96826247e-02 3.76956560e-02 3.61317865e-02 3.39968762e-02
 2.74123427e-02 2.54765105e-02 2.40624427e-02 2.36283434e-02
 2.16098024e-02 2.12526705e-02 1.96844130e-02 1.88358466e-02
 1.73865361e-02 1.70228062e-02 1.69315431e-02 1.68377122e-02
 1.62837153e-02 1.60105768e-02 1.56176707e-02 1.50329687e-02
 1.49088715e-02 1.42897214e-02 1.34144049e-02 1.27129734e-02
 1.22953967e-02 9.10847241e-03 6.38132770e-03 2.29527729e-03
 1.96800051e-03 1.42692022e-03 1.24174231e-03 7.27049285e-04
 2.09828712e-32 6.13861168e-33 5.00844236e-33 1.40931682e-33
 4.45680320e-34 3.25211394e-34]
```

In [87]:

```
def cumulative_explained_variance_plot(expl_variance):

  cum_var_exp = np.cumsum(expl_variance)

  plt.figure(dpi = 100, figsize = (8, 6))
  plt.title('Cumulative explained variance plot VS Number of Principal Components',
            fontdict= dict(family ='serif', size = 16))
  plt.xlabel('Number of Principal Components',
             fontdict= dict(family ='serif', size = 14))
  plt.ylabel('Cumulative explained variance',
             fontdict= dict(family ='serif', size = 14))

  nc = np.arange(1, expl_variance.shape[0] + 1)

  plt.plot(nc, cum_var_exp, '--r')
  plt.plot(nc, cum_var_exp, 'c*', ms = 5)
  plt.show()
```

In [94]:

```
cumulative_explained_variance_plot(varianza_expl)
```



In [95]:

```
np.cumsum(varianza_expl)
```

Out[95]:

```
array([0.09441327, 0.18107495, 0.23266776, 0.27966001, 0.32458547,
       0.36695317, 0.40876923, 0.448637  , 0.48831962, 0.52601528,
       0.56214707, 0.59614394, 0.62355628, 0.64903279, 0.67309524,
       0.69672358, 0.71833338, 0.73958605, 0.75927047, 0.77810631,
       0.79549285, 0.81251566, 0.8294472 , 0.84628491, 0.86256863,
       0.8785792 , 0.89419687, 0.90922984, 0.92413871, 0.93842844,
       0.95184284, 0.96455581, 0.97685121, 0.98595968, 0.99234101,
       0.99463629, 0.99660429, 0.99803121, 0.99927295, 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        ])
```

In [96]:

```
pca = PCA(n_components= 22)
X_q_pca = pca.fit_transform(X_q_full)
pca.components_
```

Out[96]:

```
array([[ 2.13239685e-02, -4.33733566e-03,  2.79397199e-02, ...,
        -8.26228223e-03, -4.10221547e-03, -3.54683982e-03],
       [ 7.03859559e-03, -6.26519398e-04, -1.56704643e-02, ...,
        -1.17919146e-02, -1.17170993e-02, -9.41312910e-03],
       [ 1.30962427e-02,  3.69026571e-03,  9.03592700e-03, ...,
         3.11176982e-03, -1.52335154e-02, -1.37809281e-03],
       ...,
       [-1.60519696e-03,  5.40411047e-02,  1.48295611e-01, ...,
        -1.73323756e-01, -6.79633120e-02, -4.60413424e-02],
       [-1.01371741e-01,  1.33041398e-02, -2.01591262e-02, ...,
         8.34917434e-01, -1.63733701e-01, -6.39735834e-02],
       [ 1.73703148e-02, -1.44602215e-03,  1.03153869e-01, ...,
         1.00666912e-01, -6.21617776e-03, -6.47674692e-04]])
```

In [97]:

```
X_q_pca.shape
```

Out[97]:

```
(69138, 22)
```

Clustering for durations in queueing analysis

In [98]:

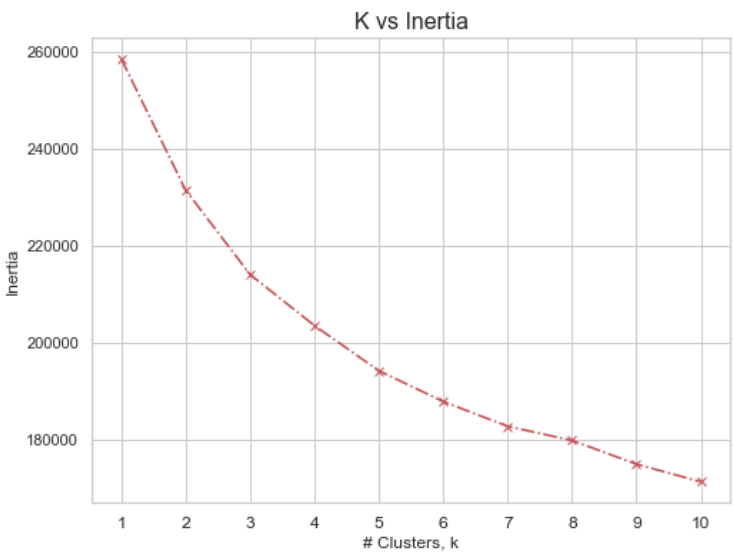
```
X_q_pca
```

Out[98]:

```
array([[-5.77955966e-01, -5.23897016e-01, -3.14291430e-01, ...,
        -4.44937693e-02,  8.94287721e-03, -1.71389302e-02],
       [-4.58642547e-01,  1.09271878e+00,  9.30175629e-01, ...,
         1.01018889e-02,  2.15317611e-02, -1.97921643e-04],
       [-3.86204767e-01,  1.00723675e+00,  4.96763139e-01, ...,
        -2.97695744e-01, -9.98337578e-02,  5.75473179e-02],
       ...,
       [-4.11527663e-01, -4.29877502e-01, -1.69407703e-02, ...,
         4.10683891e-02, -3.30926433e-02, -1.18369901e-02],
       [ 8.34834975e-01, -8.10114813e-01,  3.43559789e-01, ...,
        -4.12591776e-02, -1.25206020e-01, -5.43422102e-02],
       [-3.32071289e-01,  9.56324786e-01,  7.67337885e-02, ...,
         5.21339419e-02, -2.69114885e-02, -3.92265711e-03]])
```

In [100]:

```
sum_sq_d = []
K = range(1,11)

for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X_q_pca)
    sum_sq_d.append(km.inertia_)

plt.figure(figsize=(8,6))

plt.plot(K, sum_sq_d, 'rx-.')

plt.xlabel('# Clusters, k', fontsize=12)
plt.xticks(range(1,11), fontsize=12)

plt.ylabel('Inertia', fontsize=12)
plt.xticks(fontsize=12)

plt.title('K vs Inertia', fontsize=16)

plt.show()
```



In [101]:

```
def plot_metric(K, scores, metric_name):
  plt.figure(dpi=110, figsize=(9, 5))
  plt.plot(K, scores, 'bx-')
  plt.xticks(K); plt.xlabel('$k$', fontdict=dict(family = 'serif', size = 14));  plt.ylabel(metric_name, fontdict=dict
  plt.title(f'K vs {metric_name}', fontdict=dict(family = 'serif', size = 18))
  plt.show()
```

In [102]:

```
from sklearn.metrics import silhouette_score

silhouette_scores = []

K = range(2,10)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X_q_pca)
    y_cl = km.predict(X_q_pca)
    silhouette_scores.append(silhouette_score(X_q_pca, y_cl))

plot_metric(K, silhouette_scores, 'Coeficiente de silueta')
```

In [103]:

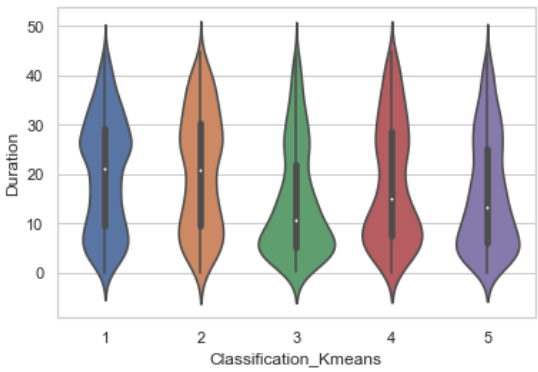
```
k = 5

kmeans = KMeans(n_clusters=k, init='k-means++')
kmeans.fit(X_q_pca)

labels = kmeans.predict(X_q_pca)

centroids = kmeans.cluster_centers_
df['Classification_Kmeans'] = pd.Series(labels+1, index=df.index)
```

In [104]:

```
sns.violinplot(x='Classification_Kmeans',y='Duration', data=df[df['ZoneId']=='LZ8'])
```

Out[104]:

```
<AxesSubplot:xlabel='Classification_Kmeans', ylabel='Duration'>
```

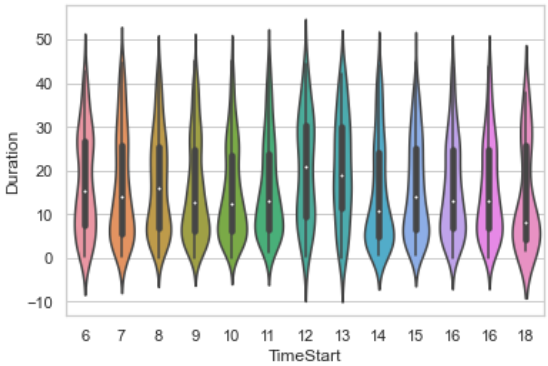


In [105]:

```
sns.violinplot(df.TimeStart.dt.hour,y='Duration',order=[6,7,8,9,10,11,12,13,14,15,16,16,18], data=df[(df['ZoneId']=='L
```

Out[105]:

```
<AxesSubplot:xlabel='TimeStart', ylabel='Duration'>
```



Clustering without LZ category

In [106]:

```
X_q_categoric_LZ = df[['VehicleType','Emission','ProfessionalActivity','day_of_week','Month']].values
X_q_numeric_LZ = df[['bin_of_day','Occupancy_Rate']].values
y_q_LZ = df['Duration'].values
y_q_round_LZ=np.rint(y_q_LZ)

print('Matrix shape - X categoric: ',X_q_categoric_LZ.shape)
print('Matrix shape - X numeric: ',X_q_numeric_LZ.shape)
print('Array shape - y: ',y_q_LZ.shape)
print('Array shape - y_round: ',y_q_round_LZ.shape)
```

```
Matrix shape - X categoric:  (69138, 5)
Matrix shape - X numeric:  (69138, 2)
Array shape - y:  (69138,)
Array shape - y_round:  (69138,)
```

In [108]:

```
X_q_categoric_onehot_LZ = enc.fit_transform(X_q_categoric_LZ)
print(X_q_categoric_onehot_LZ.shape)
print(type(X_q_categoric_onehot_LZ))
```

```
(69138, 36)
<class 'numpy.ndarray'>
```

In [109]:

```
X_q_numeric_minmax_LZ = scaler.fit_transform(X_q_numeric_LZ)
```

In [110]:

```
X_q_full_LZ = np.concatenate((X_q_numeric_minmax_LZ, X_q_categoric_onehot_LZ),
                             axis=1)
print(X_q_full_LZ.shape)
```

```
(69138, 38)
```

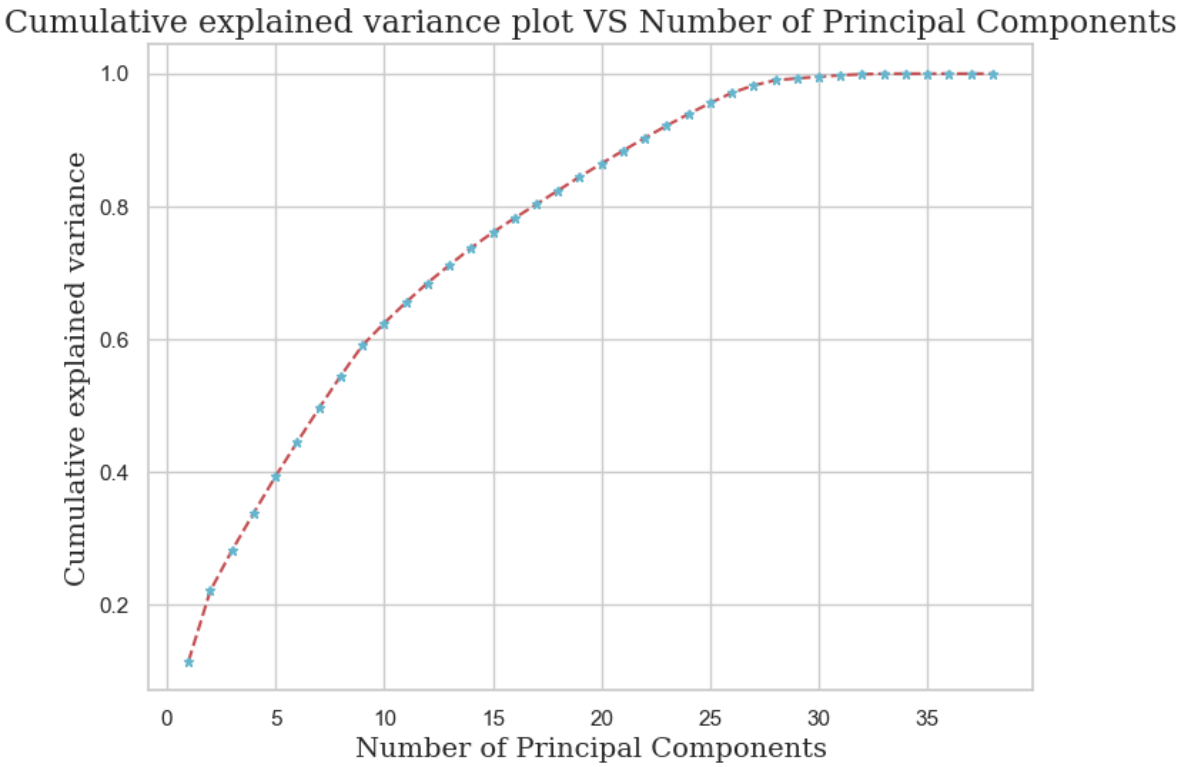
In [114]:

```
pca_q = PCA()
transf2 = pca_q.fit_transform(X_q_full_LZ)

varianza_expl2 = pca_q.explained_variance_ratio_

print(varianza_expl2)
```

```
[1.15487050e-01 1.05821130e-01 6.08921229e-02 5.63991507e-02
 5.49411684e-02 5.17202663e-02 5.07447786e-02 4.87630604e-02
 4.59081875e-02 3.35423050e-02 3.12663218e-02 2.94964476e-02
 2.65317210e-02 2.60907253e-02 2.33180001e-02 2.13962640e-02
 2.08823214e-02 2.07847741e-02 2.06711321e-02 1.99356470e-02
 1.94689559e-02 1.92462482e-02 1.83070507e-02 1.75534637e-02
 1.64680199e-02 1.53940368e-02 1.12188018e-02 7.84365446e-03
 2.82069825e-03 2.41713647e-03 2.23675450e-03 1.52728165e-03
 9.05324214e-04 2.27758046e-32 7.52256335e-33 6.64134572e-33
 3.03146030e-33 2.60139716e-33]
```

In [115]:

```
cumulative_explained_variance_plot(varianza_expl2)
```



In [116]:

```
np.cumsum(varianza_expl2)
```

Out[116]:

```
array([0.11548705, 0.22130818, 0.2822003 , 0.33859945, 0.39354062,
       0.44526089, 0.49600567, 0.54476873, 0.59067691, 0.62421922,
       0.65548554, 0.68498199, 0.71151371, 0.73760443, 0.76092243,
       0.7823187 , 0.80320102, 0.82398579, 0.84465693, 0.86459257,
       0.88406153, 0.90330778, 0.92161483, 0.93916829, 0.95563631,
       0.97103035, 0.98224915, 0.9900928 , 0.9929135 , 0.99533064,
       0.99756739, 0.99909468, 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        ])
```

In [117]:

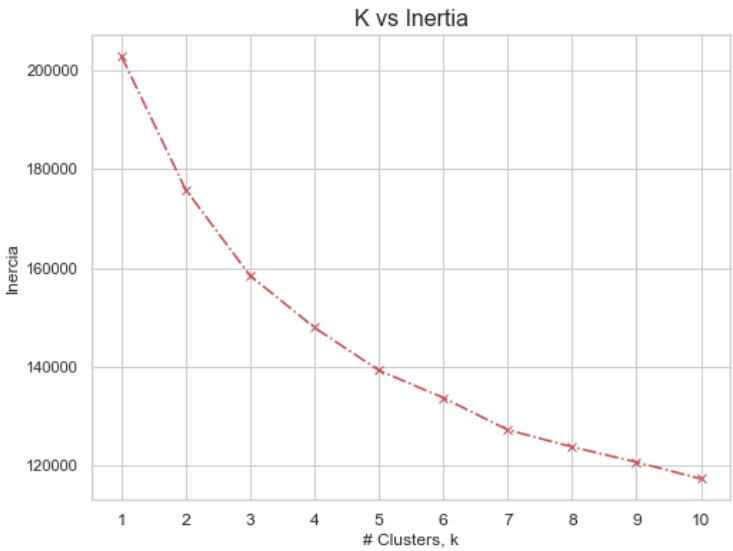
```
pca_q = PCA(n_components= 16)
X_q_pca_LZ = pca_q.fit_transform(X_q_full_LZ)
pca_q.components_
```

Out[117]:

```
array([[ 2.19213482e-02, -4.58940509e-03,  2.79035648e-02,
        -6.68169843e-01,  2.67505095e-02,  4.11704022e-03,
         6.09398729e-01,  2.16654043e-02, -1.51232572e-03,
         1.82774882e-01, -2.01766119e-01, -1.16184099e-03,
         1.37319612e-02, -8.35305115e-02,  2.71954842e-02,
        -1.91807294e-03, -2.20496731e-01,  1.21235293e-01,
        -3.17237336e-02,  1.75506311e-01,  4.15857926e-02,
        -1.97557379e-02, -2.01209965e-03, -9.21085786e-03,
        -6.54157926e-03, -4.06551799e-03,  1.91808437e-03,
         8.27658801e-04, -3.39766931e-03, -1.87196539e-03,
         3.06756481e-03,  2.86101466e-03, -4.59703495e-04,
         9.05014172e-03,  5.97353169e-03, -9.11474719e-03,
        -4.89173926e-03, -3.96217141e-03],
       [ 5.47810267e-03,  4.17999648e-04, -1.71636218e-02,
         1.81571250e-01, -1.68277658e-02, -1.82116570e-03,
        -1.45758697e-01,  1.18042706e-01,  3.81985270e-03,
         6.05961128e-01, -7.29354165e-01,  1.53047868e-03,
        -4.28264157e-02,  7.65057910e-03,  1.95783245e-02,
```

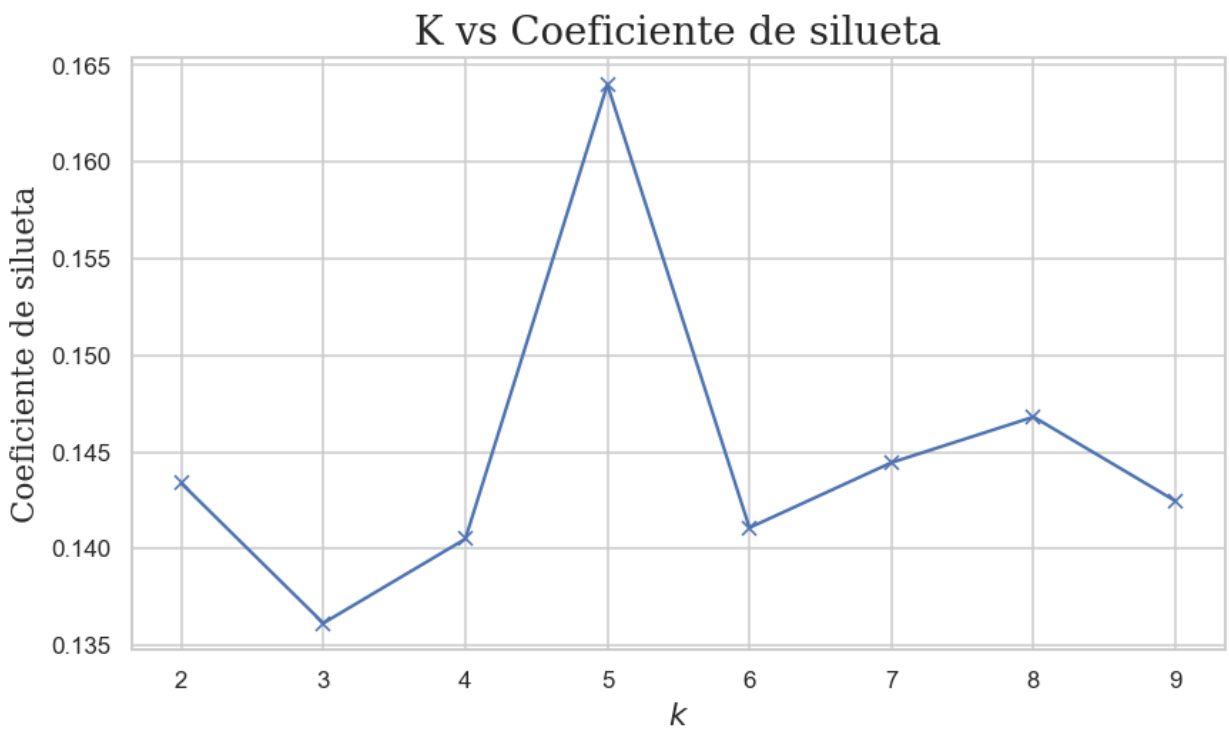
In [118]:

```
sum_sq_d = []
K = range(1,11)

for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X_q_pca_LZ)
    sum_sq_d.append(km.inertia_)

plt.figure(figsize=(8,6))

plt.plot(K, sum_sq_d, 'rx-.')

plt.xlabel('# Clusters, k', fontsize=12)
plt.xticks(range(1,11), fontsize=12)

plt.ylabel('Inercia', fontsize=12)
plt.xticks(fontsize=12)

plt.title('K vs Inertia', fontsize=16)

plt.show()
```



In [119]:

```
silhouette_scores = []

K = range(2,10)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(X_q_pca_LZ)
    y_cl_LZ = km.predict(X_q_pca_LZ)
    silhouette_scores.append(silhouette_score(X_q_pca_LZ, y_cl_LZ))

plot_metric(K, silhouette_scores, 'Coeficiente de silueta')
```



In [120]:

```
k = 5

kmeans = KMeans(n_clusters=k, init='k-means++')
kmeans.fit(X_q_pca_LZ)

labels = kmeans.predict(X_q_pca_LZ)

centroids = kmeans.cluster_centers_
df['Classification_Kmeans_q_LZ'] = pd.Series(labels+1, index=df.index)
```

In [142]:

```
df[df['ZoneId']=='LZ4'].Classification_Kmeans_q_LZ.value_counts()/len(df[df['ZoneId']=='LZ4'])
```

Out[142]:

```
1    0.327201
5    0.233430
2    0.161086
4    0.145170
3    0.133113
Name: Classification_Kmeans_q_LZ, dtype: float64
```

In [129]:

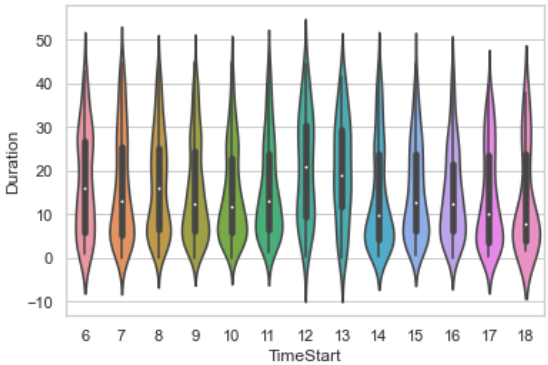
```
sns.set(style="whitegrid", font_scale=1)
```

In [134]:

```
sns.violinplot(df.TimeStart.dt.hour,y='Duration',order=[6,7,8,9,10,11,12,13,14,15,16,17,18], data=df[(df['ZoneId']=='L
```

Out[134]:

```
<AxesSubplot:xlabel='TimeStart', ylabel='Duration'>
```



In [159]:

```
data_LZ_cl = df[(df['ZoneId']=='LZ4')&(df['Classification_Kmeans_q_LZ']==5)&(df['TimeStart'].dt.hour==14)]
data2 = data_LZ_cl['Duration'].values
```

In [160]:

```
f = Fitter(data2)
f.fit()
# may take some time since by default, all distributions are tried
# but you call manually provide a smaller set of distributions
f.summary()
```

```
WARNING:root:SKIPPED kstwo distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rv_continuous distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rv_histogram distribution (taking more than 30 seconds)
WARNING:root:SKIPPED levy_stable distribution (taking more than 30 seconds)
```

Out[160]:

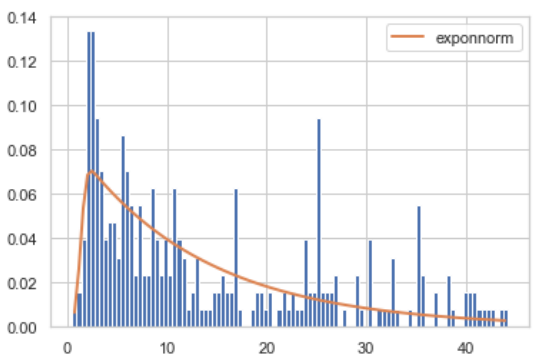
	sumsquare_error	aic	bic	kl_div
laplace_asymmetric	0.034731	852.201660	-2611.709508	inf
exponnorm	0.035739	851.019579	-2603.385822	inf
fatiguelife	0.036221	863.711206	-2599.492853	inf
recipinvgauss	0.037028	855.887301	-2593.078765	inf
levy	0.037669	926.830977	-2593.758370	inf

In [161]:

```
f = Fitter(data2,
           distributions=['exponnorm'])
f.fit()
f.summary()
```

Out[161]:

	sumsquare_error	aic	bic	kl_div
exponnorm	0.035739	851.019579	-2603.385822	inf



In [158]:

```
f.fitted_param['exponnorm']
```

Out[158]:

```
(0.41765927033912176,
 0.8460698405640245,
 -0.8575525740691228,
 46.157345056701296)
```

MACHINE LEARNING MODELLING

Data Partitioning

In [96]:

```
df_w.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69138 entries, 0 to 69137
Data columns (total 31 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   TimeStart             69138 non-null  datetime64[ns]
 1   TimeFinish            69138 non-null  datetime64[ns]
 2   VehicleType           69138 non-null  object
 3   Emission              69138 non-null  object
 4   ProfessionalActivity  69138 non-null  object
 5   ZoneId                69138 non-null  object
 6   Year                  69138 non-null  int64
 7   Duration              69138 non-null  float64
 8   TimeWeather           69138 non-null  datetime64[ns]
 9   datetime              69138 non-null  datetime64[ns]
 10  arrivals              69138 non-null  float64
 11  departures            69138 non-null  float64
 12  occupancy             69138 non-null  float64
 13  day_of_week           69138 non-null  int64
 14  dow_name              69138 non-null  object
 15  bin_of_day            69138 non-null  int64
 16  bin_of_week           69138 non-null  int64
 17  Month                 69138 non-null  int64
 18  MonthStart            69138 non-null  int64
 19  MonthEnd              69138 non-null  int64
 20  Occupancy_Rate        69138 non-null  float64
 21  logarithm_Dur         69138 non-null  float64
 22  maxtempC              69138 non-null  int64
 23  mintempC              69138 non-null  int64
 24  sunHour               69138 non-null  float64
 25  FeelsLikeC            69138 non-null  int64
 26  cloudcover            69138 non-null  int64
 27  humidity              69138 non-null  int64
 28  precipMM              69138 non-null  float64
 29  windspeedKmph         69138 non-null  int64
 30  hour                  69138 non-null  object
dtypes: datetime64[ns](4), float64(8), int64(13), object(6)
memory usage: 16.9+ MB
```

In [97]:

```
df_w.columns
```

Out[97]:

```
Index(['TimeStart', 'TimeFinish', 'VehicleType', 'Emission',
       'ProfessionalActivity', 'ZoneId', 'Year', 'Duration', 'TimeWeather',
       'datetime', 'arrivals', 'departures', 'occupancy', 'day_of_week',
       'dow_name', 'bin_of_day', 'bin_of_week', 'Month', 'MonthStart',
       'MonthEnd', 'Occupancy_Rate', 'logarithm_Dur', 'maxtempC', 'mintempC',
       'sunHour', 'FeelsLikeC', 'cloudcover', 'humidity', 'precipMM',
       'windspeedKmph', 'hour'],
      dtype='object')
```

In [98]:

```
X_categoric = df_w[['VehicleType','Emission','ProfessionalActivity','ZoneId','day_of_week','Month','hour']].values
X_numeric = df_w[['FeelsLikeC', 'precipMM']].values
y = df_w['Duration'].values
y_round=np.rint(y)

print('Matrix shape - X categoric: ',X_categoric.shape)
print('Matrix shape - X numeric: ',X_numeric.shape)
print('Array shape - y: ',y.shape)
print('Array shape - y_round: ',y_round.shape)
```

```
Matrix shape - X categoric:  (69138, 7)
Matrix shape - X numeric:  (69138, 2)
Array shape - y:  (69138,)
Array shape - y_round:  (69138,)
```

In [99]:

```
enc = OneHotEncoder(sparse=False)
X_categoric_onehot = enc.fit_transform(X_categoric)
print(X_categoric_onehot.shape)
print(type(X_categoric_onehot))
```

```
(69138, 59)
<class 'numpy.ndarray'>
```

In [100]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
X_numeric_minmax = scaler.fit_transform(X_numeric)
```

In [101]:

```
X_full = np.concatenate((X_numeric_minmax, X_categoric_onehot),
                        axis=1)
print(X_full.shape)
```

```
(69138, 61)
```

In [102]:

```
X_train, X_test, y_train, y_test = train_test_split(X_full, y_round, test_size=0.2)
print("Training matrix shape: ",X_train.shape,"Test matrix shape: ", X_test.shape)
print("Training durations: ",y_train.shape,"Test durations: ", y_test.shape)
```

```
Training matrix shape:  (55310, 61) Test matrix shape:  (13828, 61)
Training durations:  (55310,) Test durations:  (13828,)
```

Linear Regression (Base Model)

In [157]:

```
reg_full = linear_model.LinearRegression()
reg_full.fit(X_train, y_train)
```

Out[157]:

```
LinearRegression()
```

In [158]:

```
y_pred_train_LR = reg_full.predict(X_train)
y_pred_test_LR = reg_full.predict(X_test)
```

In [159]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_LR))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_LR))
```

```
MAE training 9.590124694901464
MAE testing 9.580269652155048
```

In [160]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_LR))
print('MSE testing',mean_squared_error(y_test, y_pred_test_LR))
```

```
MSE training 127.36521979983559
MSE testing 126.66406670201539
```

In [161]:

```
mean_squared_error(y_train, y_pred_train_LR)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_LR)))

mean_squared_error(y_test, y_pred_test_LR)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_LR)))
```

```
RMSE training 11.285620044988029
RMSE testing 11.254513170369272
```

In [162]:

```
print('R2 training',r2_score(y_train, y_pred_train_LR))
print('R2 testing', r2_score(y_test, y_pred_test_LR))
```

```
R2 training 0.05178365113670691
R2 testing 0.06003700156878322
```

In [113]:

```
def smape(A, F):
    return 100/len(A) * np.sum(2 * np.abs(F - A) / (np.abs(A) + np.abs(F)))
```

In [164]:

```
print('SMAPE training',smape(y_pred_train_LR, y_train),
      'SMAPE testing',smape( y_pred_test_LR,y_test))
```

```
SMAPE training 62.21297407139296 SMAPE testing 62.28820790857754
```

In [165]:

```
scores = cross_val_score(reg_full,
                         X_train,
                         y_train, scoring = 'r2',
                         cv = 5)
```

In [166]:

```
print(f'Splits R2:\n{list(scores)}')
print(f'Average R2: {np.mean(scores)}')
```

```
Splits R2:
[0.06636552337572565, 0.07906931322932997, 0.0672139406356913, 0.0805485188842332, 0.07203938748578154]
Average R2: 0.07304733672215233
```

In [105]:

```
std_slc0 = StandardScaler()
pca0 = PCA()
lin_reg1 = linear_model.LinearRegression()

pipe0 = Pipeline(steps=[('std_slc', std_slc0),
                        ('pca', pca0),
                        ('lin_reg', lin_reg1)])

n_components = list(range(1,X_full.shape[1]+1,1))


parameters0 = dict(pca__n_components=n_components)
```

In [106]:

```
lr_RS = RandomizedSearchCV(pipe0, parameters0,n_jobs=-1)
```

In [124]:

```
y_train1=stats.yeojohnson(y_train+0.001)
```

In [125]:

```
y_test1=stats.yeojohnson(y_test+0.001)
```

In [126]:

```
len(y_train1)
```

Out[126]:

```
2
```

In [107]:

```
lr_RS.fit(X_train, y_train)
```

Out[107]:

```
RandomizedSearchCV(estimator=Pipeline(steps=[('std_slc', StandardScaler()),
                                             ('pca', PCA()),
                                             ('lin_reg', LinearRegression())]),
                   n_jobs=-1,
                   param_distributions={'pca__n_components': [1, 2, 3, 4, 5, 6,
                                                               7, 8, 9, 10, 11,
                                                               12, 13, 14, 15,
                                                               16, 17, 18, 19,
                                                               20, 21, 22, 23,
                                                               24, 25, 26, 27,
                                                               28, 29, 30, ...]})
```

In [108]:

```
y_pred_train_LRRS = lr_RS.predict(X_train)
y_pred_test_LRRS = lr_RS.predict(X_test)
```

In [109]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_LRRS))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_LRRS))
```

```
MAE training 9.437113722618204
MAE testing 9.45665938465448
```

In [110]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_LRRS))
print('MSE testing',mean_squared_error(y_test, y_pred_test_LRRS))
```

```
MSE training 123.4678294236536
MSE testing 124.07763749581719
```

In [111]:

```
mean_squared_error(y_train, y_pred_train_LRRS)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_LRRS)))

mean_squared_error(y_test, y_pred_test_LRRS)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_LRRS)))
```

```
RMSE training 11.111607868515412
RMSE testing 11.139014206644015
```

In [112]:

```
mean_squared_error(y_train, y_pred_train_LRRS)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_LRRS)))

mean_squared_error(y_test, y_pred_test_LRRS)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_LRRS)))
```

```
RMSE training 11.111607868515412
RMSE testing 11.139014206644015
```

In [113]:

```
print('R2 training',r2_score(y_train, y_pred_train_LRRS))
print('R2 testing', r2_score(y_test, y_pred_test_LRRS))
```

```
R2 training 0.08113737593753556
R2 testing 0.07787642595637045
```

In [116]:

```
print('SMAPE training',smape(y_pred_train_LRRS, y_train),
      'SMAPE testing',smape( y_pred_test_LRRS,y_test))
```

```
SMAPE training 61.49071655352062 SMAPE testing 61.605386892443356
```

In [117]:

```
lr_RS.best_params_
```

Out[117]:

```
{'pca__n_components': 55}
```

Decision Trees

In [118]:

```
#https://www.dezyre.com/recipes/optimize-hyper-parameters-of-decisiontree-model-using-grid-search-in-python
#https://www.nbshare.io/notebook/312837011/Decision-Tree-Regression-With-Hyper-Parameter-Tuning-In-Python/

std_slc2 = StandardScaler()
pca2 = PCA()
dec_tree2 = DecisionTreeRegressor()

pipe2 = Pipeline(steps=[('std_slc', std_slc2),
                        ('pca', pca2),
                        ('dec_tree', dec_tree2)])

n_components = list(range(1,X_full.shape[1]+1,1))

splitter = ["best","random"]
max_depth = [2,4,6,8,12,16,24,30]
min_samples_leaf = [1,2,3,4,5,6,7,8,9,10]
min_weight_fraction_leaf = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
max_features = ["auto","log2","sqrt",None]
max_leaf_nodes = [None,10,20,30,40,50,60,70,80,90]

parameters = dict(pca__n_components=n_components,
                  dec_tree__splitter=splitter,
                  dec_tree__max_depth=max_depth,
                  dec_tree__min_samples_leaf = min_samples_leaf,
                  dec_tree__min_weight_fraction_leaf = min_weight_fraction_leaf,
                  dec_tree__max_features = max_features,
                  dec_tree__max_leaf_nodes = max_leaf_nodes)
```

In [119]:

```
dt_RS = RandomizedSearchCV(pipe2,parameters,n_jobs=-1)
```

In [120]:

```
dt_RS.fit(X_train,y_train)
```

Out[120]:

```
RandomizedSearchCV(estimator=Pipeline(steps=[('std_slc', StandardScaler()),
                                             ('pca', PCA()),
                                             ('dec_tree',
                                              DecisionTreeRegressor())]),
                   n_jobs=-1,
                   param_distributions={'dec_tree__max_depth': [2, 4, 6, 8, 12,
                                                                16, 24, 30],
                                        'dec_tree__max_features': ['auto',
                                                                   'log2',
                                                                   'sqrt',
                                                                   None],
                                        'dec_tree__max_leaf_nodes': [None, 10,
                                                                     20, 30, 40,
                                                                     50, 60, 70,
                                                                     80, 90],
                                        'dec_tree__min_samples_leaf': [1, 2, 3,
                                                                       4, 5, 6,
                                                                       7, 8, 9,
                                                                       10],
                                        'dec_tree__min_weight_fraction_leaf': [0.1,
                                                                               0.2,
                                                                               0.3,
                                                                               0.4,
                                                                               0.5,
                                                                               0.6,
                                                                               0.7,
                                                                               0.8,
                                                                               0.9],
                                        'dec_tree__splitter': ['best',
                                                               'random'],
                                        'pca__n_components': [1, 2, 3, 4, 5, 6,
                                                              7, 8, 9, 10, 11,
                                                              12, 13, 14, 15,
                                                              16, 17, 18, 19,
                                                              20, 21, 22, 23,
                                                              24, 25, 26, 27,
                                                              28, 29, 30, ...]})
```

In [121]:

```
best_dt_RS = dt_RS.best_estimator_
```

In [122]:

```
y_pred_train_DTRS = best_dt_RS.predict(X_train)
y_pred_test_DTRS = best_dt_RS.predict(X_test)
```

In [123]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_DTRS))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_DTRS))
```

```
MAE training 9.862544071954245
MAE testing 9.858957865408334
```

In [124]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_DTRS))
print('MSE testing',mean_squared_error(y_test, y_pred_test_DTRS))
```

```
MSE training 131.03793570008338
MSE testing 131.09859237247738
```

In [125]:

```
mean_squared_error(y_train, y_pred_train_DTRS)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_DTRS)))

mean_squared_error(y_test, y_pred_test_DTRS)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_DTRS)))
```

```
RMSE training 11.44718025105237
RMSE testing 11.449829359972025
```

In [126]:

```
print('R2 training',r2_score(y_train, y_pred_train_DTRS))
print('R2 testing', r2_score(y_test, y_pred_test_DTRS))
```

```
R2 training 0.024799723043968047
R2 testing 0.025697901810283308
```

In [127]:

```
print('SMAPE training',smape(y_pred_train_DTRS, y_train),
      'SMAPE testing',smape( y_pred_test_DTRS,y_test))
```

```
SMAPE training 63.5584807316417 SMAPE testing 63.589958617088406
```

In [128]:

```
dt_RS.best_params_
```

Out[128]:

```
{'pca__n_components': 22,
 'dec_tree__splitter': 'best',
 'dec_tree__min_weight_fraction_leaf': 0.2,
 'dec_tree__min_samples_leaf': 6,
 'dec_tree__max_leaf_nodes': 30,
 'dec_tree__max_features': None,
 'dec_tree__max_depth': 8}
```

Random Forest

In [189]:

```
randomF = RandomForestRegressor()
randomF.fit(X_train, y_train)
```

Out[189]:

```
RandomForestRegressor()
```

In [190]:

```
y_pred_train_RF = randomF.predict(X_train)
y_pred_test_RF = randomF.predict(X_test)
```

In [191]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_RF))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_RF))
```

```
MAE training 3.38702462168499
MAE testing 9.040163075494394
```

In [192]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_RF))
print('MSE testing',mean_squared_error(y_test, y_pred_test_RF))
```

```
MSE training 17.39423916259951
MSE testing 120.52063890586857
```

In [193]:

```
mean_squared_error(y_train, y_pred_train_RF)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_RF)))

mean_squared_error(y_test, y_pred_test_RF)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_RF)))
```

```
RMSE training 4.170640138228125
RMSE testing 10.978189236202324
```

In [194]:

```
print('R2 training',r2_score(y_train, y_pred_train_RF))
print('R2 testing', r2_score(y_test, y_pred_test_RF))
```

```
R2 training 0.8705023084329
R2 testing 0.10562684375738851
```

In [129]:

```
std_slc3 = StandardScaler()
pca3 = PCA()
random_forest2 = RandomForestRegressor()

pipe3 = Pipeline(steps=[('std_slc', std_slc3),
                        ('pca', pca3),
                        ('rand_forest', random_forest2)])

n_components = list(range(1,X_full.shape[1]+1,1))

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 150, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(2, 24, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

parameters = dict(pca__n_components=n_components,
                  rand_forest__n_estimators=n_estimators,
                  rand_forest__max_features=max_features,
                  rand_forest__max_depth = max_depth,
                  rand_forest__min_samples_split =min_samples_split,
                  rand_forest__min_samples_leaf = min_samples_leaf,
                  rand_forest__bootstrap = bootstrap)
```

In [130]:

```
rf_RS = RandomizedSearchCV(pipe3, parameters, n_iter = 100, cv = 3, verbose=2, n_jobs = -1)
```

In [131]:

```
rf_RS.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[131]:

```
RandomizedSearchCV(cv=3,
                   estimator=Pipeline(steps=[('std_slc', StandardScaler()),
                                             ('pca', PCA()),
                                             ('rand_forest',
                                              RandomForestRegressor())]),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'pca__n_components': [1, 2, 3, 4, 5, 6,
                                                              7, 8, 9, 10, 11,
                                                              12, 13, 14, 15,
                                                              16, 17, 18, 19,
                                                              20, 21, 22, 23,
                                                              24, 25, 26, 27,
                                                              28, 29, 30, ...],
                                        'rand_forest__bootstrap': [True, False],
                                        'rand_forest__max_depth': [2, 4, 6, 8,
                                                                   10, 13, 15,
                                                                   17, 19, 21,
                                                                   24, None],
                                        'rand_forest__max_features': ['auto',
                                                                      'sqrt'],
                                        'rand_forest__min_samples_leaf': [1, 2,
                                                                          4],
                                        'rand_forest__min_samples_split': [2, 5,
                                                                           10],
                                        'rand_forest__n_estimators': [10, 25,
                                                                      41, 56,
                                                                      72, 87,
                                                                      103, 118,
                                                                      134,
                                                                      150]},
                   verbose=2)
```

In [134]:

```
rf_RS.best_params_
```

Out[134]:

```
{'rand_forest__n_estimators': 118,
 'rand_forest__min_samples_split': 2,
 'rand_forest__min_samples_leaf': 4,
 'rand_forest__max_features': 'sqrt',
 'rand_forest__max_depth': 13,
 'rand_forest__bootstrap': False,
 'pca__n_components': 52}
```

In [135]:

```
best_random = rf_RS.best_estimator_
```

In [136]:

```
y_pred_train_RFRS = best_random.predict(X_train)
y_pred_test_RFRS = best_random.predict(X_test)
```

In [137]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_RFRS))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_RFRS))
```

```
MAE training 7.704172957132895
MAE testing 9.263997014066511
```

In [138]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_RFRS))
print('MSE testing',mean_squared_error(y_test, y_pred_test_RFRS))
```

```
MSE training 83.06625203605202
MSE testing 119.84778100694713
```

In [139]:

```
mean_squared_error(y_train, y_pred_train_RFRS)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_RFRS)))

mean_squared_error(y_test, y_pred_test_RFRS)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_RFRS)))
```

```
RMSE training 9.1140689066987
RMSE testing 10.947501130712302
```

In [140]:

```
print('R2 training',r2_score(y_train, y_pred_train_RFRS))
print('R2 testing', r2_score(y_test, y_pred_test_RFRS))
```

```
R2 training 0.3818108354769669
R2 testing 0.10931198889848381
```

In [141]:

```
print('SMAPE training',smape(y_pred_train_RFRS, y_train),
      'SMAPE testing',smape( y_pred_test_RFRS,y_test))
```

```
SMAPE training 53.44326971605828 SMAPE testing 60.614918609756174
```

In [142]:

```
rf_RS.best_params_
```

Out[142]:

```
{'rand_forest__n_estimators': 118,
 'rand_forest__min_samples_split': 2,
 'rand_forest__min_samples_leaf': 4,
 'rand_forest__max_features': 'sqrt',
 'rand_forest__max_depth': 13,
 'rand_forest__bootstrap': False,
 'pca__n_components': 52}
```

XGBoost

In [143]:

```
import sys
print(sys.base_prefix)
```

```
/Users/juanpabl/opt/anaconda3
```

In [144]:

```
conda install -c conda-forge py-xgboost
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /Users/juanpabl/opt/anaconda3

  added / updated specs:
    - py-xgboost


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.13.0               |   py38h50d1736_1         989 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         989 KB

The following packages will be UPDATED:

  conda                               4.11.0-py38h50d1736_0 --> 4.13.0-py38h50d1736_1



Downloading and Extracting Packages
conda-4.13.0         | 989 KB    | ##################################### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

Note: you may need to restart the kernel to use updated packages.
```

In [145]:

```
import xgboost as xgb # XGBoost library

def XGBmodel(x_train, x_val, y_train, y_val):
    matrix_train = xgb.DMatrix(x_train, label=y_train)
    matrix_val = xgb.DMatrix(x_val, label=y_val)
    model=xgb.train(params={'objective':'reg:gamma','eval_metric':'rmse'},
                    dtrain=matrix_train,
                    num_boost_round=500,
                    early_stopping_rounds=90,
                    evals=[(matrix_val,'validation')],)
    return model
```

In [146]:

```
model=XGBmodel(X_train, X_test, y_train, y_test)
```

```
[0]	validation-rmse:20.27463
[1]	validation-rmse:20.09158
[2]	validation-rmse:19.85234
[3]	validation-rmse:19.54303
[4]	validation-rmse:19.14887
[5]	validation-rmse:18.65558
[6]	validation-rmse:18.05254
[7]	validation-rmse:17.33763
[8]	validation-rmse:16.52223
[9]	validation-rmse:15.63328
[10]	validation-rmse:14.71944
[11]	validation-rmse:13.84011
[12]	validation-rmse:13.06038
[13]	validation-rmse:12.41407
[14]	validation-rmse:11.92751
[15]	validation-rmse:11.58452
[16]	validation-rmse:11.36023
[17]	validation-rmse:11.22457
[18]	validation-rmse:11.13329
```

In [147]:

```
y_pred_train_xg = model.predict(xgb.DMatrix(X_train), ntree_limit = model.best_ntree_limit)
y_pred_test_xg = model.predict(xgb.DMatrix(X_test), ntree_limit = model.best_ntree_limit)
```

In [148]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_xg))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_xg))
```

```
MAE training 8.914976906718236
MAE testing 9.223523922831397
```

In [149]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_xg))
print('MSE testing',mean_squared_error(y_test, y_pred_test_xg))
```

```
MSE training 113.21713354579069
MSE testing 120.8405962645373
```

In [150]:

```
mean_squared_error(y_train, y_pred_train_xg)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_xg)))

mean_squared_error(y_test, y_pred_test_xg)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_xg)))
```

```
RMSE training 10.64035401411958
RMSE testing 10.99275198776618
```

In [151]:

```
print('R2 training',r2_score(y_train, y_pred_train_xg))
print('R2 testing', r2_score(y_test, y_pred_test_xg))
```

```
R2 training 0.1574243031214595
R2 testing 0.10193355736020648
```

In [152]:

```
print('SMAPE training',smape(y_pred_train_xg, y_train),
      'SMAPE testing',smape( y_pred_test_xg,y_test))
```

```
SMAPE training 58.71175602363137 SMAPE testing 60.38802914487616
```

In [153]:

```
xgb.XGBRegressor()
```

Out[153]:

```
XGBRegressor(base_score=None, booster=None, colsample_bylevel=None,
             colsample_bynode=None, colsample_bytree=None, gamma=None,
             gpu_id=None, importance_type='gain', interaction_constraints=None,
             learning_rate=None, max_delta_step=None, max_depth=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             random_state=None, reg_alpha=None, reg_lambda=None,
             scale_pos_weight=None, subsample=None, tree_method=None,
             validate_parameters=None, verbosity=None)
```

In [154]:

```
xgb2 = xgb.XGBRegressor()
std_slc4 = StandardScaler()
pca4 = PCA()

pipe4 = Pipeline(steps=[('std_slc', std_slc4),
                        ('pca', pca4),
                        ('xgb', xgb2)])

n_components = list(range(1,X_full.shape[1]+1,1))

learning_rate = [0.17,0.2,0.21,0.25,0.3,0.35]
min_child_weight = [3,4, 5,7, 9]
gamma = [1,5,7,8,9,10]
subsample = [.5,.6,.7, .8,.9]
colsample_bytree = [.8, .9, 1]
max_depth = [2,3,4,5,6]

parameters = dict(pca__n_components=n_components,
                  xgb__learning_rate=learning_rate,
                  xgb__min_child_weight=min_child_weight,
                  xgb__gamma = gamma,
                  xgb__subsample =subsample,
                  xgb__colsample_bytree = colsample_bytree,
                  xgb__max_depth = max_depth)
```

In [155]:

```
xgb_RS = RandomizedSearchCV(pipe4, parameters, verbose=2, n_jobs = -1)
```

In [156]:

```
xgb_RS.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[156]:

```
RandomizedSearchCV(estimator=Pipeline(steps=[('std_slc', StandardScaler()),
                                             ('pca', PCA()),
                                             ('xgb',
                                              XGBRegressor(base_score=None,
                                                           booster=None,
                                                           colsample_bylevel=None,
                                                           colsample_bynode=None,
                                                           colsample_bytree=None,
                                                           gamma=None,
                                                           gpu_id=None,
                                                           importance_type='gain',
                                                           interaction_constraints=None,
                                                           learning_rate=None,
                                                           max_delta_step=None,
                                                           max_depth=None,
                                                           min_child_weight=...
                   param_distributions={'pca__n_components': [1, 2, 3, 4, 5, 6,
                                                              7, 8, 9, 10, 11,
                                                              12, 13, 14, 15,
                                                              16, 17, 18, 19,
                                                              20, 21, 22, 23,
                                                              24, 25, 26, 27,
                                                              28, 29, 30, ...],
                                        'xgb__colsample_bytree': [0.8, 0.9, 1],
                                        'xgb__gamma': [1, 5, 7, 8, 9, 10],
                                        'xgb__learning_rate': [0.17, 0.2, 0.21,
                                                               0.25, 0.3,
                                                               0.35],
                                        'xgb__max_depth': [2, 3, 4, 5, 6],
                                        'xgb__min_child_weight': [3, 4, 5, 7,
                                                                  9],
                                        'xgb__subsample': [0.5, 0.6, 0.7, 0.8,
                                                           0.9]},
                   verbose=2)
```

In [157]:

```
best_xgb_RS = xgb_RS.best_estimator_
```

In [158]:

```
xgb_RS.best_params_
```

Out[158]:

```
{'xgb__subsample': 0.7,
 'xgb__min_child_weight': 4,
 'xgb__max_depth': 3,
 'xgb__learning_rate': 0.3,
 'xgb__gamma': 7,
 'xgb__colsample_bytree': 1,
 'pca__n_components': 51}
```

In [159]:

```
y_pred_train_xgRS = best_xgb_RS.predict(X_train)
y_pred_test_xgRS = best_xgb_RS.predict(X_test)
```

In [160]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_xgRS))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_xgRS))
```

```
MAE training 9.025423436922658
MAE testing 9.373683989160075
```

In [161]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_xgRS))
print('MSE testing',mean_squared_error(y_test, y_pred_test_xgRS))
```

```
MSE training 114.74733876806378
MSE testing 123.21240411585474
```

In [162]:

```
mean_squared_error(y_train, y_pred_train_xgRS)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_xgRS)))

mean_squared_error(y_test, y_pred_test_xgRS)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_xgRS)))
```

```
RMSE training 10.71201842642477
RMSE testing 11.10010829297871
```

In [251]:

```
print('R2 training',r2_score(y_train, y_pred_train_xgRS))
print('R2 testing', r2_score(y_test, y_pred_test_xgRS))
```

```
R2 training 0.1921614402611147
R2 testing 0.10359785718224424
```

In [163]:

```
print('SMAPE training',smape( y_pred_train_xgRS, y_train),
      'SMAPE testing',smape( y_pred_test_xgRS,y_test))
```

```
SMAPE training 59.442007045321425 SMAPE testing 61.1170753146781
```

CatBoost

https://towardsdatascience.com/catboost-regression-in-6-minutes-3487f3e5b329 (https://towardsdatascience.com/catboost-regression-in-6-minutes-3487f3e5b329)

In [77]:

```
pip install catboost
```

```
Requirement already satisfied: catboost in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages (0.2
6.1)
Requirement already satisfied: scipy in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages (from c
atboost) (1.6.2)
Requirement already satisfied: graphviz in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages (fro
m catboost) (0.17)
Requirement already satisfied: matplotlib in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages (f
rom catboost) (3.3.4)
Requirement already satisfied: numpy>=1.16.0 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages
(from catboost) (1.20.1)
Requirement already satisfied: pandas>=0.24.0 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-package
s (from catboost) (1.2.4)
Requirement already satisfied: plotly in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages (from
catboost) (5.3.1)
Requirement already satisfied: six in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages (from cat
boost) (1.15.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site
-packages (from pandas>=0.24.0->catboost) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages
(from pandas>=0.24.0->catboost) (2021.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /Users/juanpabl/opt/anaconda3/
lib/python3.8/site-packages (from matplotlib->catboost) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-pack
ages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages
(from matplotlib->catboost) (8.2.0)
Requirement already satisfied: cycler>=0.10 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packages
(from matplotlib->catboost) (0.10.0)
Requirement already satisfied: tenacity>=6.2.0 in /Users/juanpabl/opt/anaconda3/lib/python3.8/site-packag
es (from plotly->catboost) (8.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [93]:

```
import catboost as cb
```

In [477]:

```
df_X = df[{'VehicleType','Emission','ProfessionalActivity','ZoneId','day_of_week','hour','Month'}]
```

In [478]:

```
df_y= df['Duration'].values
```

In [78]:

```
df_w.columns
```

Out[78]:

```
Index(['TimeStart', 'TimeFinish', 'VehicleType', 'Emission',
       'ProfessionalActivity', 'ZoneId', 'Year', 'Duration', 'TimeWeather',
       'datetime', 'arrivals', 'departures', 'occupancy', 'day_of_week',
       'dow_name', 'bin_of_day', 'bin_of_week', 'Month', 'MonthStart',
       'MonthEnd', 'Occupancy_Rate', 'logarithm_Dur', 'maxtempC', 'mintempC',
       'sunHour', 'FeelsLikeC', 'cloudcover', 'humidity', 'precipMM',
       'windspeedKmph', 'hour'],
      dtype='object')
```

In [79]:

```
features = ['VehicleType','Emission','ProfessionalActivity','ZoneId','day_of_week','hour','Month','FeelsLikeC',
            'precipMM']
cat_features = ['VehicleType','Emission','ProfessionalActivity','ZoneId','day_of_week','hour','Month']
target = 'Duration'
df_CB = df_w[features + [target]]
df_train, df_test = train_test_split(df_CB)
```

In [125]:

```
df_train
```

Out[125]:

	VehicleType	Emission	ProfessionalActivity	ZoneId	day_of_week	hour	Month	FeelsLikeC	precipMM	Duration
54202	Van	High Emissions	Automotive	LZ7	4	8	6	27	0.0	13.612383
58088	LV	Medium Emissions	Install & Maintenance	LZ1	2	9	7	25	0.0	4.655767
12553	Van	Medium Emissions	Unspecified	LZ4	1	7	10	17	0.0	17.799767
19676	LV	Medium Emissions	Unspecified	LZ2	1	8	11	5	0.3	28.830217
30905	LV	Medium Emissions	Automotive	LZ5	1	8	1	3	0.0	14.977333
...	...	...	...	...	...	...	...	...	...	...
30806	Van	Medium Emissions	Unspecified	LZ1	0	15	1	4	0.2	30.192450
40849	LV	Medium Emissions	Commercial Agent	LZ7	0	8	3	15	0.0	5.135383
41529	LV	Medium Emissions	Commercial Agent	LZ4	3	8	3	9	0.0	13.156667
28397	Van	Low Emissions	Local Commerce	LZ3	1	8	1	4	0.0	23.757100
48984	LV	Medium Emissions	Unspecified	LZ3	2	9	5	10	0.4	18.058333

51853 rows × 10 columns

In [128]:

```
from catboost import CatBoostRegressor, Pool

train_pool = Pool(df_train[features], label=df_train[target],
                  cat_features=cat_features)
test_pool = Pool(df_test[features], label=df_test[target],
                 cat_features=cat_features)
```

In [129]:

```
cb_tweedie = CatBoostRegressor(loss_function='Tweedie:variance_power=1.9', n_estimators=500, silent=True)
cb_tweedie.fit(train_pool, eval_set=test_pool)
```

Out[129]:

```
<catboost.core.CatBoostRegressor at 0x7fda7d73abb0>
```

In [143]:

```
cb_final = CatBoostRegressor(nan_mode = 'Min', eval_metric='RMSE',iterations= 200,
 sampling_frequency= 'PerTree',
 leaf_estimation_method= 'Newton',
 grow_policy= 'SymmetricTree',
 penalties_coefficient= 1,
 boosting_type= 'Plain',
 model_shrink_mode= 'Constant',
 feature_border_type= 'GreedyLogSum',
 #bayesian_matrix_reg= 0.10000000149011612,
 l2_leaf_reg= 6,
 random_strength= 1,
 rsm= 1,
 boost_from_average= True,
 model_size_reg= 0.5,
 #pool_metainfo_options= {'tags': {}},
 subsample= 0.800000011920929,
 use_best_model= False,
 random_seed= 0,
 depth= 10,
 posterior_sampling= False,
 border_count= 254,
 #classes_count= 0,
 #auto_class_weights= 'None',
 sparse_features_conflict_fraction= 0,
 leaf_estimation_backtracking= 'AnyImprovement',
 best_model_min_trees= 1,
 model_shrink_rate= 0,
 min_data_in_leaf= 1,
 loss_function= 'RMSE',
 learning_rate= 0.10000000149011612,
 score_function= 'Cosine',
 task_type= 'CPU',
 leaf_estimation_iterations= 1,
 bootstrap_type= 'MVS',
 max_leaves= 1024)
```

In [144]:

```
cb_final.fit(train_pool, eval_set=test_pool)
```

```
0:	learn: 11.5357187	test: 11.4967768	best: 11.4967768 (0)	total: 74.2ms	remaini
ng: 14.8s
1:	learn: 11.4579129	test: 11.4193114	best: 11.4193114 (1)	total: 201ms	remaini
ng: 19.9s
2:	learn: 11.3975931	test: 11.3598592	best: 11.3598592 (2)	total: 263ms	remaini
ng: 17.3s
3:	learn: 11.3538347	test: 11.3151546	best: 11.3151546 (3)	total: 290ms	remaini
ng: 14.2s
4:	learn: 11.2995979	test: 11.2610111	best: 11.2610111 (4)	total: 362ms	remaini
ng: 14.1s
5:	learn: 11.2601592	test: 11.2212184	best: 11.2212184 (5)	total: 420ms	remaini
ng: 13.6s
6:	learn: 11.2317745	test: 11.1934424	best: 11.1934424 (6)	total: 477ms	remaini
ng: 13.1s
7:	learn: 11.2127703	test: 11.1744140	best: 11.1744140 (7)	total: 489ms	remaini
ng: 11.7s
8:	learn: 11.1775473	test: 11.1443113	best: 11.1443113 (8)	total: 556ms	remaini
ng: 11.8s
9:	learn: 11.1579940	test: 11.1265831	best: 11.1265831 (9)	total: 606ms	remaini
```

In [145]:

```
CB_pred_train0 = cb_final.predict(train_pool)
CB_pred_test0 = cb_final.predict(test_pool)
```

In [146]:

```
test_pool
```

Out[146]:

```
<catboost.core.Pool at 0x7fdb1dd57dc0>
```

In [147]:

```
from catboost.utils import eval_metric

eval_metric(df_test[target].to_numpy(), CB_pred_test0, 'MAE')
```

Out[147]:

```
[9.08191896801512]
```

In [148]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test0, 'RMSE')
```

Out[148]:

```
[10.812070507040799]
```

In [149]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test0, 'R2')
```

Out[149]:

```
[0.12720255918995393]
```

In [150]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test0, 'SMAPE')
```

Out[150]:

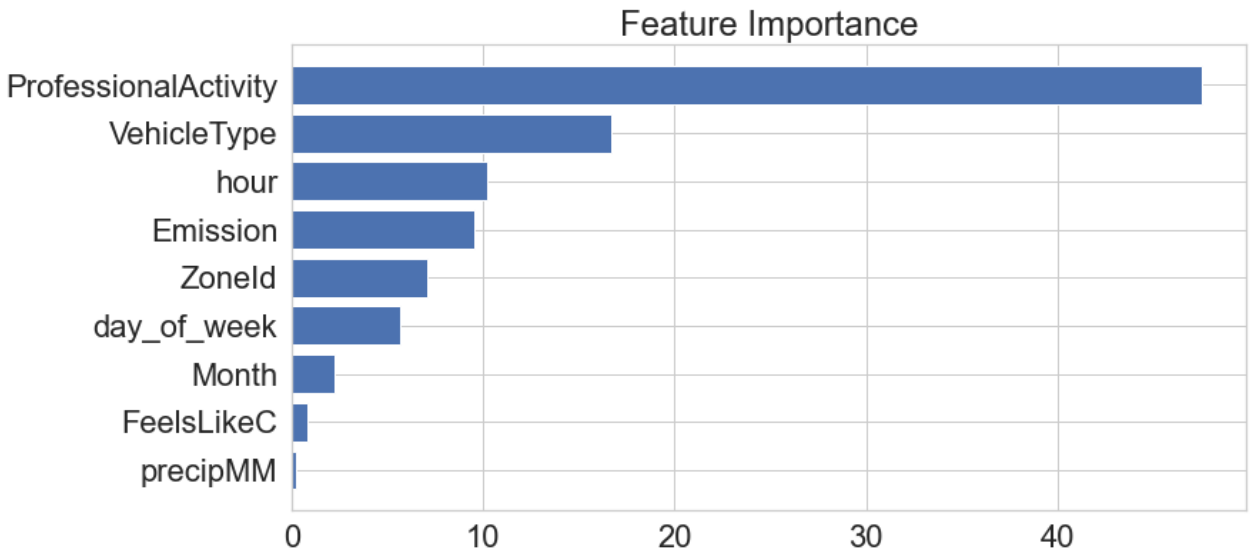
```
[59.50777148094466]
```

In [151]:

```
feature_importance = cb_tweedie.feature_importances_
sorted_idx = np.argsort(feature_importance)
fig = plt.figure(figsize=(12, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), np.array(df_train.columns)[sorted_idx])
plt.title('Feature Importance')
```

Out[151]:

```
Text(0.5, 1.0, 'Feature Importance')
```



In [137]:

```
sns.set(style="whitegrid", font_scale=2)
```

Random search

In [82]:

```
from scipy import stats

params_distribution = {
    'learning_rate': stats.uniform(0.01, 0.1),
    'depth': list(range(3, 10)),
    'l2_leaf_reg': stats.uniform(1, 10),
    'boosting_type': ['Ordered', 'Plain'],

}

cb_tweedie_model_rs = CatBoostRegressor(loss_function='RMSE')
cb_tweedie_results_rs = cb_tweedie_model_rs.randomized_search(
    params_distribution,
    train_pool,
    n_iter=20,
    verbose=5,
    partition_random_seed=123)
```

```
0:      learn: 19.3555547       test: 19.4509140        best: 19.4509140 (0)    total: 77.5ms   remaini
ng: 1m 17s
1:      learn: 18.0562671       test: 18.1528238        best: 18.1528238 (1)    total: 94.9ms   remaini
ng: 47.4s
2:      learn: 16.9486864       test: 17.0460000        best: 17.0460000 (2)    total: 109ms    remaini
ng: 36.2s
3:      learn: 16.0103894       test: 16.1082040        best: 16.1082040 (3)    total: 122ms    remaini
ng: 30.4s
4:      learn: 15.2205711       test: 15.3184482        best: 15.3184482 (4)    total: 138ms    remaini
ng: 27.4s
5:      learn: 14.5600235       test: 14.6578351        best: 14.6578351 (5)    total: 150ms    remaini
ng: 24.8s
6:      learn: 14.0106469       test: 14.1078445        best: 14.1078445 (6)    total: 167ms    remaini
ng: 23.7s
7:      learn: 13.5564563       test: 13.6529496        best: 13.6529496 (7)    total: 179ms    remaini
ng: 22.2s
8:      learn: 13.1830543       test: 13.2786796        best: 13.2786796 (8)    total: 191ms    remaini
ng: 21s
9:      learn: 12.8773919       test: 12.9719580        best: 12.9719580 (9)    total: 204ms    remaini
```

In [83]:

```
cb_tweedie_results_rs['params']
```

Out[83]:

```
{'depth': 8,
 'learning_rate': 0.05884606200007537,
 'l2_leaf_reg': 9.141341319053957,
 'boosting_type': 'Ordered'}
```

In [84]:

```
cb_tweedie_rs = CatBoostRegressor(depth= 8, learning_rate = 0.05884606200007537, l2_leaf_reg = 9.141341319053957,
 boosting_type= 'Ordered',loss_function='RMSE', n_estimators=500, silent=True)
cb_tweedie_rs.fit(train_pool, eval_set=test_pool)
```

Out[84]:

```
<catboost.core.CatBoostRegressor at 0x7fda7fd8cb50>
```

In [85]:

```
CB_pred_train_rs = cb_tweedie_rs.predict(train_pool)
CB_pred_test_rs = cb_tweedie_rs.predict(test_pool)
```

In [88]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test_rs, 'MAE')
```

Out[88]:

```
[9.075403043339582]
```

In [89]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test_rs, 'RMSE')
```

Out[89]:

```
[10.80393411064953]
```

In [90]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test_rs, 'R2')
```

Out[90]:

```
[0.12851567566078714]
```

In [91]:

```
eval_metric(df_test[target].to_numpy(), CB_pred_test_rs, 'SMAPE')
```

Out[91]:

```
[59.48890015585724]
```

Grid search

In [103]:

```
train_dataset = cb.Pool(X_train, y_train)
test_dataset = cb.Pool(X_test, y_test)
```

In [104]:

```
CB = cb.CatBoostRegressor(loss_function='RMSE')
```

In [105]:

```
grid = {'iterations': [200, 350, 500],
        'learning_rate': [0.1, 0.2, 0.3],
        'depth': [8, 10, 12, 14],
        'l2_leaf_reg': [3, 4.5, 6, 7.5]}
CB.grid_search(grid, train_dataset)
```

```
0:      learn: 19.4178291       test: 19.3305231        best: 19.3305231 (0)    total: 8.67ms   remaini
ng: 1.72s
1:      learn: 18.1716819       test: 18.0830185        best: 18.0830185 (1)    total: 18.2ms   remaini
ng: 1.8s
2:      learn: 17.0776052       test: 16.9895516        best: 16.9895516 (2)    total: 26.6ms   remaini
ng: 1.75s
3:      learn: 16.1517954       test: 16.0648024        best: 16.0648024 (3)    total: 35ms     remaini
ng: 1.71s
4:      learn: 15.3622520       test: 15.2755579        best: 15.2755579 (4)    total: 41.5ms   remaini
ng: 1.62s
5:      learn: 14.6672455       test: 14.5798353        best: 14.5798353 (5)    total: 53.7ms   remaini
ng: 1.74s
6:      learn: 14.0860162       test: 13.9969512        best: 13.9969512 (6)    total: 67.5ms   remaini
ng: 1.86s
7:      learn: 13.5897542       test: 13.4986741        best: 13.4986741 (7)    total: 83.4ms   remaini
ng: 2s
8:      learn: 13.1858383       test: 13.0931501        best: 13.0931501 (8)    total: 90.8ms   remaini
ng: 1.93s
9:      learn: 12.8428409       test: 12.7497089        best: 12.7497089 (9)    total: 97.4ms   remaini
```

In [106]:

```
CB.get_all_params()''
```

Out[106]:

```
{'nan_mode': 'Min',
 'eval_metric': 'RMSE',
 'iterations': 200,
 'sampling_frequency': 'PerTree',
 'leaf_estimation_method': 'Newton',
 'grow_policy': 'SymmetricTree',
 'penalties_coefficient': 1,
 'boosting_type': 'Plain',
 'model_shrink_mode': 'Constant',
 'feature_border_type': 'GreedyLogSum',
 'bayesian_matrix_reg': 0.10000000149011612,
 'l2_leaf_reg': 6,
 'random_strength': 1,
 'rsm': 1,
 'boost_from_average': True,
 'model_size_reg': 0.5,
 'pool_metainfo_options': {'tags': {}},
 'subsample': 0.800000011920929,
 'use_best_model': False,
 'random_seed': 0,
 'depth': 10,
 'posterior_sampling': False,
 'border_count': 254,
 'classes_count': 0,
 'auto_class_weights': 'None',
 'sparse_features_conflict_fraction': 0,
 'leaf_estimation_backtracking': 'AnyImprovement',
 'best_model_min_trees': 1,
 'model_shrink_rate': 0,
 'min_data_in_leaf': 1,
 'loss_function': 'RMSE',
 'learning_rate': 0.10000000149011612,
 'score_function': 'Cosine',
 'task_type': 'CPU',
 'leaf_estimation_iterations': 1,
 'bootstrap_type': 'MVS',
 'max_leaves': 1024}
```

In [107]:

```
CB_pred_train = CB.predict(X_train)
CB_pred_test = CB.predict(X_test)
```

In [108]:

```
print('MAE training',mean_absolute_error(y_train, CB_pred_train))
print('MAE testing',mean_absolute_error(y_test, CB_pred_test))
```

```
MAE training 8.514410854271915
MAE testing 9.059739084336188
```

In [109]:

```
print('MSE training',mean_squared_error(y_train, CB_pred_train))
print('MSE testing',mean_squared_error(y_test, CB_pred_test))
```

```
MSE training 103.35023431026485
MSE testing 117.26374533976781
```

In [110]:

```
mean_squared_error(y_train, CB_pred_train)

print('RMSE training', np.sqrt(mean_squared_error(y_train, CB_pred_train)))

mean_squared_error(y_test, CB_pred_test)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, CB_pred_test)))
```

```
RMSE training 10.166131727961469
RMSE testing 10.8288385960715
```

In [111]:

```
print('R2 training',r2_score(y_train, CB_pred_train))
print('R2 testing', r2_score(y_test, CB_pred_test))
```

```
R2 training 0.23152767621218384
R2 testing 0.12531086010791737
```

In [114]:

```
print('SMAPE training',smape(CB_pred_train, y_train),
      'SMAPE testing',smape(CB_pred_test,y_test))
```

```
SMAPE training 57.027504003816624 SMAPE testing 58.97175360065646
```

In [115]:

```
feature_importance = CB.get_feature_importance(type='PredictionValuesChange',prettified=False)
feature_importance
```

Out[115]:

```
array([6.71767600e+00, 2.55127728e+00, 1.66657213e+00, 3.95118076e+00,
       2.46215780e+00, 2.54691004e-01, 4.27142922e+00, 2.44702180e+00,
       1.44584202e-01, 4.12442057e+00, 3.92042238e+00, 3.43181059e-01,
       2.45856752e+00, 3.70013613e+00, 3.69472290e+00, 1.22422579e-01,
       5.02953095e+00, 2.73136810e+00, 5.65510957e+00, 9.21454968e+00,
       1.81000139e+00, 1.19526822e+00, 6.88423444e-01, 2.21021969e+00,
       1.68849688e+00, 7.38360923e-01, 7.41897668e-01, 1.28334801e+00,
       1.45828271e+00, 2.07261561e+00, 9.54776258e-01, 1.46255557e+00,
       1.53315583e+00, 1.14435055e+00, 1.36769433e-01, 2.64304246e-01,
       4.05200972e-01, 3.42304626e-01, 3.13728201e-01, 5.33915449e-01,
       5.48344042e-01, 6.77480414e-01, 5.59562667e-01, 6.64582687e-01,
       4.24834321e-01, 4.00121110e-01, 2.35188879e-03, 1.18045324e-01,
       4.42916316e-01, 1.27549601e+00, 1.09432098e+00, 7.02068448e-01,
       8.41330163e-01, 5.42753356e-01, 6.42157609e-01, 6.65450243e-01,
       8.13438741e-01, 9.44417118e-01, 1.07104209e+00, 8.48466353e-01,
       2.81822820e-01])
```

In [ ]:

```
feature_importance = CB.feature_importances_
sorted_idx = np.argsort(feature_importance)
fig = plt.figure(figsize=(12, 6))
plt.barh(range(len(sorted_idx)), feature_importance[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), np.array(df_train.columns)[sorted_idx])
plt.title('Feature Importance')
```

In [ ]:

```
X_train
```

In [ ]:

```
pd.DataFrame({'feature_importance': CB.get_feature_importance(CB_pred_train),
              'feature_names': CB_pred_train.columns}).sort_values(by=['feature_importance'], ascending=False)
```

In [ ]:

```
object_importance = CB.get_object_importance(train_dataset,
                      train_dataset,
                      top_size=-1,
                      type='Average',
                      update_method='SinglePoint',
                      importance_values_sign='All',
                      thread_count=-1,
                      verbose=False,
                      log_cout=sys.stdout,
                      log_cerr=sys.stderr)
object_importance
```

In [123]:

```
feature_importance2 = CB.get_feature_importance(type='Interaction',prettified=True)
feature_importance2.head(40)
```

Out[123]:

	First Feature Index	Second Feature Index	Interaction
0	0	1	0.939572
1	0	9	0.741483
2	0	6	0.733218
3	0	10	0.602920
4	0	20	0.585683
5	0	3	0.574796
6	2	16	0.469574
7	0	7	0.458914
8	0	18	0.453173
9	0	12	0.452202
10	0	49	0.438985
11	0	28	0.433600
12	0	23	0.427062
13	0	29	0.423459
14	0	16	0.421951
15	0	19	0.411428
16	6	10	0.407966
17	0	32	0.386877
18	1	10	0.381112
19	1	6	0.380485
20	18	19	0.380382
21	4	16	0.378277
22	3	14	0.372618
23	0	14	0.370888
24	1	3	0.360736
25	0	31	0.357240
26	6	18	0.352806
27	9	16	0.352148
28	0	27	0.350283
29	0	13	0.338735
30	1	9	0.333070
31	13	16	0.315691
32	0	17	0.313124
33	4	17	0.307358
34	1	7	0.304642
35	12	52	0.303300
36	0	24	0.300100
37	3	13	0.299710
38	0	50	0.294528
39	6	16	0.292053

In [ ]:

```
X_categoric[61521]
```

In [284]:

```
X_full[61521]
```

Out[284]:

```
array([0.31403118, 0.70833333, 0.06185567, 0.        , 1.        ,
       0.        , 0.        , 0.        , 0.        , 1.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 1.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 1.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 1.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 1.        , 0.        , 0.        ,
       0.        , 0.        ])
```

Neural Networks

In [152]:

```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit

params = {
      'hidden_layer_sizes' : [(80,), (100,), (120,), (180,) ], # Some architectures.
      'activation' : ['logistic', 'tanh', 'relu'],            # Activation functions.
      'learning_rate':['adaptative','invscaling','constant'],
    'max_iter':[200,400]
 }

gsearch_nn = GridSearchCV(estimator = MLPRegressor(solver = 'adam', #Model to be explored.
                                                   random_state=1234,
                                                   max_iter= 2000,
                                                   n_iter_no_change=50,
                                                   validation_fraction=0.2),
                          param_grid = params,
                          verbose = 3)

gsearch_nn.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
[CV 1/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=200; tot
al time=   0.1s
[CV 2/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=200; tot
al time=   0.0s
[CV 3/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=200; tot
al time=   0.0s
[CV 4/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=200; tot
al time=   0.0s
[CV 5/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=200; tot
al time=   0.0s
[CV 1/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=400; tot
al time=   0.0s
[CV 2/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=400; tot
al time=   0.0s
[CV 3/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=400; tot
al time=   0.0s
[CV 4/5] END activation=logistic, hidden_layer_sizes=(80,), learning_rate=adaptative, max_iter=400; tot
al time=   0.0s
```

In [153]:

```
y_pred_train_nn = gsearch_nn.predict(X_train)
y_pred_test_nn = gsearch_nn.predict(X_test)
```

In [154]:

```
gsearch_nn.best_params_
```

Out[154]:

```
{'activation': 'logistic',
 'hidden_layer_sizes': (100,),
 'learning_rate': 'invscaling',
 'max_iter': 200}
```

In [155]:

```
print('MAE training',mean_absolute_error(y_train, y_pred_train_nn))
print('MAE testing',mean_absolute_error(y_test, y_pred_test_nn))
```

```
MAE training 8.887686811749388
MAE testing 9.107662935325816
```

In [156]:

```
print('MSE training',mean_squared_error(y_train, y_pred_train_nn))
print('MSE testing',mean_squared_error(y_test, y_pred_test_nn))
```

```
MSE training 113.00915128383913
MSE testing 119.00776924691371
```

In [157]:

```
mean_squared_error(y_train, y_pred_train_nn)

print('RMSE training', np.sqrt(mean_squared_error(y_train, y_pred_train_nn)))

mean_squared_error(y_test, y_pred_test_nn)

print('RMSE testing',np.sqrt(mean_squared_error(y_test, y_pred_test_nn)))
```

```
RMSE training 10.63057624420422
RMSE testing 10.909068211672054
```

In [158]:

```
print('R2 training',r2_score(y_train, y_pred_train_nn))
print('R2 testing', r2_score(y_test, y_pred_test_nn))
```

```
R2 training 0.15970770965387882
R2 testing 0.11230190523552563
```

In [159]:

```
print('SMAPE training',smape(y_pred_train_nn, y_train),
      'SMAPE testing',smape(y_pred_test_nn,y_test))
```

```
SMAPE training 58.7624365258294 SMAPE testing 59.174702577179204
```