Linear Model

Libraries

In [1]:
```
import pandas as pd
import numpy as np
import os
from functools import reduce
from datetime import date,datetime

from scipy.stats import ks_2samp

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

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import roc_auc_score,accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV

import matplotlib.pyplot as plt
import seaborn as sns
import pygal
from scikitplot.metrics import plot_roc_curve

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
pd.set_option('display.max_columns',500)
```

Data Reading

Select specific columns to save memory and processing power

In [2]:
```
csv = '2018_Yellow_Taxi_Trip_Data.csv'

fields = ['VendorID', 'tpep_pickup_datetime',
          'passenger_count', 'trip_distance',
          'PULocationID','payment_type',
          'total_amount']

models = []
```

In [3]:
```
iterator = pd.read_csv(csv, usecols=fields,
                     dtype={'VendorID': 'int8','passenger_count':'int8',
                            'trip_distance':'float32', 'PULocationID':'int16',
                            'payment_type':'int8', 'total_amount':'float32'},
                     parse_dates=["tpep_pickup_datetime"],
                     chunksize=1000000)
```

Functions

Data Filter

In [4]:
```
def data_filter(df):
    ''' data_filter

    Filters data where:
        -Data is not from 2018
        -Total amount is less than zero
        -Passenger count is less than one
        -Trip distance is less or equal to zero

        -Drop passenger_count (we dont use it later in the model)

    Parameter
    ---------
    df: Dataframe chunk
    '''

    #Filters data not from 2018
    start_date = pd.to_datetime('2018-01-01 00:00:00')
    end_date = pd.to_datetime('2019-01-01 00:00:00')
    mask = (df['tpep_pickup_datetime'] >= start_date) & (df['tpep_pickup_datetim
    df = df.loc[mask]

    #Filters data where total amount is less than zero
    df  = df[df["total_amount"] > 0.00]

    #Filters where passenger count is less than one
    df = df[df["passenger_count"] >= 1]

    #Filters where trip distance = 0
    df = df[df["trip_distance"] > 0]

    df.drop(['passenger_count'], axis=1, inplace = True)

    return df
```

In [5]:
```
def dates(df):
    '''dates

    Splits Datetime into two new columns:
        -pickup_date
        -pickup_hour

    Adds column:
        -pickup_hour where: First two digits represent month,
                            Middle two represent day of the week
                            Last two represent the pickup hour

    Drops the original datetime column

    Parameter
    ---------
    df: Dataframe chunk
    '''
    df['pickup_date'] = df['tpep_pickup_datetime'].dt.date

    df['pickup_time'] = df['tpep_pickup_datetime'].dt.time
    df['pickup_time'] = df['pickup_time'].astype(str)
```

```
    ##df['month'] = pd.to_datetime(df['tpep_pickup_datetime'].dt.month)
    df['month'] = df['tpep_pickup_datetime'].dt.strftime('%m')

    #Day of week {01: Monday .... 07:Sunday}
    df['pickup_day_of_week'] = df['tpep_pickup_datetime'].dt.dayofweek +1
    df['pickup_day_of_week'] = '0' + df['pickup_day_of_week'].astype(str)

    df['hour'] = df['tpep_pickup_datetime'].dt.strftime('%H')

    df["pickup_hour"] = df['month'] + df['pickup_day_of_week'] + df['hour']

    df.drop(["tpep_pickup_datetime",
             "month",
             "pickup_day_of_week",
             "hour"], axis=1, inplace=True)

    return df
```

Hours Catalog

In [6]:

```
def hours_catalog(df):
    '''hours_catalog

    Creates id_hour (int16) column based on pickup_hour sorted column

    Parameter
    --------
    df: Chunk of Dataframe that contains pickup_hour values:
        First two values: Month number
        Third and Fourth values: Day of the week {01: Monday .... 07:Sunday}
        Last two values: Hour

    '''
    catfh = df[['pickup_hour']].drop_duplicates().sort_values('pickup_hour',asce
    catfh["id_hour"] = (catfh.index+1)
    catfh["id_hour"] = catfh["id_hour"].astype('int16')

    df = df.merge(catfh, on='pickup_hour', how='inner')
    df.drop('pickup_hour', axis=1, inplace=True)

    return df
```

Data Engineering

In [7]:

```
def trans(df,ancla,k):
    '''trans

    Parameters
    -----------
    df: Chunk of Dataframe
    ancla: current ancla to run
    k: current step
    '''

    aux = df.loc[(df['id_hour']>=(ancla-k+1))&(df['id_hour']<=ancla)].reset_inde
    aux = aux[['id_hour', 'trip_distance', 'total_amount', 'pickup_time','PULoca
    aux['hora'] = aux['pickup_time'].map(lambda x:int(x.split(':')[0])//6).astyp
    aux.drop('pickup_time',axis=1,inplace=True)
    aux['n'] = 1
```

```
    t = aux.copy()

    for v in vard:
        t[v] = 'total_%s'%v
    aux = pd.concat([aux,t],ignore_index=True)

    def piv(aux,v,ancla):
        aux = aux.pivot_table(index='PULocationID',
                        columns=v,
                        values=varc,
                        aggfunc=['min','max','mean','sum','std'])

        aux.columns = ["v_%s_"%v+"_".join(x)+"_%d"%k for x in aux.columns]
        return aux.reset_index().assign(ancla=ancla)

    aux = reduce(lambda x,y:pd.merge(x,y,on=um,how='outer'),map(lambda v:piv(aux
    return aux
```

TAD & Target

In [8]:
```
def TAD(X, y, um):

    tad = X.merge(y,on=um,how='inner')

    return tad
```

In [9]:
```
def target(df, ancla, vdes):
    aux = df.loc[(df['id_hour']>ancla)&(df['id_hour']<=(ancla+vdes))].reset_inde
    aux = aux.groupby(um[0]).sum()
    return aux.assign(ancla=ancla).reset_index()
```

Impute

In [10]:
```
def impute(tad, varc):
    '''impute

    Returns
    X: Based on TAD column with only columns from varc list
    Xi: Imputed df based on X with median strategy

    Parameters
    ..........
    tad: TAD table
    varc: list of continued variables

    '''

    X = tad[varc].copy()
    im = SimpleImputer(strategy='median')
    im.fit(X)
    Xtrans = im.transform(X)

    Xi = pd.DataFrame(Xtrans, columns=varc)

    #Kolmogorov-Smirnov (Two distributions are statistically equal)
    ks = pd.DataFrame(map(lambda v: (v,ks_2samp(tad[v].dropna(),Xi[v]).statistic
    print(ks.loc[ks['ks']>.1])
```

```
    return X, Xi
```

Extreme Values

In [11]:
```
def extremo(df,v,ci,cs):
    aux = df[um+[v]].copy()
    aux['ol_%s'%v] = ((aux[v]<ci)|(aux[v]>cs)).astype(int)
    return aux.drop(v,axis=1)


def extreme(X, um, tad):

    cotas = X.describe(percentiles=[0.01,0.99]).T[['1%', '99%']].reset_index().v
    ext = reduce(lambda x,y:pd.merge(x,y,on=um,how='outer'),map(lambda z:extremo
    varol = [v for v in ext if v[:2]=='ol']
    ext['extremo'] = ext[varol].max(axis=1)
    print(ext['extremo'].describe())
```

Modelling (Find parameters $\vec{\theta}$ of model f for $\vec{y} = f(\mathcal{X}) \rightarrow y = \theta_0 + \vec{\theta} \cdot \vec{x}$

In [12]:
```
def regression(tad, tgt, Xi):
    '''regression

    Creates a linear model for the df passed.
    Appends the model to the models list so we can dump it with pickle

    Parameters
    -----------
    tad: TAD table for the dataframe chunk
    tgt: target column
    Xi: table with imputed values

    '''

    y = tad[tgt].copy()
    Xt,Xv,yt,yv = train_test_split(Xi,y,train_size=0.7)
    modelo = LinearRegression()
    hiperparametros=dict(fit_intercept=[True,False],normalize=[True,False])

    grid = GridSearchCV(param_grid=hiperparametros,
                        estimator=modelo,
                        cv=10,
                        scoring='neg_mean_absolute_error',
                        n_jobs=-1,
                        verbose=True)
    grid.fit(Xt,yt)

    modelo = grid.best_estimator_
    modelo.fit(Xt,yt)
    modelo.intercept_

    print(mean_absolute_error(y_true=yt,y_pred=modelo.predict(Xt)))
    print(mean_absolute_error(y_true=yv,y_pred=modelo.predict(Xv)))

    Xv['y^'] = modelo.predict(Xv)
```

```
    Xv['y'] = yv
    modelo.coef_

    models.append(modelo)
```

First Chunk

Get the first chunk of data to test our functions

In [13]:
```
%%time
df = iterator.get_chunk()
```

```
CPU times: user 1min 18s, sys: 26.2 ms, total: 1min 18s
Wall time: 1min 19s
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
VendorID                0
tpep_pickup_datetime    0
passenger_count         0
trip_distance           0
PULocationID            0
payment_type            0
total_amount            0
dtype: int64
```

No missing data, no need to worry. We next filter the data for 2018.

In [15]:
```
%%time
df = data_filter(df)
df = dates(df)
```

```
CPU times: user 11.6 s, sys: 318 ms, total: 11.9 s
Wall time: 11.9 s
```

Hours Catalog

In [16]:
```
df = hours_catalog(df)
df.head()
```

Out[16]:

	VendorID	trip_distance	PULocationID	payment_type	total_amount	pickup_date	pickup_time	id_I
0	2	8.00	230	2	26.80	2018-09-22	23:46:37	
1	2	1.70	141	1	11.76	2018-09-22	23:00:43	
2	2	3.84	163	2	15.30	2018-09-22	23:10:32	
3	2	1.80	166	2	9.80	2018-09-22	23:27:25	
4	2	2.43	229	1	13.30	2018-09-22	23:50:39	

Data Engineering

In [17]:
```
horai,horaf = df[['id_hour']].describe().T[['min','max']].values[0].tolist()
horai,horaf
```

Out[17]:
```
(1.0, 134.0)
```

In [18]:
```
vobs = 24
vdes = 1
anclai = int(horai)+vobs-1
anclaf = int(horaf)-vdes
anclai,anclaf
```

Out[18]: (24, 133)

In [19]:
```
um = ['PULocationID', 'ancla']
ancla = 24
step = 4
varc = ['trip_distance', 'total_amount', 'n']
vard = ['hora']
```

In [20]:
```
%%time
X_test = pd.concat(map(lambda ancla:reduce(lambda x,y:pd.merge(x,y,on=um,how='ou
       map(lambda k:trans(df,ancla,k),range(step,vobs+step,step))),range(anclai,
```

```
CPU times: user 4min 58s, sys: 2.82 s, total: 5min 1s
Wall time: 5min 1s
```

In [21]:
```
X_test.head()
```

Out[21]:

	PULocationID	v_hora_min_n_3_4	v_hora_min_n_total_hora_4	v_hora_min_total_amount_3_4	v_hora
0	1	1.0	1.0	105.30	
1	4	1.0	1.0	5.80	
2	7	1.0	1.0	4.80	
3	10	1.0	1.0	35.38	
4	12	1.0	1.0	7.55	

In [22]:
```
y_test= pd.concat(map(lambda ancla: target(df,ancla,vdes),range(anclai,anclaf+1)
y_test.head()
```

Out[22]:

	PULocationID	y	ancla
0	4	7	24
1	7	21	24
2	10	6	24
3	12	1	24
4	13	27	24

TAD (Tabla analítica de datos) $\vec{y} = f(\mathcal{X})$

In [23]:
```
tad_test = TAD(X_test,y_test,um)
tad_test.head()
```

Out[23]:

	PULocationID	v_hora_min_n_3_4	v_hora_min_n_total_hora_4	v_hora_min_total_amount_3_4	v_hora
0	4	1.0	1.0	5.80	
1	7	1.0	1.0	4.80	

	PULocationID	v_hora_min_n_3_4	v_hora_min_n_total_hora_4	v_hora_min_total_amount_3_4	v_hora
2	10	1.0	1.0	35.38	
3	12	1.0	1.0	7.55	
4	13	1.0	1.0	0.31	

Exploratory Analysis

Variable Selection

In [24]:
```
varc_test = [v for v in tad_test.columns if v[:2]=="v_"]
tgt_test = 'y'
```

Missing Values Counting

In [46]:
```
miss_test = 1-tad_test[varc_test].describe().T[['count']]/len(tad_test)
miss_test.describe()
```

Out[46]:

	count
count	450.000000
mean	0.293301
std	0.224234
min	0.000000
25%	0.113302
50%	0.231716
75%	0.467293
max	0.736255

Impute

In [26]:
```
X_test, Xi_test = impute(tad_test, varc_test)
```

```
                         variable        ks
2     v_hora_min_total_amount_3_4  0.329459
4    v_hora_min_trip_distance_3_4  0.345802
8     v_hora_max_total_amount_3_4  0.345092
10   v_hora_max_trip_distance_3_4  0.346276
14   v_hora_mean_total_amount_3_4  0.346986
..                            ...       ...
444              v_hora_sum_n_2_4  0.328851
445   v_hora_sum_total_amount_2_4  0.328851
446  v_hora_sum_trip_distance_2_4  0.328851
448   v_hora_std_total_amount_2_4  0.344966
449  v_hora_std_trip_distance_2_4  0.344966

[181 rows x 2 columns]
```

Valores Extremos

In [27]:
```
extreme(X_test, um, tad_test)
```

```
count    9585.000000
mean        0.509442
std         0.499937
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: extremo, dtype: float64
```

Modelación (encontrar los parámetros $\vec{\theta}$ del modelo f para $\vec{y} = f(\mathcal{X}) \rightarrow y = \theta_0 + \vec{\theta} \cdot \vec{x}$

In [28]:
```
tad_test.head()
```

Out[28]:

	PULocationID	v_hora_min_n_3_4	v_hora_min_n_total_hora_4	v_hora_min_total_amount_3_4	v_hora
0	4	1.0	1.0	5.80	
1	7	1.0	1.0	4.80	
2	10	1.0	1.0	35.38	
3	12	1.0	1.0	7.55	
4	13	1.0	1.0	0.31	

In [29]:
```
y_test = tad_test['y'].copy()
Xt_test,Xv_test,yt_test,yv_test = train_test_split(Xi_test,y_test,train_size=0.7
modelo_test = LinearRegression()
hiperparametros=dict(fit_intercept=[True,False],normalize=[True,False])

grid_test = GridSearchCV(param_grid=hiperparametros,
                estimator=modelo_test,
                cv=10,
                scoring='neg_mean_absolute_error',
                n_jobs=-1,
                verbose=True)
```

In [30]:
```
grid_test.fit(Xt_test,yt_test)
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    7.7s finished
```

Out[30]:
```
GridSearchCV(cv=10, estimator=LinearRegression(), n_jobs=-1,
             param_grid={'fit_intercept': [True, False],
                         'normalize': [True, False]},
             scoring='neg_mean_absolute_error', verbose=True)
```

In [31]:
```
modelo_test = grid_test.best_estimator_
```

In [32]:
```
modelo_test.fit(Xt_test,yt_test)
```

Out[32]:
```
LinearRegression(fit_intercept=False, normalize=True)
```

In [33]:
```
modelo_test.intercept_
```

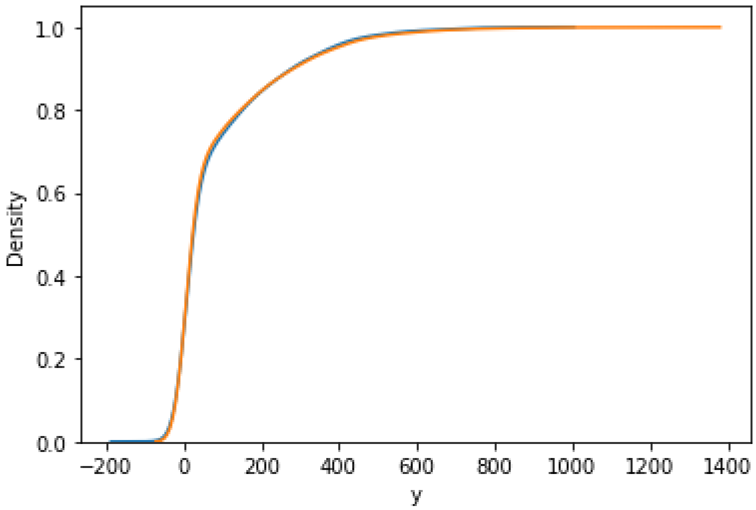
Out[33]:
```
0.0
```

In [34]:
```
print(mean_absolute_error(y_true=yt_test,y_pred=modelo_test.predict(Xt_test)))
print(mean_absolute_error(y_true=yv_test,y_pred=modelo_test.predict(Xv_test)))
```

```
22.515774748259656
22.701732484535974
```

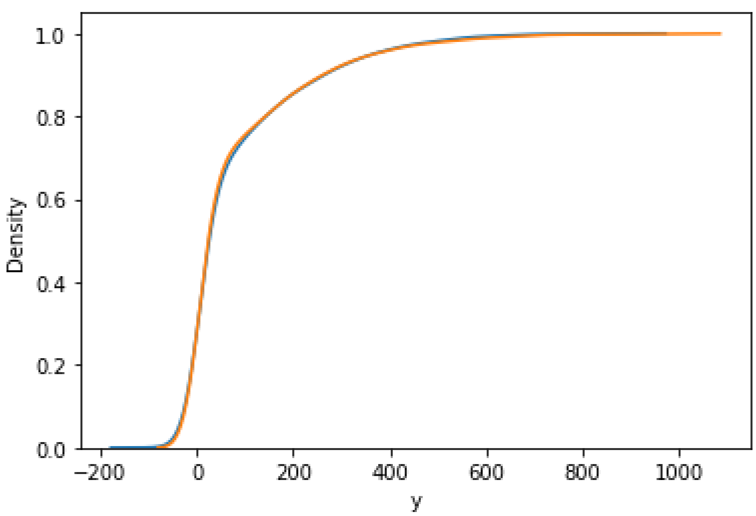
In [35]:
```
sns.distplot(modelo_test.predict(Xt_test),hist=False,kde_kws={'cumulative':True}
sns.distplot(yt_test,hist=False,kde_kws={'cumulative':True})
```

Out[35]:
```
<AxesSubplot:xlabel='y', ylabel='Density'>
```



In [36]:
```
sns.distplot(modelo_test.predict(Xv_test),hist=False,kde_kws={'cumulative':True}
sns.distplot(yv_test,hist=False,kde_kws={'cumulative':True})
```

Out[36]:
```
<AxesSubplot:xlabel='y', ylabel='Density'>
```



In [37]:
```
Xv_test['y^'] = modelo_test.predict(Xv_test)
Xv_test['y'] = yv_test
```

In [38]:
```
modelo_test.coef_
```

Out[38]:
```
array([-1.48760393e-01, -1.48760393e-01,  1.33688497e-01, -8.21227846e-01,
       -1.80719876e-01,  1.03225620e+00, -1.48760393e-01, -1.48760393e-01,
```

```
 3.47944646e-01, -1.04221773e-03, -1.30113183e+00, -4.62142203e-03,
-1.48760393e-01, -1.48760393e-01,  2.49133424e-02,  7.01695131e-01,
-7.00091044e-01, -8.87602136e-02, -1.24168639e+00,  1.17520706e+00,
 1.25832842e-02,  2.84669706e-02,  3.69635076e-01, -5.15159571e-01,
 1.14446785e-11, -1.80533366e-12, -7.77923529e-01, -7.69044832e-01,
 2.89467352e+00,  1.30259978e-01, -1.48760393e-01, -1.48760393e-01,
-1.48760393e-01,  1.07424143e+00,  5.39737947e-01, -3.65926858e-01,
-5.71023681e-01, -2.96726946e+00,  1.29557904e+00, -1.48760393e-01,
-1.48760393e-01, -1.48760393e-01, -4.76540925e-01, -1.75090748e-01,
 2.93042754e-01,  1.09945962e+00,  1.09507451e+00, -9.74702102e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -7.57862669e-01,
-7.88799788e-01, -1.52582439e-02, -1.28742316e+00,  2.97879228e+00,
 2.75856969e-01, -1.70366949e+00, -1.33331046e+00,  1.45210675e+00,
 7.35876946e-02,  3.09158991e-02, -5.04361851e-02,  1.00563833e-01,
 2.19445916e-01, -1.41264099e-01, -1.61537450e-14, -1.33920652e-13,
 9.45493683e-14,  1.89428444e+00,  8.11460776e-01, -5.57794434e-01,
-3.36485940e+00, -3.24264437e+00,  1.94858723e+00, -1.48760393e-01,
-1.48760393e-01, -1.48760393e-01, -1.22439797e-01, -9.20638780e-01,
-4.29445645e-01, -1.84611341e+00,  4.41350368e+00,  1.44743570e+00,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01,  3.17672330e-02,
 1.37878153e-01, -1.32618603e-01,  2.20882601e-01, -6.91002117e-01,
-2.05031525e-02, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
 1.13389600e-01,  8.67916863e-01,  5.64349997e-01,  2.75574543e+00,
-3.32265811e+00, -1.58444104e+00,  2.08273892e+00,  2.18622092e+00,
-2.02852314e+00,  1.10005306e-01,  1.05868640e-01, -1.22938339e-01,
-1.24074186e+00, -1.23515380e+00,  1.28288192e+00, -8.54871729e-14,
-1.13242749e-14, -4.32986980e-14, -7.15690269e-01, -7.86635339e-01,
 3.68908113e-02,  2.34001202e+00,  3.99790013e+00, -8.48601770e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
-1.17033693e-01, -3.50870627e-01,  1.07519623e-01,  5.33245394e-01,
-6.34462692e-01,  1.06644777e+00,  6.06055298e-01, -1.69160828e+00,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
-2.33149899e-02,  2.14080413e-01, -4.45045130e-02, -2.67635320e-02,
 4.76397650e-01,  6.29695696e-01,  5.40700948e-01, -9.24102899e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
 3.16897603e-01, -1.41349375e-02, -1.90821221e-01, -2.53143358e-01,
-1.99790477e+00, -1.08691507e+00, -1.57045496e+00,  2.78054243e+00,
 2.66365062e+00,  2.65850481e+00,  2.66739894e+00, -2.65055484e+00,
-8.80118936e-02, -8.24347544e-02, -1.10679953e-01,  8.67703212e-02,
-4.03816320e-01, -4.31824601e-01, -3.14663990e-01,  3.96437487e-01,
-3.17523785e-14,  1.62092562e-14, -2.98649994e-14,  2.17603713e-14,
-3.79441557e-01, -7.12657867e-01,  1.79564622e-01,  7.61302476e-01,
 2.62448112e-01, -8.98544562e-01,  9.45667919e-02, -4.62498233e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
-1.48760393e-01, -3.69573799e-01, -4.13698407e-01,  1.13239005e+00,
-2.45156342e-01, -5.44315796e-01,  9.49376652e-01,  1.69307493e+00,
-4.70068902e+00, -9.38987518e-01,  2.91975550e+00, -1.48760393e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
-1.10542426e-01,  2.11426621e-01, -1.84120913e-01, -1.62264554e-02,
 8.07796667e-02,  7.63924785e-01, -1.33127957e+00, -1.41430534e+00,
-1.08195940e+00,  1.43765357e+00, -1.48760393e-01, -1.48760393e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01,  2.53777735e-01,
 6.11569094e-02, -9.08936296e-01,  1.15865762e-01,  4.37881421e-01,
-4.85943772e-01,  6.73858664e-01,  7.44649012e+00,  2.98742216e+00,
-6.81289537e+00, -1.37684477e+00, -1.64640277e+00, -1.56546075e+00,
-1.81046231e+00,  1.44055825e+00,  1.13495202e-01,  1.21187129e-01,
 1.35991214e-01,  1.77801935e-01, -1.17284849e-01, -9.53549904e-02,
-1.19104619e-01, -1.64111090e-01, -3.02896808e-01,  1.11545796e-01,
 5.80091530e-15, -1.33226763e-15,  1.26565425e-14, -5.32907052e-15,
 6.69603262e-15, -1.71111043e-02, -4.31129795e-01,  8.54122193e-01,
-2.04420416e-01, -5.79524192e-01, -6.89687791e-01,  2.52517627e+00,
-1.33413629e+00, -9.11990196e-02,  8.68699261e-01, -1.48760393e-01,
-1.48760393e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
-2.68845444e-01, -4.30198881e-01, -8.09855452e-01,  5.15595154e-01,
 3.71544952e-01,  2.29458350e+00,  2.46001943e+00,  4.05058079e+00,
```

```
       -1.50003270e+00, -2.84399717e+00, -1.48760393e-01, -1.48760393e-01,
       -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,  7.03519969e-03,
       -7.25624041e-02,  1.98796634e-01, -3.07641675e-02, -1.08718149e-01,
       -3.99784656e-01, -1.23347457e-01,  2.61781404e-01,  6.70140867e-01,
       -1.70499938e-01, -1.48760393e-01, -1.48760393e-01, -1.48760393e-01,
       -1.48760393e-01, -1.48760393e-01,  5.86854001e-01,  4.22347144e-01,
        6.62088394e-01, -1.86258510e-01, -4.58021734e-01, -3.24426286e+00,
       -2.56458123e+00, -5.96989635e+00, -2.53422546e-01,  5.30742409e+00,
        5.05152213e-01,  5.78096564e-01,  3.89240524e-01,  6.26597780e-01,
       -3.50714541e-01, -9.69596369e-02, -9.48210783e-02, -8.78706361e-02,
       -1.15754293e-01,  8.14079268e-02,  3.33510099e-01,  3.70735829e-01,
        3.19555166e-01,  4.19383341e-01, -3.08197800e-01, -8.88178420e-16,
        0.00000000e+00,  3.10862447e-15, -3.77475828e-15, -1.33226763e-15,
       -1.99070463e-01,  1.25050175e-01, -5.04444050e-01,  5.06474401e-01,
        3.82348449e-02,  2.19918105e+00,  1.19021973e+00,  1.61582891e+00,
       -2.16726845e+00, -8.94120871e-01, -1.48760393e-01,  9.33433484e-01,
       -2.46596863e+00, -1.48760393e-01, -4.43811363e-01,  5.91878118e-01,
       -1.48760393e-01, -1.14077106e+00,  2.18176411e+00, -2.34762466e+00,
        1.65298362e-01, -8.15424772e-02,  0.00000000e+00,  1.16371907e+00,
        1.35341950e-01, -1.48760393e-01, -1.94399739e-02,  2.39253097e+00,
       -1.48760393e-01,  8.96549189e-02,  8.85652794e-01, -1.48760393e-01,
        1.38956986e-01, -3.21611261e+00, -1.15765211e+00,  1.69119867e-03,
        3.05248060e-01,  0.00000000e+00, -2.71598848e-01,  1.88633632e+00,
       -1.48760393e-01, -1.36075873e-01, -3.31376252e+00, -1.48760393e-01,
       -6.35789860e-02,  1.34656102e+00, -1.48760393e-01,  2.50974719e-01,
        2.69207013e+00,  2.31366120e+00,  1.00786138e-01, -1.25805647e+00,
        0.00000000e+00,  1.72419164e-01, -5.60703566e+00, -1.48760393e-01,
        7.79090694e-01, -1.00211327e+00, -1.48760393e-01,  1.73270799e-01,
       -1.38140334e+00, -1.48760393e-01, -1.15770623e+00,  2.04440461e+00,
        2.16136467e+00, -2.87131010e-02, -5.52172746e-01,  0.00000000e+00,
        7.53421935e-02,  2.47762726e+00, -1.48760393e-01,  8.44966709e-01,
        7.30641236e-01, -1.48760393e-01,  4.05826464e-03, -3.63405248e-02,
       -1.48760393e-01, -7.38617825e-01, -1.53793510e+00, -8.27014282e-01,
       -3.97980927e-02,  5.59548517e-01,  0.00000000e+00,  7.26728373e-01,
        1.23617094e+00, -1.48760393e-01,  8.36611784e-01, -4.70918669e+00,
       -1.48760393e-01, -2.49362165e-01,  9.43478171e-01, -1.48760393e-01,
       -7.02875995e-01,  5.00561668e+00, -1.57273582e+00,  4.92869140e-02,
        1.77723583e-01,  0.00000000e+00,  8.96051179e-01, -5.01175561e+00,
       -1.48760393e-01,  2.65063958e-01, -3.83080933e-01, -1.48760393e-01,
        5.09553565e-02,  8.35719221e-01, -1.48760393e-01, -2.39717421e-01,
       -4.79719337e-01,  2.12053206e+00,  1.27340988e-01, -1.32746825e+00,
        0.00000000e+00, -8.11836185e-02, -5.24474048e-02, -1.48760393e-01,
        3.24389840e-01,  7.88814318e-02, -1.48760393e-01,  1.65148631e-01,
       -4.13702407e-01, -1.48760393e-01, -1.30164145e-01, -1.14939492e+00,
       -6.19870349e-01, -6.10749375e-02,  6.09387869e-01,  0.00000000e+00,
        2.40205003e-01,  4.95726094e-01])
```

Juntamos todas las funciones para predecir con todos los chunks

In [39]:
```
%%time
number = 1

for chunk in iterator:

    print(f'Model number {number}')

    chunk = data_filter(chunk)
    chunk = dates(chunk)
    chunk = hours_catalog(chunk)
```

```
    hri,hrf = chunk[['id_hour']].describe().T[['min','max']].values[0].tolist()
    vobs = 24
    vdes = 1
    anclai = int(hri)+vobs-1
    anclaf = int(hrf)-vdes
    um = ['PULocationID', 'ancla']
    ancla = 24
    step = 4
    varc = ['trip_distance', 'total_amount', 'n']
    vard = ['hora']

    print(f'Hra inicial: {hri}, Modelo: {number}')
    print(f'Hra final: {hrf}, Modelo: {number}')

    X = pd.concat(map(lambda ancla:reduce(lambda x,y:pd.merge(x,y,on=um,how='out
           map(lambda k:trans(chunk,ancla,k),range(step,vobs+step,step))),range(
    y= pd.concat(map(lambda ancla: target(chunk,ancla,vdes),range(anclai,anclaf+

    tad = TAD(X, y, um)
    varc = [v for v in tad.columns if v[:2]=="v_"]
    tgt = 'y'

    miss = 1-tad[varc].describe().T[['count']]/len(tad)

    X, Xi = impute(tad, varc)

    extreme(X, um, tad)


    regression(tad, tgt, Xi)

    number += 1
```

```
Model number 1
Hra inicial: 1.0, Modelo: 1
Hra final: 112.0, Modelo: 1
                         variable        ks
3     v_hora_min_total_amount_1_4  0.294406
4     v_hora_min_total_amount_2_4  0.316723
6    v_hora_min_trip_distance_1_4  0.312624
7    v_hora_min_trip_distance_2_4  0.323951
12    v_hora_max_total_amount_1_4  0.312452
..                            ...       ...
444             v_hora_sum_n_0_8  0.266582
445   v_hora_sum_total_amount_0_8  0.266818
446  v_hora_sum_trip_distance_0_8  0.266700
448   v_hora_std_total_amount_0_8  0.288464
449  v_hora_std_trip_distance_0_8  0.288464

[156 rows x 2 columns]
count    9722.000000
mean        0.358465
std         0.479574
min         0.000000
25%         0.000000
50%         0.000000
75%         1.000000
max         1.000000
Name: extremo, dtype: float64
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    3.9s finished
```

```
14   v_hora_mean_total_amount_0_4  0.345952
..                          ...       ...
444              v_hora_sum_n_3_4  0.341100
445   v_hora_sum_total_amount_3_4  0.341568
446  v_hora_sum_trip_distance_3_4  0.341568
448   v_hora_std_total_amount_3_4  0.351450
449  v_hora_std_trip_distance_3_4  0.351450

[164 rows x 2 columns]
count    9209.000000
mean        0.404061
std         0.490736
min         0.000000
25%         0.000000
50%         0.000000
75%         1.000000
max         1.000000
Name: extremo, dtype: float64
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    2.9s finished
18.75571170775452
20.830105257517218
Model number 112
Hra inicial: 1.0, Modelo: 112
Hra final: 30.0, Modelo: 112
                           variable        ks
3       v_hora_min_total_amount_2_4  0.195548
6      v_hora_min_trip_distance_2_4  0.225138
12      v_hora_max_total_amount_2_4  0.225138
15     v_hora_max_trip_distance_2_4  0.225138
21     v_hora_mean_total_amount_2_4  0.225138
24    v_hora_mean_trip_distance_2_4  0.225138
27                 v_hora_sum_n_2_4  0.225138
30      v_hora_sum_total_amount_2_4  0.225138
33     v_hora_sum_trip_distance_2_4  0.225138
39      v_hora_std_total_amount_2_4  0.249995
42     v_hora_std_trip_distance_2_4  0.249995
49      v_hora_min_total_amount_1_8  0.385702
53     v_hora_min_trip_distance_1_8  0.409316
61      v_hora_max_total_amount_1_8  0.417187
65     v_hora_max_trip_distance_1_8  0.417187
73     v_hora_mean_total_amount_1_8  0.417187
77    v_hora_mean_trip_distance_1_8  0.417187
81                 v_hora_sum_n_1_8  0.417187
85      v_hora_sum_total_amount_1_8  0.417187
89     v_hora_sum_trip_distance_1_8  0.417187
97      v_hora_std_total_amount_1_8  0.422749
101    v_hora_std_trip_distance_1_8  0.422749
170    v_hora_min_total_amount_0_16  0.231921
175   v_hora_min_trip_distance_0_16  0.237683
185    v_hora_max_total_amount_0_16  0.237683
190   v_hora_max_trip_distance_0_16  0.239123
200   v_hora_mean_total_amount_0_16  0.239123
205  v_hora_mean_trip_distance_0_16  0.239123
210               v_hora_sum_n_0_16  0.239123
215    v_hora_sum_total_amount_0_16  0.239123
220   v_hora_sum_trip_distance_0_16  0.239123
230    v_hora_std_total_amount_0_16  0.275000
235   v_hora_std_trip_distance_0_16  0.275000
count    640.000000
mean       0.325000
std        0.468741
min        0.000000
25%        0.000000
```

```
50%        0.000000
75%        1.000000
max        1.000000
Name: extremo, dtype: float64
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    0.2s finished
15.399852347320953
38.200627576156414
CPU times: user 7h 51min 52s, sys: 5min 39s, total: 7h 57min 32s
Wall time: 7h 52min 21s
```

Persistencia de los Modelos

In [40]:
```
import pickle
```

In [45]:
```
with open("models_other.pkl", "wb") as f:
    for model in models:
         pickle.dump(model, f)
```

Si queremos abrir los modelos de nuevo para evitar entrenamiento

In [42]:
```
new_models = []
with open("models.pckl", "rb") as f:
    while True:
        try:
            new_models.append(pickle.load(f))
        except EOFError:
            break
```

In [44]:
```
len(new_models)
```

Out[44]:
```
112
```

In [ ]:
```

```