```
# This is formatted as code
```

Karolina Kotlowska

4. Laboratorium Eksploracji Danych:

Bike Sharing Dataset

Data:

- Uzupełnij dane
- Uzupełnij kod i wygeneruj rezultaty zgodnie z opisem
- Odpowiedz na pytania zaznaczone w tekście
- Wydrukuj jako PDF
- Wyślij jako sprawozdanie

4.1.Upgrade scikit-learn

Chcemy obliczać MAPE, odpowiednia funkcja pojawiła się w nowszej wersji

In [1]:
```
# Być może niepotrzebne
!pip install scikit-learn --upgrade
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/c
olab-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.9/
dist-packages (1.2.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/py
thon3.9/dist-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/
dist-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.
9/dist-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.
9/dist-packages (from scikit-learn) (1.1.1)
```

4.2 Załaduj zbiór danych

Zbiór jest opublikowany w repozytorum UCI jako Bike Sharing Dataset

In [2]:
```
!wget https://dysk.agh.edu.pl/s/G6ZNziBRbEEcMeN/download -O Bike-Sharing-
!unzip Bike-Sharing-Dataset.zip
!cat Readme.txt
```

```
--2023-03-25 08:24:05--  https://dysk.agh.edu.pl/s/G6ZNziBRbEEcMeN/downl
oad
Resolving dysk.agh.edu.pl (dysk.agh.edu.pl)... 149.156.96.4, 2001:6d8:1
0:1060::6004
Connecting to dysk.agh.edu.pl (dysk.agh.edu.pl)|149.156.96.4|:443... con
nected.
HTTP request sent, awaiting response... 200 OK
Length: 279992 (273K) [application/zip]
Saving to: ‘Bike-Sharing-Dataset.zip’

Bike-Sharing-Datase 100%[===================>] 273.43K   440KB/s    in
0.6s

2023-03-25 08:24:07 (440 KB/s) - ‘Bike-Sharing-Dataset.zip’ saved [27999
2/279992]

Archive:  Bike-Sharing-Dataset.zip
  inflating: Readme.txt
  inflating: day.csv
  inflating: hour.csv
==========================================
Bike Sharing Dataset
==========================================

Hadi Fanaee-T

Laboratory of Artificial Intelligence and Decision Support (LIAAD), Univ
ersity of Porto
INESC Porto, Campus da FEUP
Rua Dr. Roberto Frias, 378
4200 - 465 Porto, Portugal


=========================================
Background
=========================================

Bike sharing systems are new generation of traditional bike rentals wher
e whole process from membership, rental and return
back has become automatic. Through these systems, user is able to easily
rent a bike from a particular position and return
back at another position. Currently, there are about over 500 bike-shari
ng programs around the world which is composed of
over 500 thousands bicycles. Today, there exists great interest in these
systems due to their important role in traffic,
environmental and health issues.

Apart from interesting real world applications of bike sharing systems,
the characteristics of data being generated by
these systems make them attractive for the research. Opposed to other tr
ansport services such as bus or subway, the duration
of travel, departure and arrival position is explicitly recorded in thes
e systems. This feature turns bike sharing system into
a virtual sensor network that can be used for sensing mobility in the ci
ty. Hence, it is expected that most of important
events in the city could be detected via monitoring these data.

=========================================
Data Set
=========================================
```

```
Bike-sharing rental process is highly correlated to the environmental an
d seasonal settings. For instance, weather conditions,
precipitation, day of week, season, hour of the day, etc. can affect the
rental behaviors. The core data set is related to
the two-year historical log corresponding to years 2011 and 2012 from Ca
pital Bikeshare system, Washington D.C., USA which is
publicly available in http://capitalbikeshare.com/system-data. We aggreg
ated the data on two hourly and daily basis and then
extracted and added the corresponding weather and seasonal information.
Weather information are extracted from http://www.freemeteo.com.

=========================================
Associated tasks
=========================================

        - Regression:
                Predication of bike rental count hourly or daily based o
n the environmental and seasonal settings.

        - Event and Anomaly Detection:
                Count of rented bikes are also correlated to some events
in the town which easily are traceable via search engines.
                For instance, query like "2012-10-30 washington d.c." in
Google returns related results to Hurricane Sandy. Some of the important
events are
                identified in [1]. Therefore the data can be used for va
lidation of anomaly or event detection algorithms as well.


=========================================
Files
=========================================

        - Readme.txt
        - hour.csv : bike sharing counts aggregated on hourly basis. Rec
ords: 17379 hours
        - day.csv - bike sharing counts aggregated on daily basis. Recor
ds: 731 days


=========================================
Dataset characteristics
=========================================
Both hour.csv and day.csv have the following fields, except hr which is
not available in day.csv

        - instant: record index
        - dteday : date
        - season : season (1:springer, 2:summer, 3:fall, 4:winter)
        - yr : year (0: 2011, 1:2012)
        - mnth : month ( 1 to 12)
        - hr : hour (0 to 23)
        - holiday : weather day is holiday or not (extracted from htt
p://dchr.dc.gov/page/holiday-schedule)
        - weekday : day of the week
        - workingday : if day is neither weekend nor holiday is 1, other
wise is 0.
        + weathersit :
                - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
                - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clo
```

```
uds, Mist
                - 3: Light Snow, Light Rain + Thunderstorm + Scattered c
louds, Light Rain + Scattered clouds
                - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Sno
w + Fog
        - temp : Normalized temperature in Celsius. The values are divid
ed to 41 (max)
        - atemp: Normalized feeling temperature in Celsius. The values a
re divided to 50 (max)
        - hum: Normalized humidity. The values are divided to 100 (max)
        - windspeed: Normalized wind speed. The values are divided to 67
(max)
        - casual: count of casual users
        - registered: count of registered users
        - cnt: count of total rental bikes including both casual and reg
istered

=========================================
License
=========================================
Use of this dataset in publications must be cited to the following publi
cation:

[1] Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble d
etectors and background knowledge", Progress in Artificial Intelligence
(2013): pp. 1-15, Springer Berlin Heidelberg, doi:10.1007/s13748-013-004
0-3.

@article{
        year={2013},
        issn={2192-6352},
        journal={Progress in Artificial Intelligence},
        doi={10.1007/s13748-013-0040-3},
        title={Event labeling combining ensemble detectors and backgroun
d knowledge},
        url={http://dx.doi.org/10.1007/s13748-013-0040-3},
        publisher={Springer Berlin Heidelberg},
        keywords={Event labeling; Event detection; Ensemble learning; Ba
ckground knowledge},
        author={Fanaee-T, Hadi and Gama, Joao},
        pages={1-15}
}

=========================================
Contact
=========================================

For further information about this dataset please contact Hadi Fanaee-T
(hadi.fanaee@fe.up.pt)
```

Załaduj do Pandas DataFrame

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('day.csv',parse_dates=['dteday'])
df.head()
```

Out[3]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	te
0	1	2011-01-01	1	0	1	0	6	0	2	0.3441
1	2	2011-01-02	1	0	1	0	0	0	2	0.3634
2	3	2011-01-03	1	0	1	0	1	1	1	0.1963
3	4	2011-01-04	1	0	1	0	2	1	1	0.2000
4	5	2011-01-05	1	0	1	0	3	1	1	0.2269

4.2.1 Narysuj histogramy

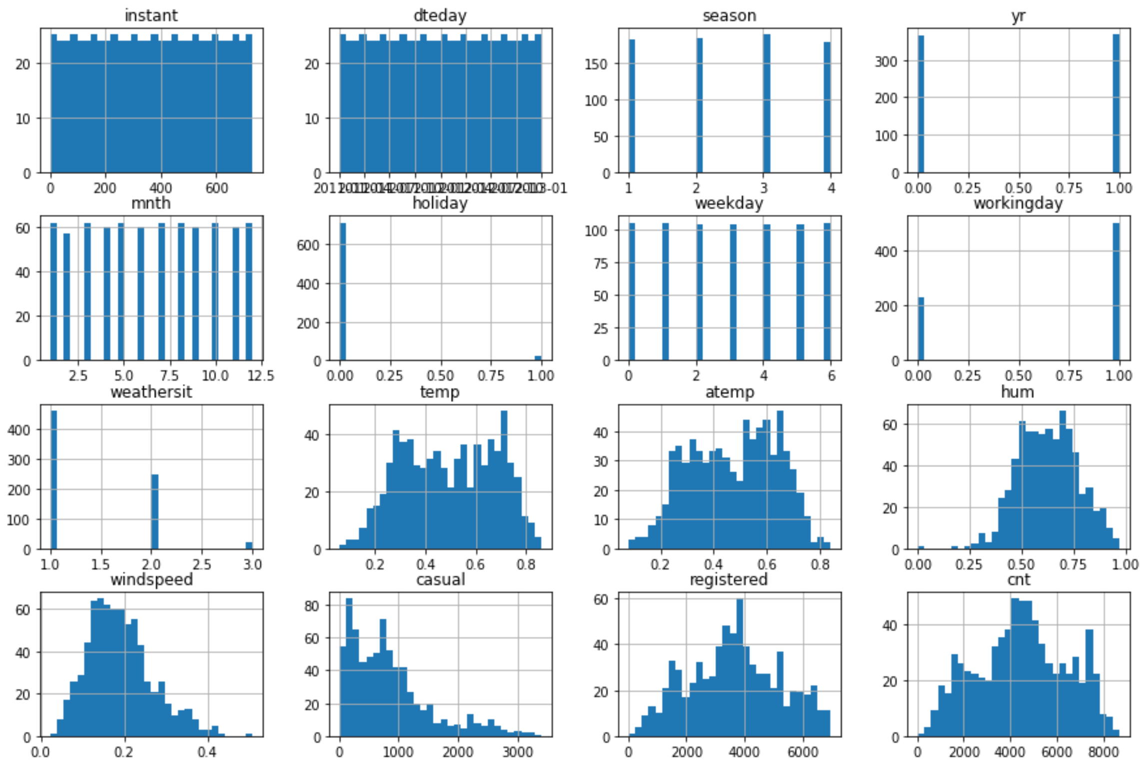
TODO 4.2.1

In [5]:

```
plt.rcParams["figure.figsize"] = (15,10)
df.hist(bins=30)
```

Out[5]:

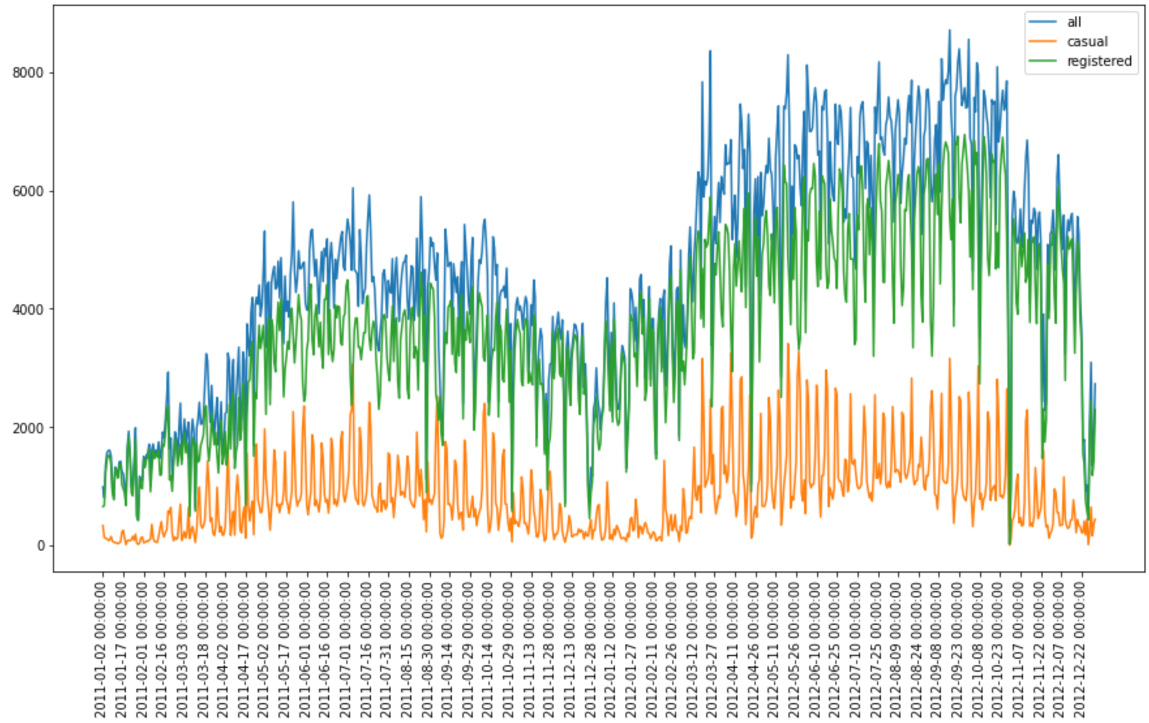
```
array([[<Axes: title={'center': 'instant'}>,
        <Axes: title={'center': 'dteday'}>,
        <Axes: title={'center': 'season'}>,
        <Axes: title={'center': 'yr'}>],
       [<Axes: title={'center': 'mnth'}>,
        <Axes: title={'center': 'holiday'}>,
        <Axes: title={'center': 'weekday'}>,
        <Axes: title={'center': 'workingday'}>],
       [<Axes: title={'center': 'weathersit'}>,
        <Axes: title={'center': 'temp'}>,
        <Axes: title={'center': 'atemp'}>,
        <Axes: title={'center': 'hum'}>],
       [<Axes: title={'center': 'windspeed'}>,
        <Axes: title={'center': 'casual'}>,
        <Axes: title={'center': 'registered'}>,
        <Axes: title={'center': 'cnt'}>]], dtype=object)
```



2.2 Narysuj wykresy dzienne dla wypozyczeń (registered,casual, wszystkich)

TODO 4.2.2

In [6]:
```
plt.rcParams["figure.figsize"] = (15,8)
fig = plt.figure()
plt.plot(df['instant'], df['cnt'],label='all')
plt.plot(df.instant, df['casual'],label='casual')
plt.plot(df.instant, df['registered'],label='registered')
tick_marks = np.arange(1,df['instant'].max(),15)
labels = df['dteday'].iloc[tick_marks]
plt.xticks(tick_marks, labels,rotation=90)
plt.legend()
plt.show()
```



In [7]:
```
df2 = df[df.cnt<30]
df2.head()
# len(df)
df.cnt.quantile([0.002,0.1,0.25,0.5,0.75, 0.9,0.99])
```

Out[7]:
```
0.002     435.6
0.100    1746.0
0.250    3152.0
0.500    4548.0
0.750    5956.0
0.900    7290.0
0.990    8163.7
Name: cnt, dtype: float64
```

4.3. Dane BASIC - regresja (bez przetwarzania wstępnego)

Będziemy starali się wyznaczyć wartość cnt (całkowitej liczby wypozyczeń).

Porównamy wyniki dla:

1. LinearRegression
2. Ridge
3. Lasso - ma zdolnosc usuwania atrybutow
4. i obiecującego algorytmu XGBRegressor

TODO 4.3.1

Jaka zależność zachodzi pomiędzy cnt, casual i regestered?

Czy są potrzebne?

Dopsasuj modele, narysuj rysunki, umieść w sprawozdaniu metryki

In [8]:
```
# Użyteczne funkcje

import sklearn.metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from xgboost import XGBRegressor

scores={'r2':sklearn.metrics.r2_score,
        'mse':sklearn.metrics.mean_squared_error,
        'rmse':lambda y_true,y_pred : np.sqrt(sklearn.metrics.mean_square
        'maxe':sklearn.metrics.max_error,
        'med':sklearn.metrics.median_absolute_error,
        'mae':sklearn.metrics.mean_absolute_error,
        'mape':sklearn.metrics.mean_absolute_percentage_error,
        }

def train_and_test(X,y,regr=sklearn.linear_model.LinearRegression()):
  # print(regr)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0

  regr.fit(X_train, y_train)
  y_pred = regr.predict(X_test)

  for k in scores:
    r = scores[k](y_test,y_pred)
    print(f'{k}:{r}')
  return scores['r2'](y_test,y_pred)
```

Przygotuj dane

TODO 4.3.2

1. Usuń z DataFrame:

- Wszelkie klucze lub identyfikatory
- Zmienną wyjściową (objaśnianą) i zmienne z nią w oczywisty sposób powiązane

2. Zamień datę na postać numeryczną
3. Przekonwertuj do tablicy `numpy`

In [9]:
```
df2 = df.drop(columns=['cnt', 'casual', 'registered', 'instant'])
df2.dteday = pd.to_numeric(df2.dteday)
X=df2.to_numpy()
y=df.cnt.to_numpy()
```

TODO 4.3.3

Wyznacz **najlepszy algorytm na podstawie miary r2**

Narysuj **wszystkie przebiegi** pokazujące prawdziwą i przewidywaną wartość cnt

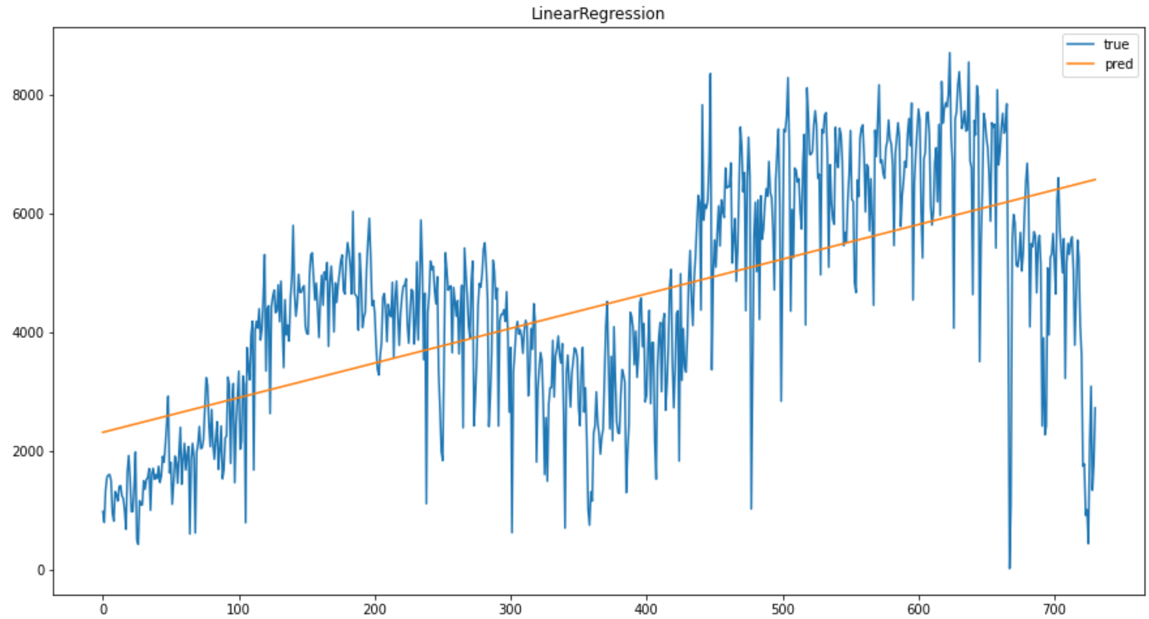
In [10]:
```
# print(f'cnt.min={y.min()} cnt.max={y.max()}')

predictors =[LinearRegression(),
             Ridge(solver='svd'),
```

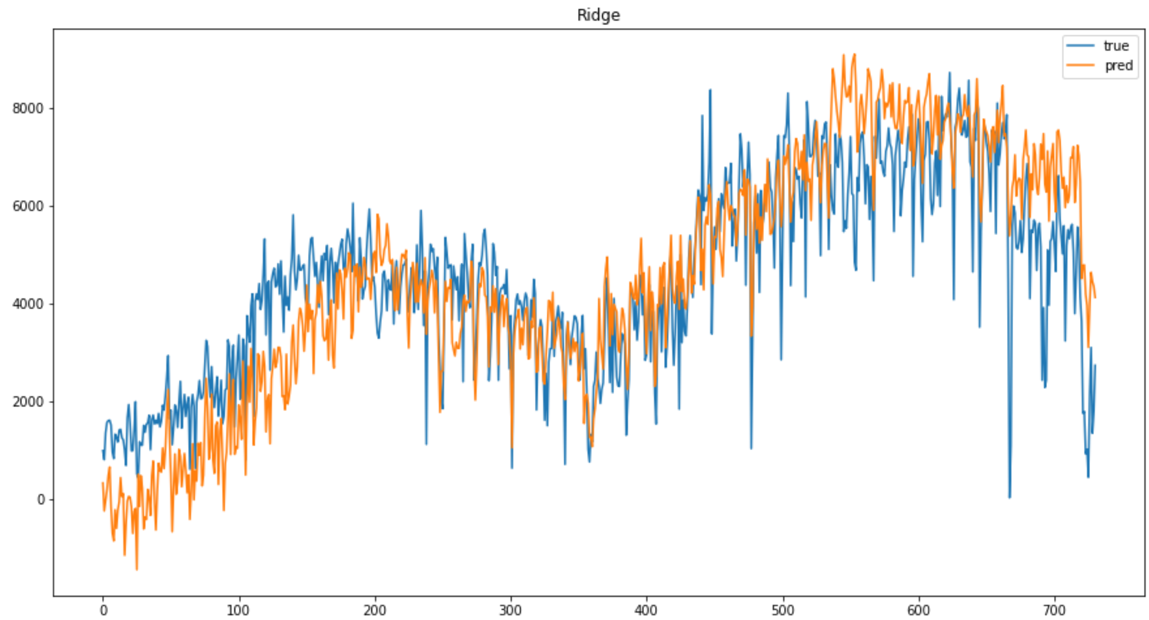
```
            Lasso(max_iter=10000),
            XGBRegressor()
            ]
max_r2= 0
best_reg = 0

for reg in predictors:
  print(f'------ {reg.__class__.__name__} -----')
  r2 = train_and_test(X,y,reg)
  if r2>max_r2:
    best_reg = r2

print(f'Best: {best_reg.__class__.__name__} r2={max_r2}')

def plot(X,y,reg,start=0,end=-1):
  y_pred=reg.predict(X)
  if end==-1:
    end=X.shape[0]
  x = np.arange(start,end)
  plt.plot(x,y[start:end],label='true')
  plt.plot(x,y_pred[start:end],label='pred')
  plt.legend()
  plt.title(reg.__class__.__name__)
  plt.show()

for i in range(0, 3):
  plot(X,y,predictors[i])
```

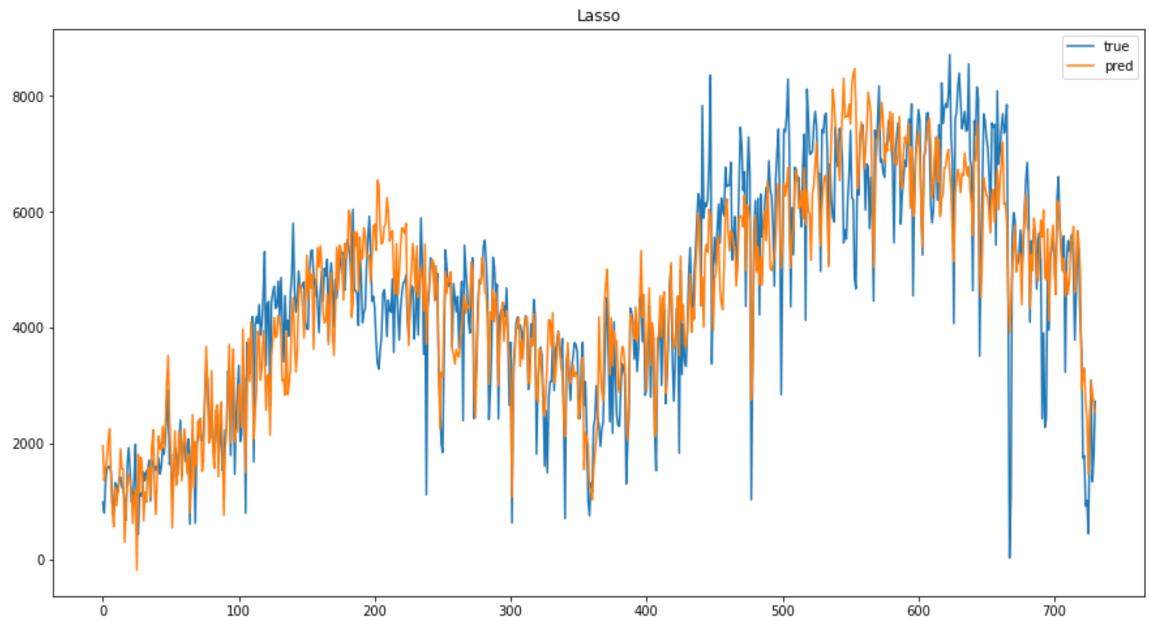
```
------ LinearRegression -----
r2:0.42865980977823803
mse:1787556.037548185
rmse:1336.9951524026499
maxe:5529.289984757212
med:1127.6418553782642
mae:1132.4053138852832
mape:0.3297016233664532
------ Ridge -----
r2:0.5442683688786971
mse:1425850.7324618066
rmse:1194.0899180806305
maxe:4249.749983079091
med:802.2861230188137
mae:952.1767853124898
mape:0.2875528430585596
------ Lasso -----
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_coordinate_
descent.py:631: ConvergenceWarning: Objective did not converge. You migh
t want to increase the number of iterations, check the scale of the feat
ures or consider increasing regularisation. Duality gap: 1.917e+08, tole
rance: 2.047e+05
  model = cd_fast.enet_coordinate_descent(
```

```
r2:0.7342477188928976
mse:831461.0151979976
rmse:911.8448416249321
maxe:3628.5010443074716
med:489.2271755198162
mae:683.5284031807931
mape:0.17103062632090218
------ XGBRegressor -----
r2:0.8196949268678938
mse:564121.7397165124
rmse:751.0803816613188
maxe:3317.3564453125
med:373.423583984375
mae:511.89759465997867
mape:0.14640597095033023
Best: float64 r2=0
```







4.4 Dane PREPROCESSED - Przetwarzanie wstępne & regresja

4.4.1 Czy atrybuty są skorelowane z wartością wyjściową?

TODO 4.4.1

dla

- season
- mnth
- weekday
- weathersit
- holiday
- workingday

oblicz współczynnik Pearsona i wyświetl zależność cnt od atrybutu (wykres typu scatter)

Wyniki możesz zebrać w postaci tabelki (wprowadzić do DateFrame i sformatować).

In [11]:
```
import pandas as pd
wyniki ={'Atrybut':['season','mnth','other'],
         'r':[.1,.2,.3]}
wyniki = pd.DataFrame(wyniki)
wyniki.head()
```

Out[11]:

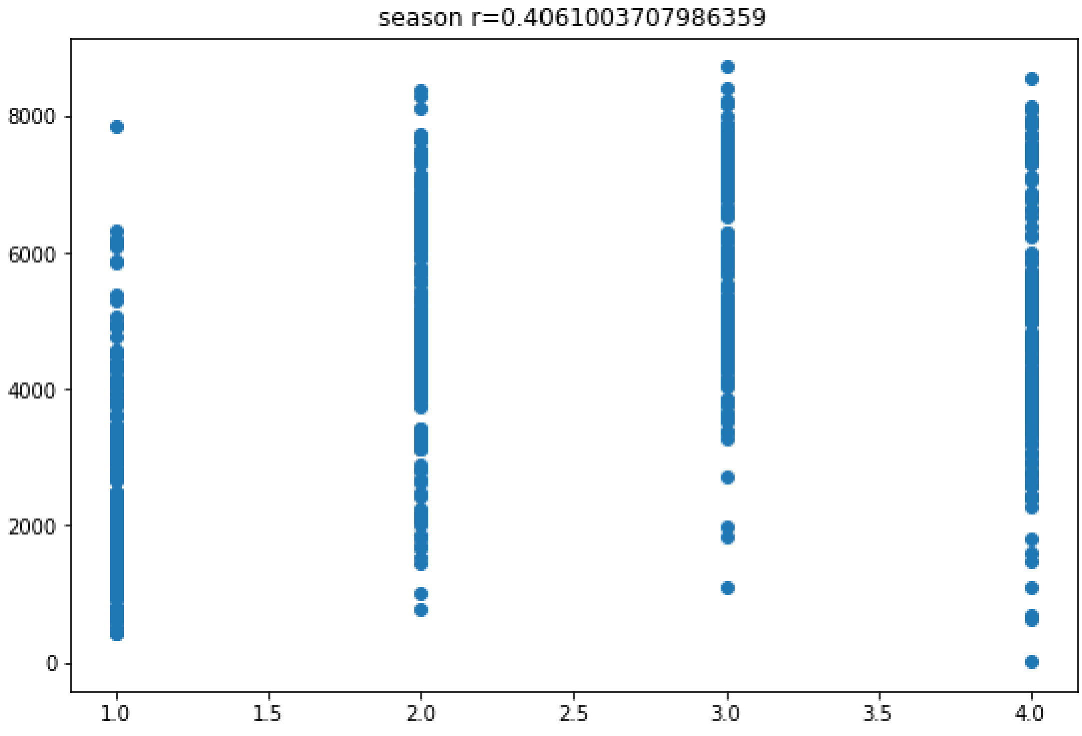
	Atrybut	r
0	season	0.1
1	mnth	0.2
2	other	0.3

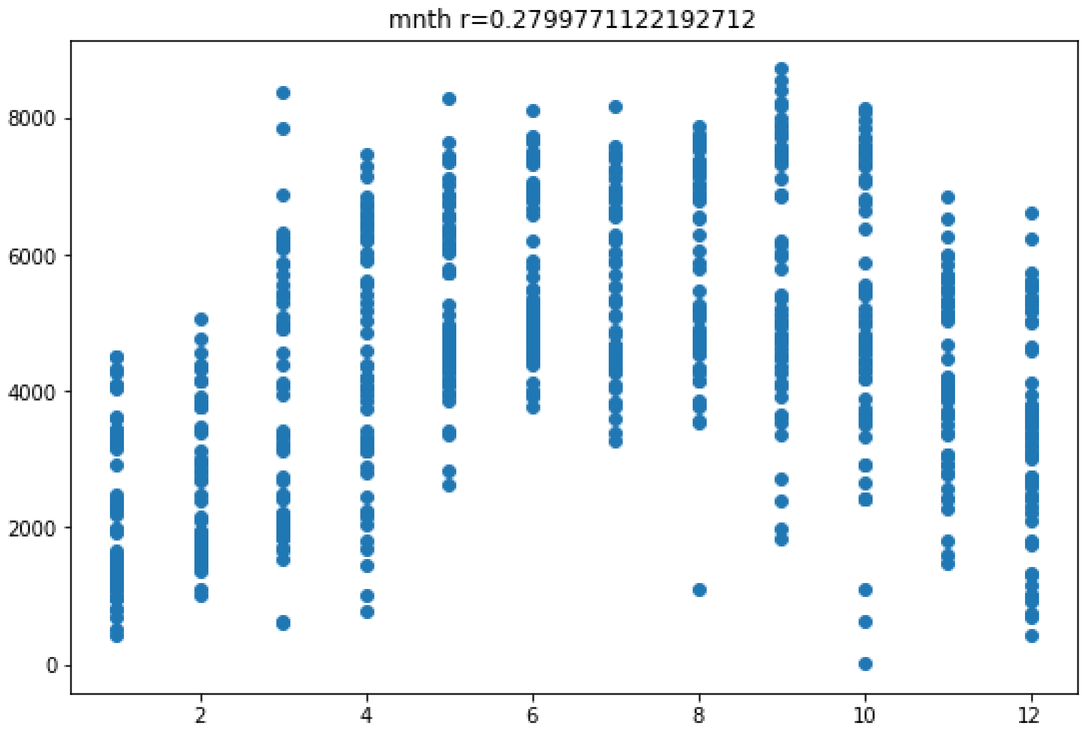
lub zapisać bezpośrednio jako tabelkę w języku markdown lub html

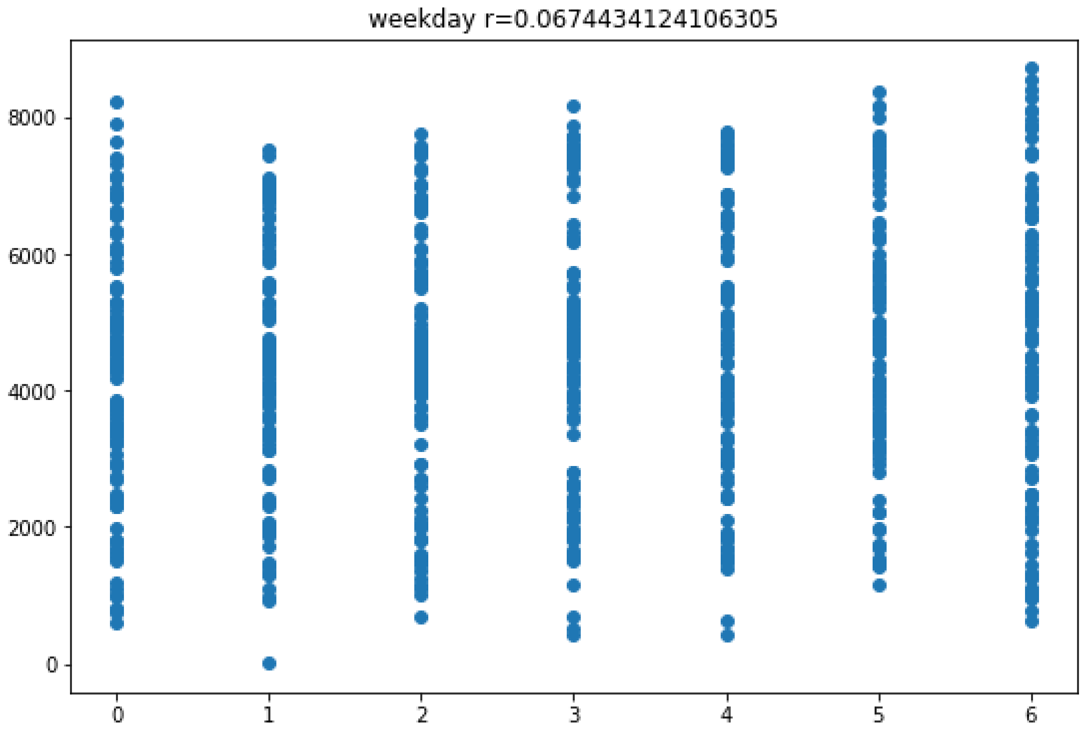
Atrybut	r
season	.1
mnth	.2
other	.3

In [12]:

```
import scipy.stats
plt.rcParams["figure.figsize"] = (9,6)
series=[df.season, df.mnth, df.weekday]

for s in series:
  r,p = scipy.stats.pearsonr(s,df.cnt)
  plt.figure()
  plt.scatter(s,df.cnt)
  plt.title(f'{s.name} r={r}')
  plt.show()
```







4.4.2 Konwersja one hot

TODO 4.4.2

1. Dla wybranych przez siebie atrybutów dokonaj konwersji one-hot za pomocą funkcji pd.get_dummies(). Oczywiście usuń też ten atrybut z DataFrame.

Jakie miałyby być te wybrane atrybuty? Np. do month ma słabą korelację, ale średnie wartości cnt zmieniają się w zależności od miesiąca. 2. df.dteday to sa wartości liczone w nanosekundach od 1970. Przeskalujmy je dzieląc przez 1e18 2. Dodaj cechy wielomianowe pochodne de.dteday

Konwersja one-hot zamienia atrybut dyskretny z k wartościami $\{a_1, \ldots, a_k\}$, na k kolumn, w których umieszane są zera i jedynki. Wartość x' po konwersji jest ustalana jako

- $x'[i,j] = 1$, jeżeli przed konwersją $x[i] = a_j$,
- $x'[i,j] = 0$, jeżeli przed konwersją $x[i] \neq a_j$

In [27]:
```
df_season = pd.get_dummies(df.season, prefix='season_')
df_month = pd.get_dummies(df.mnth, prefix='month_')
# df_hr = pd.get_dummies(df.hr, prefix='hr_')
df_weekday = pd.get_dummies(df.weekday, prefix='weekday_')
df_weathersit = pd.get_dummies(df.weathersit, prefix='weathersit_')
df_workingday = pd.get_dummies(df.workingday, prefix='workingday_')

# usuń te, które podlegają konwersji one-hot
df2 = df.drop(columns=['instant','casual','registered','cnt'])
# Polynomial features, normalziacja
df2['dteday']=pd.to_numeric(df2.dteday)/1e18

degree=6 # dobierz wartość eksperymentalnie
for i in range(2,degree):
  df2['dteday'+str(degree)]=df2.dteday**degree
# konkatenacja kilku data frame w jedną
df3 = pd.concat([df2,df_season,df_month, df_weekday, df_weathersit, df_wo
df3.head()

# Konwersja na numpy
X=df2.to_numpy()
y=df.cnt.to_numpy()
```

TODO 4.4.2

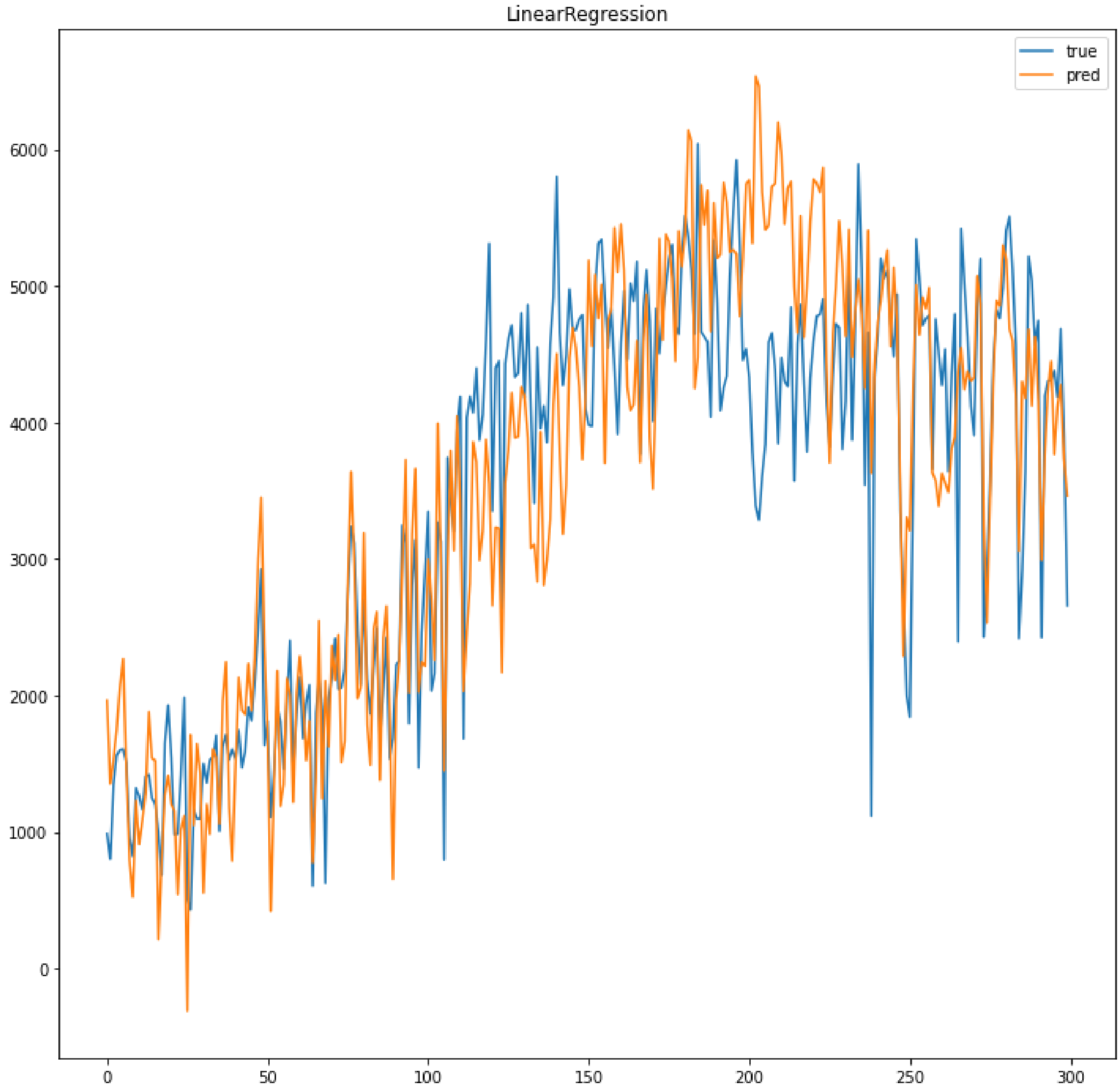
a. Znajdź najlepszą metodę predykcji

b. wyświetl wykresy dla wszystkich metod

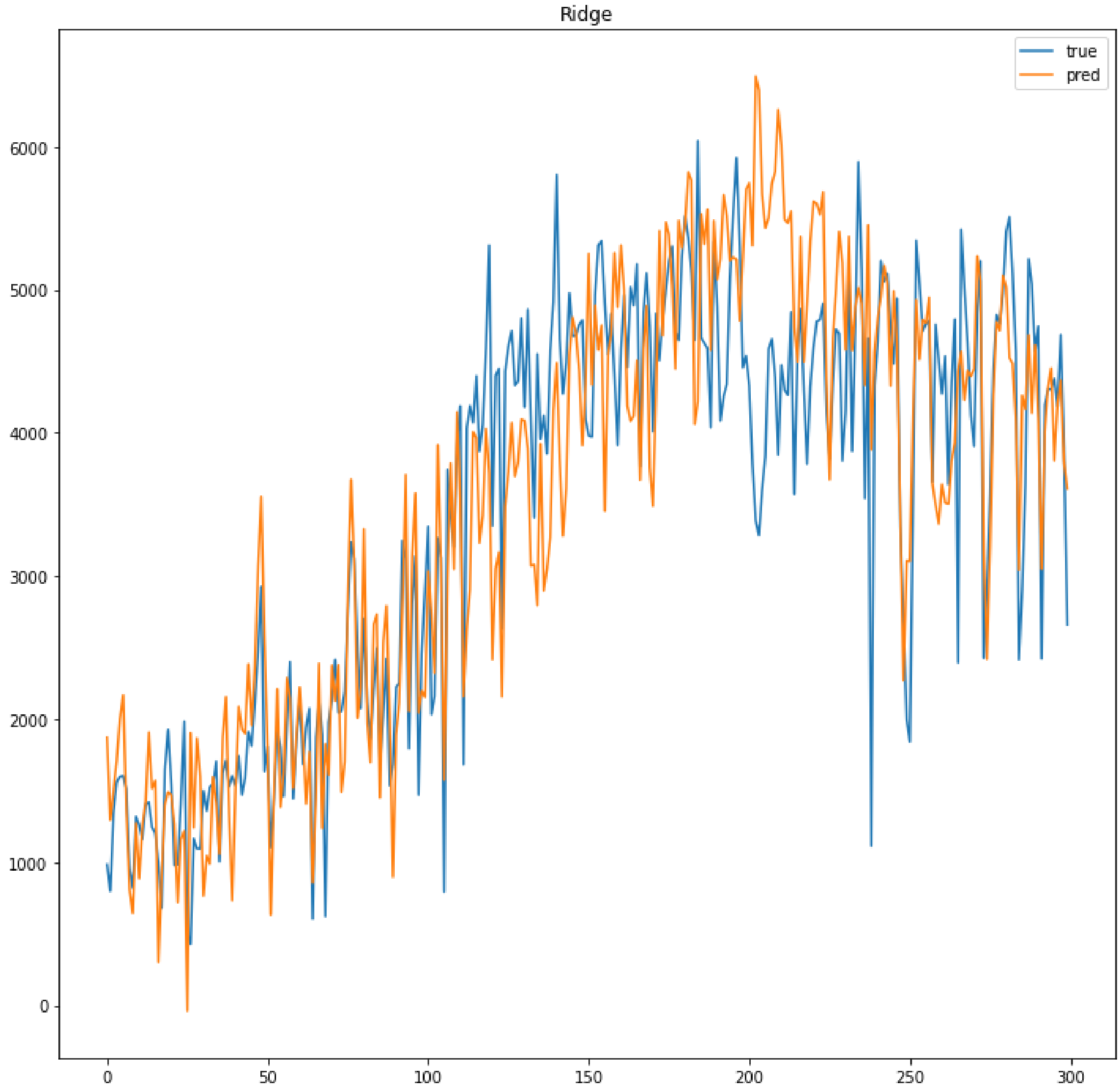
c. możesz wybrać bardziej interesujące fragmenty wykresu do wyświetlenia (parametry start=, end= funkcji plot)

In [28]:
```
predictors =[LinearRegression(),
            Ridge(solver='svd'),
            Lasso(max_iter=10000),
            XGBRegressor()
            ]

best_reg = None
for reg in predictors:
  print(f'------ {reg.__class__.__name__} -----')
  r2 = train_and_test(X,y,reg)
  if r2>max_r2:
    best_reg = r2

for i in range (0,3):
  print(f'Best: {best_reg.__class__.__name__} r2={best_reg}')
  plot(X,y,predictors[i],0,300)
```

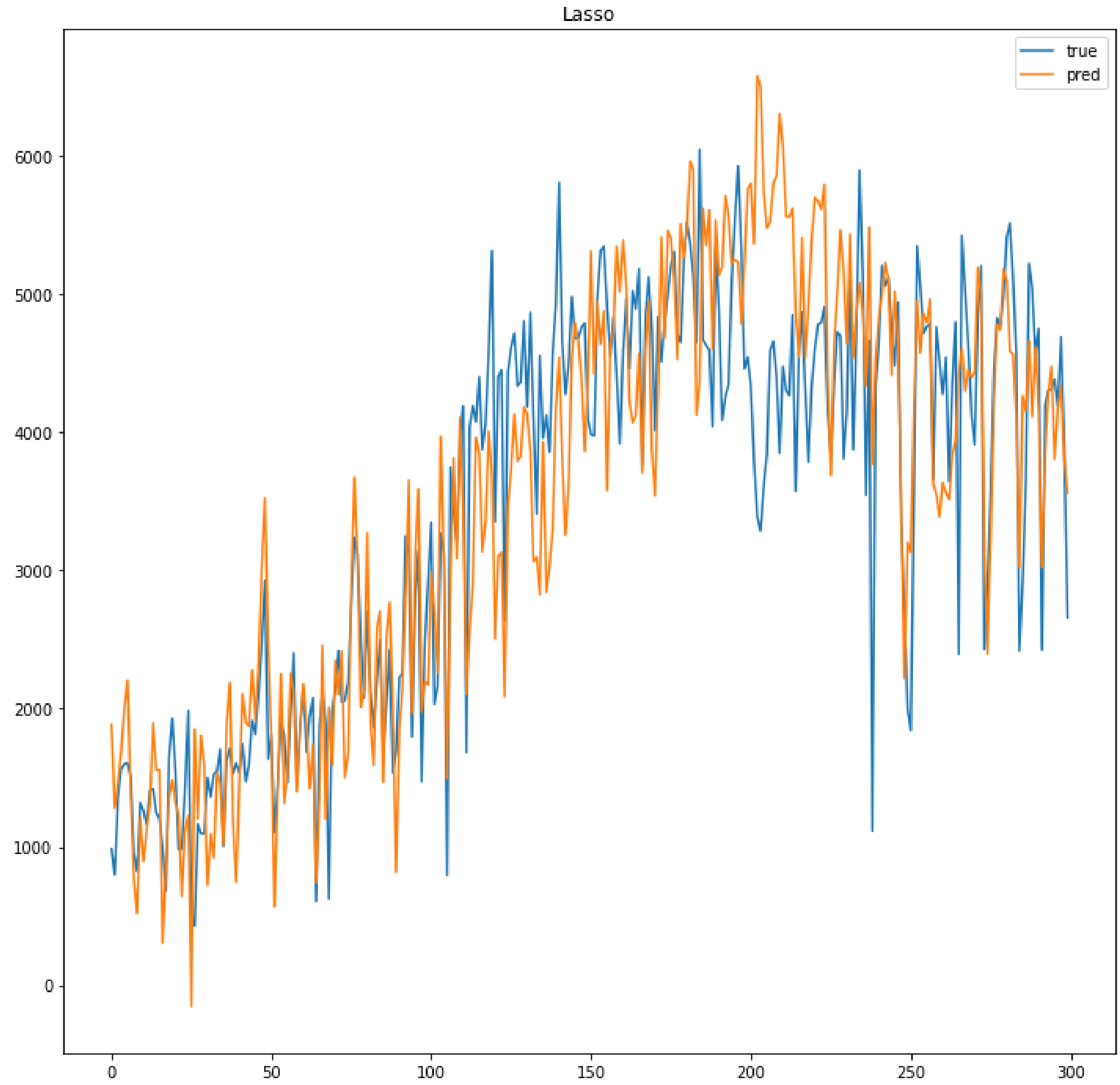
```
------ LinearRegression -----
r2:0.7303675431126788
mse:843600.9482210644
rmse:918.4775164483148
maxe:3691.5488313433016
med:521.7964163817305
mae:688.0868092715275
mape:0.1723821714572377
------ Ridge -----
r2:0.7395176072893664
mse:814973.0786209182
rmse:902.7585937674137
maxe:3484.122999752686
med:494.0441420496169
mae:679.9776511193787
mape:0.1708506331214793
------ Lasso -----
r2:0.734155932566067
mse:831748.1877188503
rmse:912.0022958956026
maxe:3587.778192869624
med:501.7873141001403
mae:683.0224823816452
mape:0.17048840381525843
------ XGBRegressor -----
r2:0.8190069576886221
mse:566274.1936854463
rmse:752.5119226201311
maxe:3317.3564453125
med:376.391357421875
mae:514.2312133789062
mape:0.1467634211922126
Best: float64 r2=0.8190069576886221
```




```
Best: float64 r2=0.8190069576886221
```




```
Best: float64 r2=0.8190069576886221
```



4.5 Dane TIMESERIES - regresja szeregów czasowych

Interesuje nas zależność cnt od dteday. Dodamy cechy odpowiedzialne za ogólny trend i oscylacje

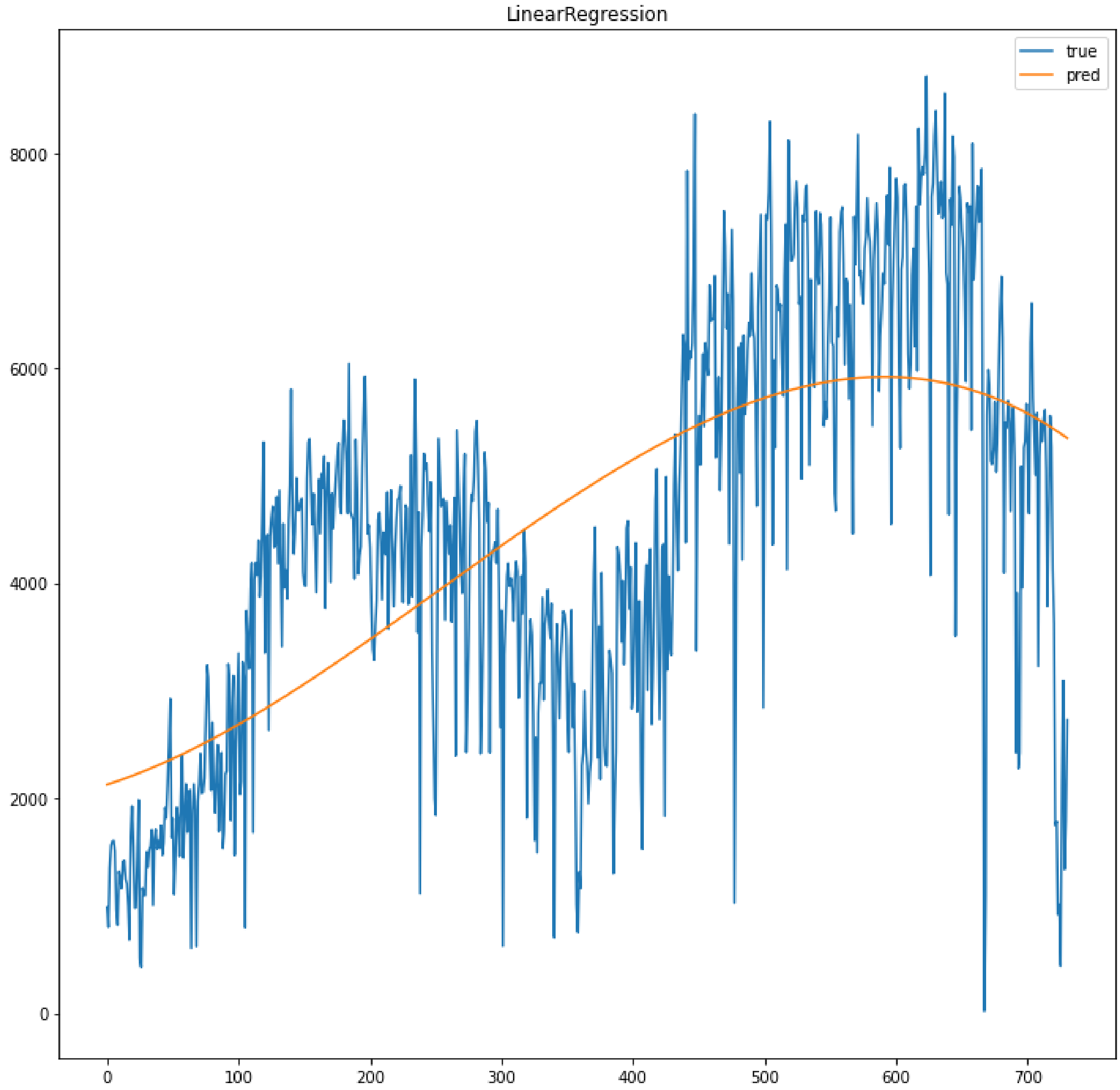
4.5.1 Dopasuj krzywą trendu

1. Zastosuj cechy wielomianowe wybranego przez siebie stopnia
2. Oblicz metryki

In [39]:
```
from sklearn.preprocessing import PolynomialFeatures
df2.info()
df2.max()

df2.shape

x=df2.dteday.to_numpy()
y=df.cnt
```

```
# X = np.stack((x,x**2,x**3,...),axis=-1)
poly = PolynomialFeatures(degree=3)
X=poly.fit_transform(x.reshape(-1,1))

regr = LinearRegression()
regr.fit(X,y)
print(regr.coef_)
plot(X,y,regr)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   dteday      731 non-null    float64
 1   season      731 non-null    int64
 2   yr          731 non-null    int64
 3   mnth        731 non-null    int64
 4   holiday     731 non-null    int64
 5   weekday     731 non-null    int64
 6   workingday  731 non-null    int64
 7   weathersit  731 non-null    int64
 8   temp        731 non-null    float64
 9   atemp       731 non-null    float64
 10  hum         731 non-null    float64
 11  windspeed   731 non-null    float64
 12  dteday6     731 non-null    float64
dtypes: float64(6), int64(7)
memory usage: 74.4 KB
[ 0.00000000e+00 -2.05642838e+08  1.56382973e+08 -3.96213528e+07]
```

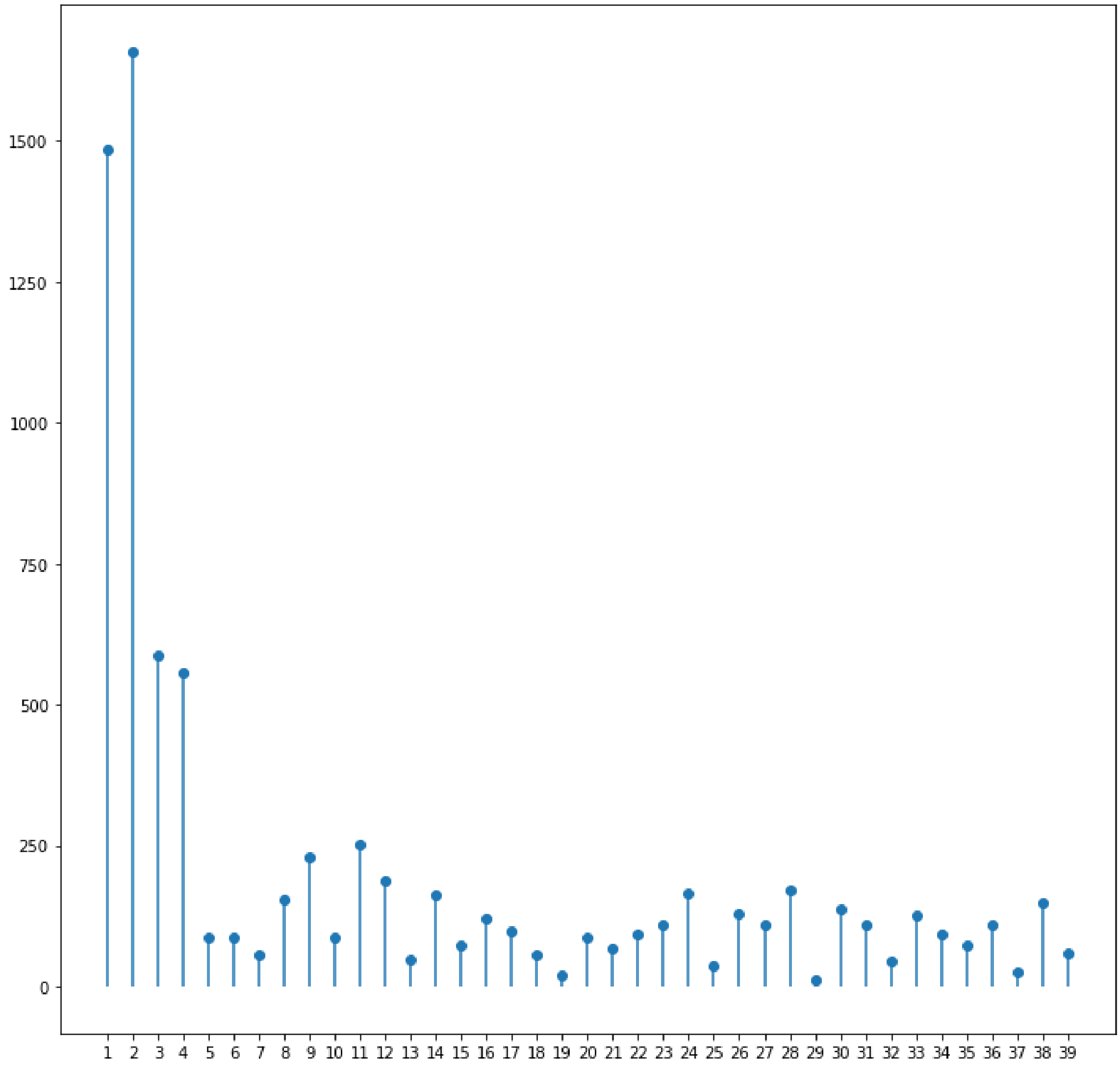


4.5.2 Czy przebieg jest periodyczny?

Zastosujemy transformację Fouriera i zobaczymy, dla jakich częstotliwości mamy duże amplitudy?

In [40]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack

y=df.cnt.to_numpy()
x = np.arange(-y.shape[0]//2,y.shape[0]//2)

yf = scipy.fftpack.fft(y)

N=y.shape[0]

yf = scipy.fftpack.fft(y)
yf[0]=0
xf = np.arange(1,40)

plt.scatter(xf, 2.0/N * np.abs(yf[1:40]))
plt.vlines(xf, np.zeros(39),2.0/N * np.abs(yf[1:40]))
```

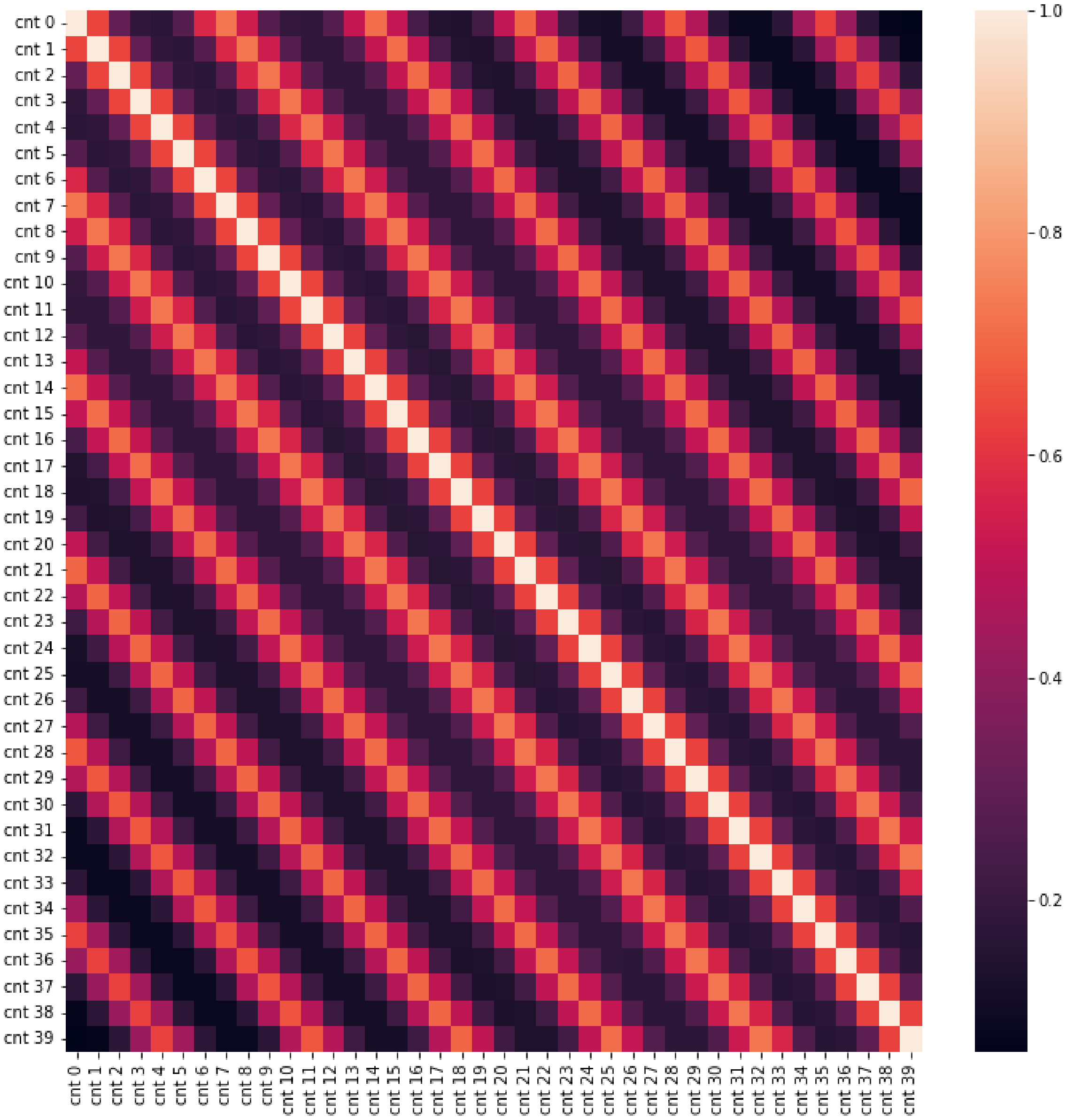
```
plt.xticks(xf)
plt.show()
```



Autokorelacja

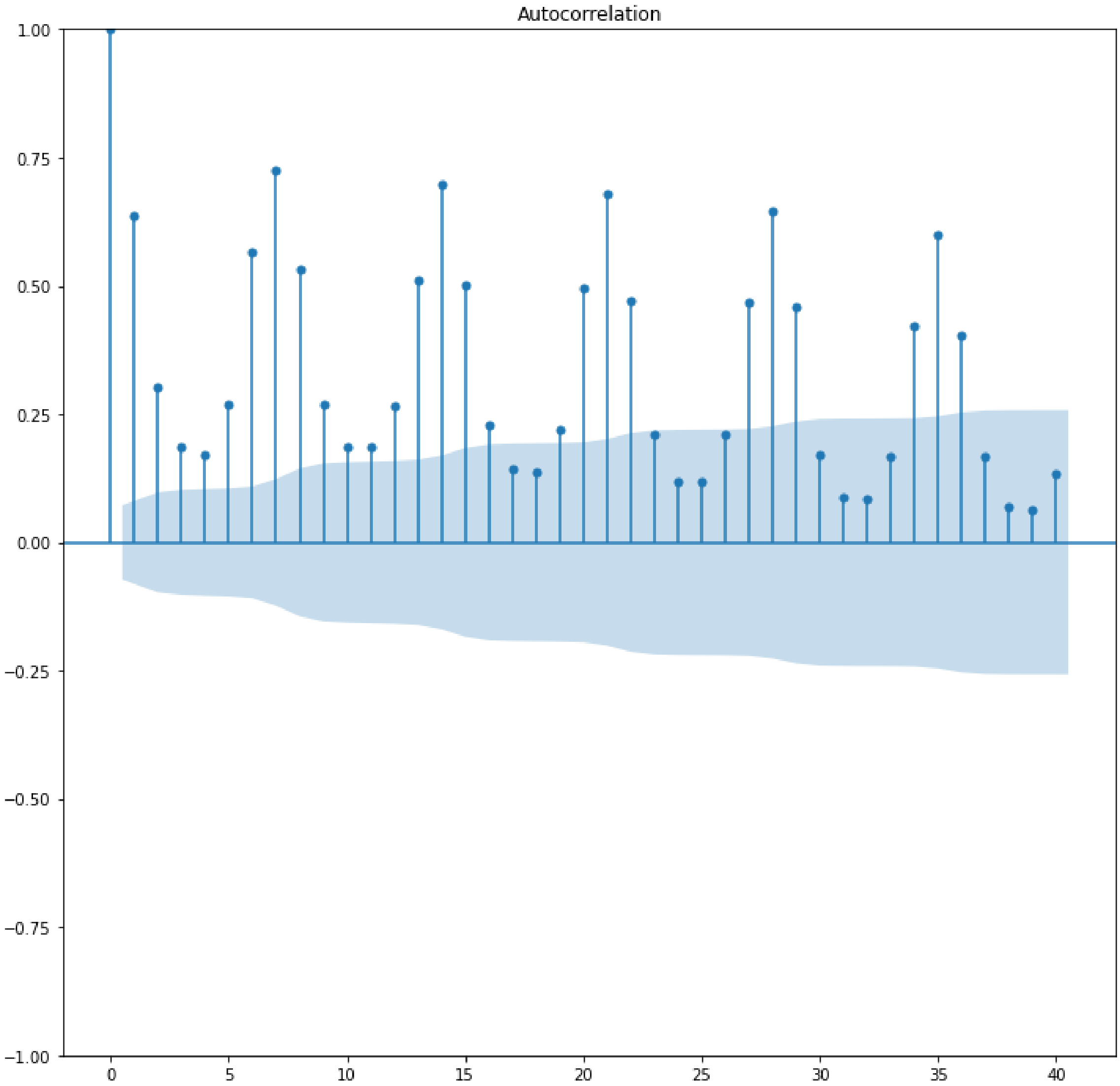
In [41]:
```
shifted = [pd.DataFrame(data = df.casual).shift(i) for i in range(40)] #c
for i in range(len(shifted)):
  shifted[i].columns=['cnt '+str(i)]
# print(shifted)
df_shifted = pd.concat(shifted,axis=1)
# df_shifted.head(20)

corr_mat = df_shifted.corr()
import seaborn as sn
plt.rcParams['figure.figsize'] = (12, 12)
sn.heatmap(corr_mat,xticklabels=df_shifted.columns,yticklabels=df_shifted
```

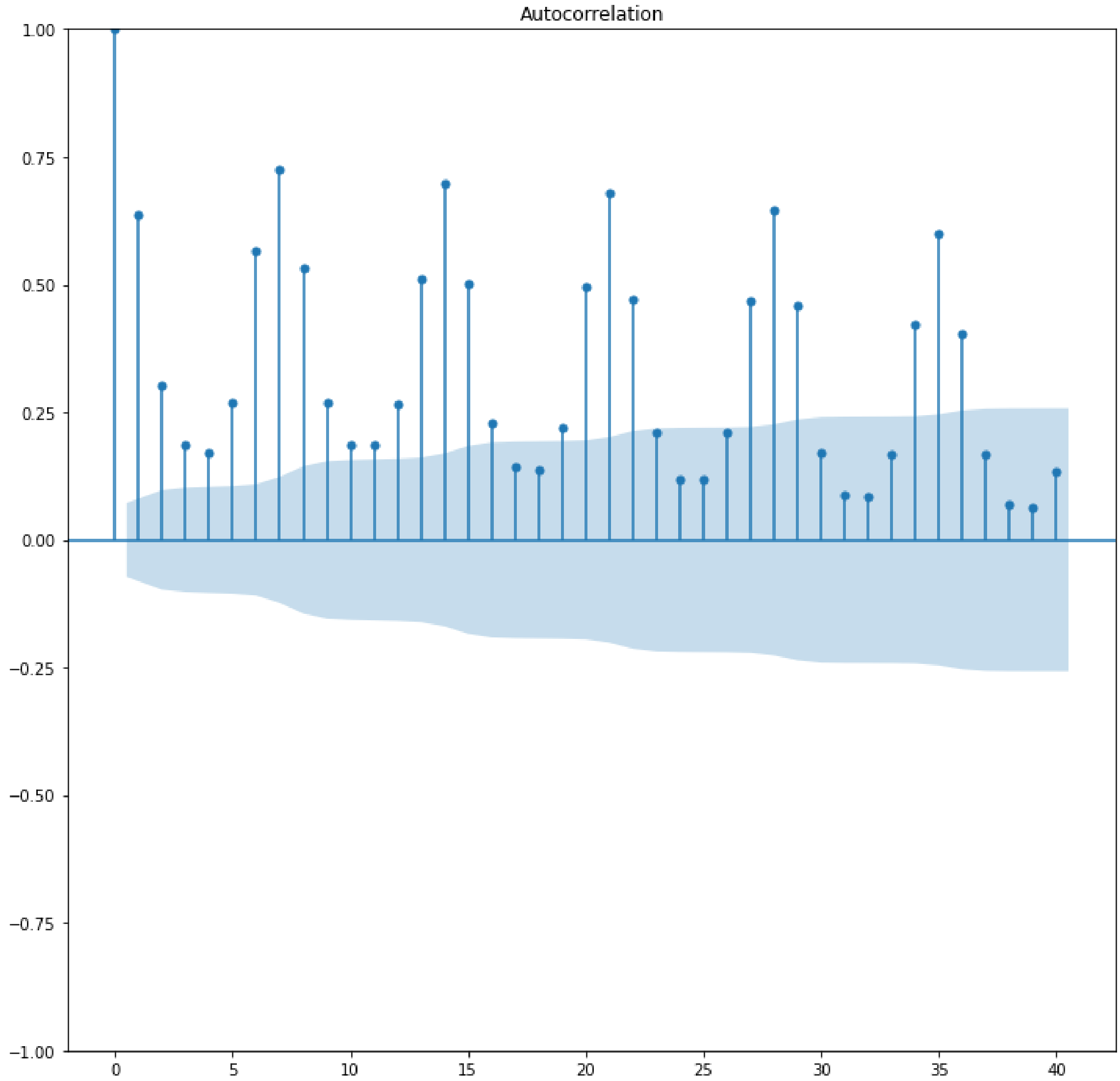
Out[41]: <Axes: >



In [42]:
```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(df.casual,lags=40)
```

Out[42]:


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40



4.5.3 Dodajemy cechy okresowe

Dodajemy ortogonalne funkcje $\cos(\frac{2\pi x}{T_i})$ i $\sin(\frac{2\pi x}{T_i})$

Okres T_i to wielokrotność jednego dnia. Ile to jeden dzień? Zastosowaliśmy wcześniej konwersję, dzieląc przez 1e18, ale dane są w nanosekundach. 1ns = 1e-9s.

TODO 4.5.1

Oblicz okres T dla jednego dnia dla obecnego skalowania Oczekiwany wynik:

```
8.6400000000042e-05
8.639999999981995e-05
8.64e-05
```

In [33]:

```
T = 24*60*60/1e9 #godziny razy minuty razy sekundy
#print(x[2]-x[1])
#print(df.dteday[1]-df.dteday[0])
print(T)
```

```
8.64e-05
```

Przygotowujemy dane dla liniowej regresji

- wpierw cechy wielomianowe
- następnie okresowe

In [46]:
```
flist = [x]
T=1
degree=10
for i in range (2,degree):
  flist.append(x**i)

# Wybierz okresy
periods=[1,2,3,4,9,11,24,28,90]

for p in periods:
  flist.append(np.cos(2*np.pi*x/T/p))
  flist.append(np.sin(2*np.pi*x/T/p))

X=np.stack(flist,axis=1)
```

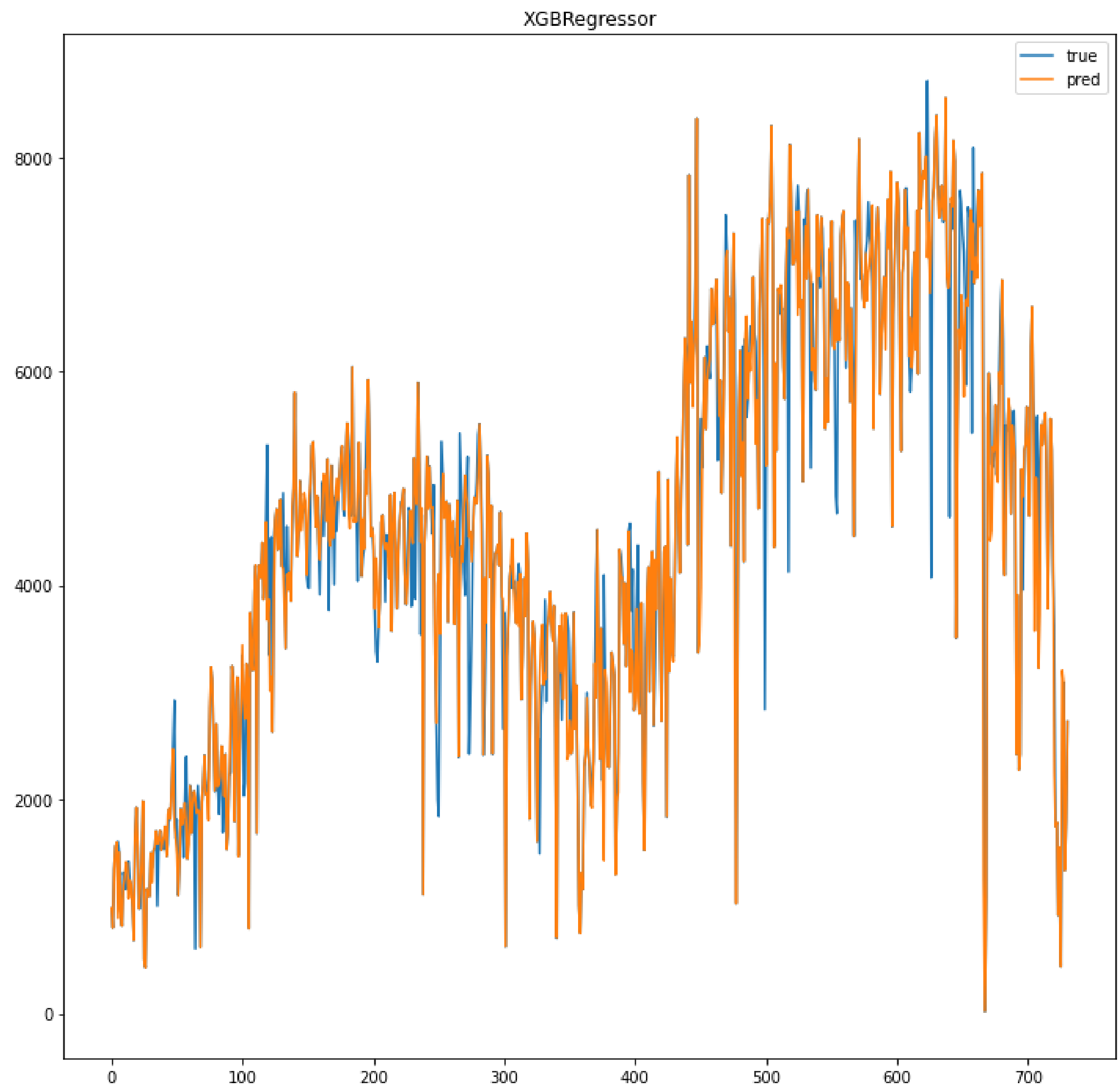
TODO 4.5.2

Przetestuj kilka algorytmów regresji, zrób wykresy, wybierz najlepszy

Przetestuj wszystkie 4, poniżej przykład dla XGBRegressor, który zwykle okazywał się najlepszy

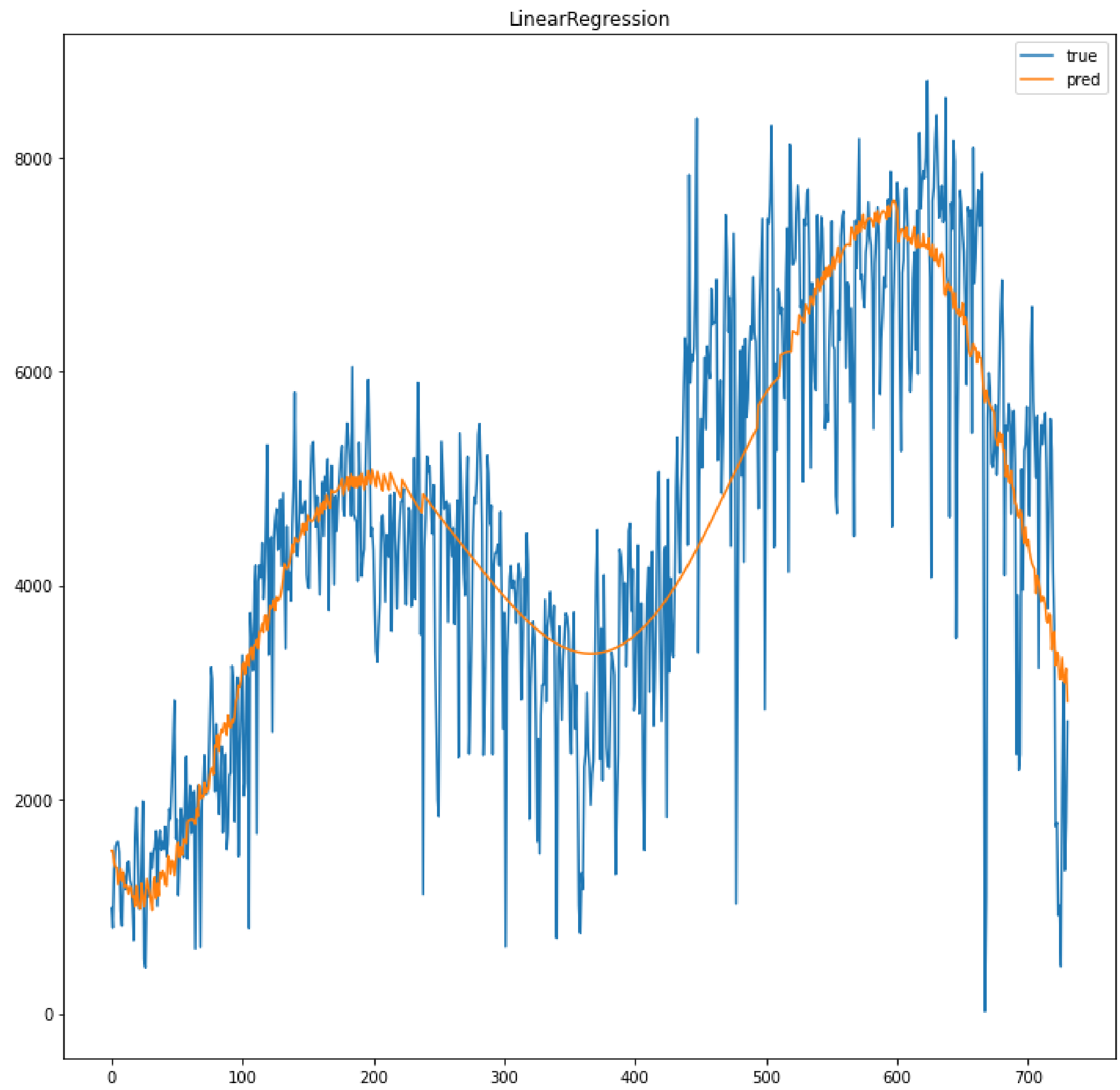
In [50]:
```
regr = XGBRegressor()
train_and_test(X,y,regr)
plot(X,y,regr)
```

```
r2:0.7150050584830872
mse:891665.6610162712
rmse:944.2804991189171
maxe:3533.0810546875
med:491.8321533203125
mae:689.4961750377308
mape:0.18718394540839065
```



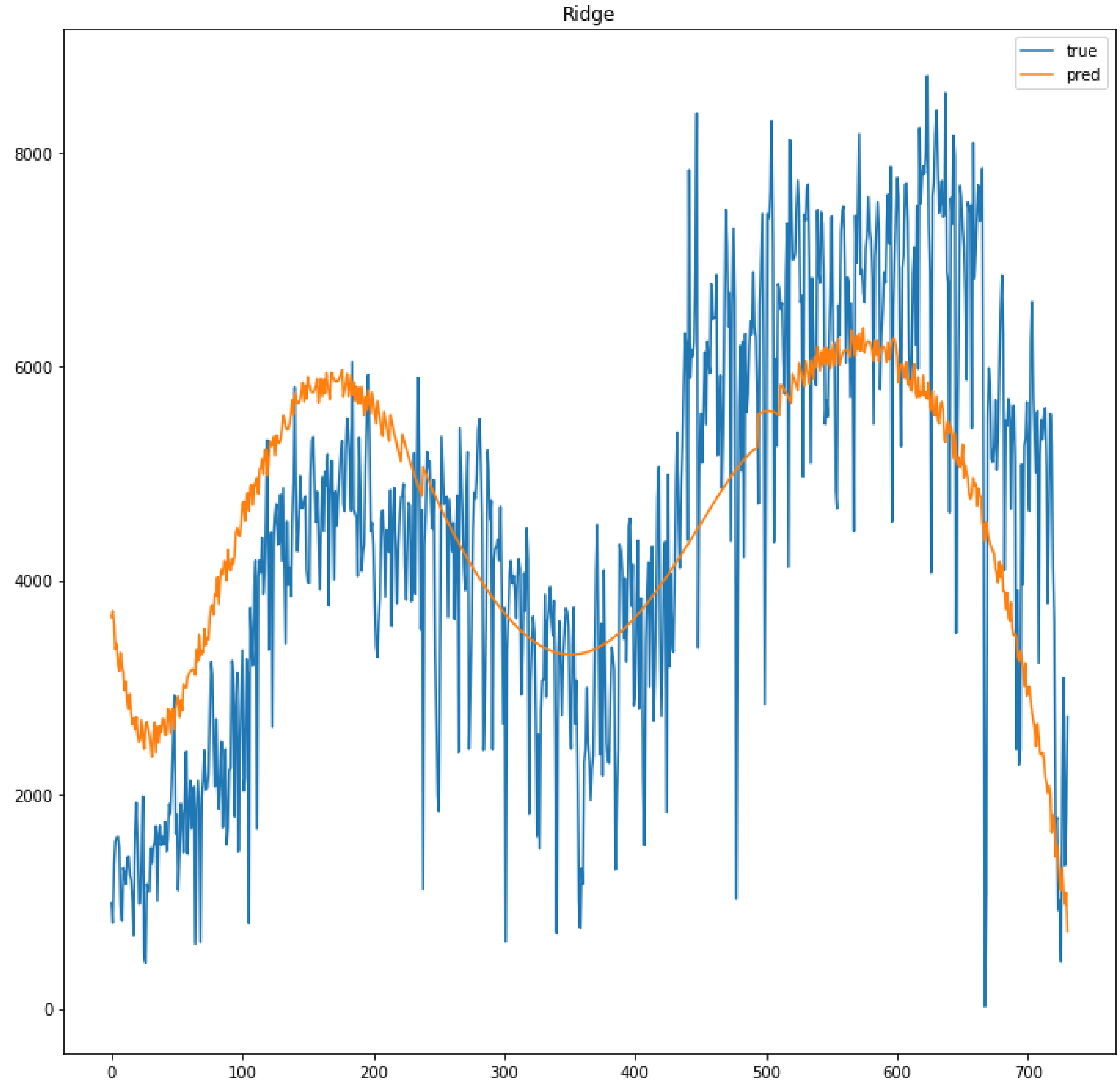
In [47]:
```
regr = LinearRegression()
train_and_test(X,y,regr)
plot(X,y,regr)
```

```
r2:0.6879129727073827
mse:976428.8587169902
rmse:988.1441487541128
maxe:3000.002961329237
med:607.3702839599543
mae:757.9700429267363
mape:0.20380782878391096
```



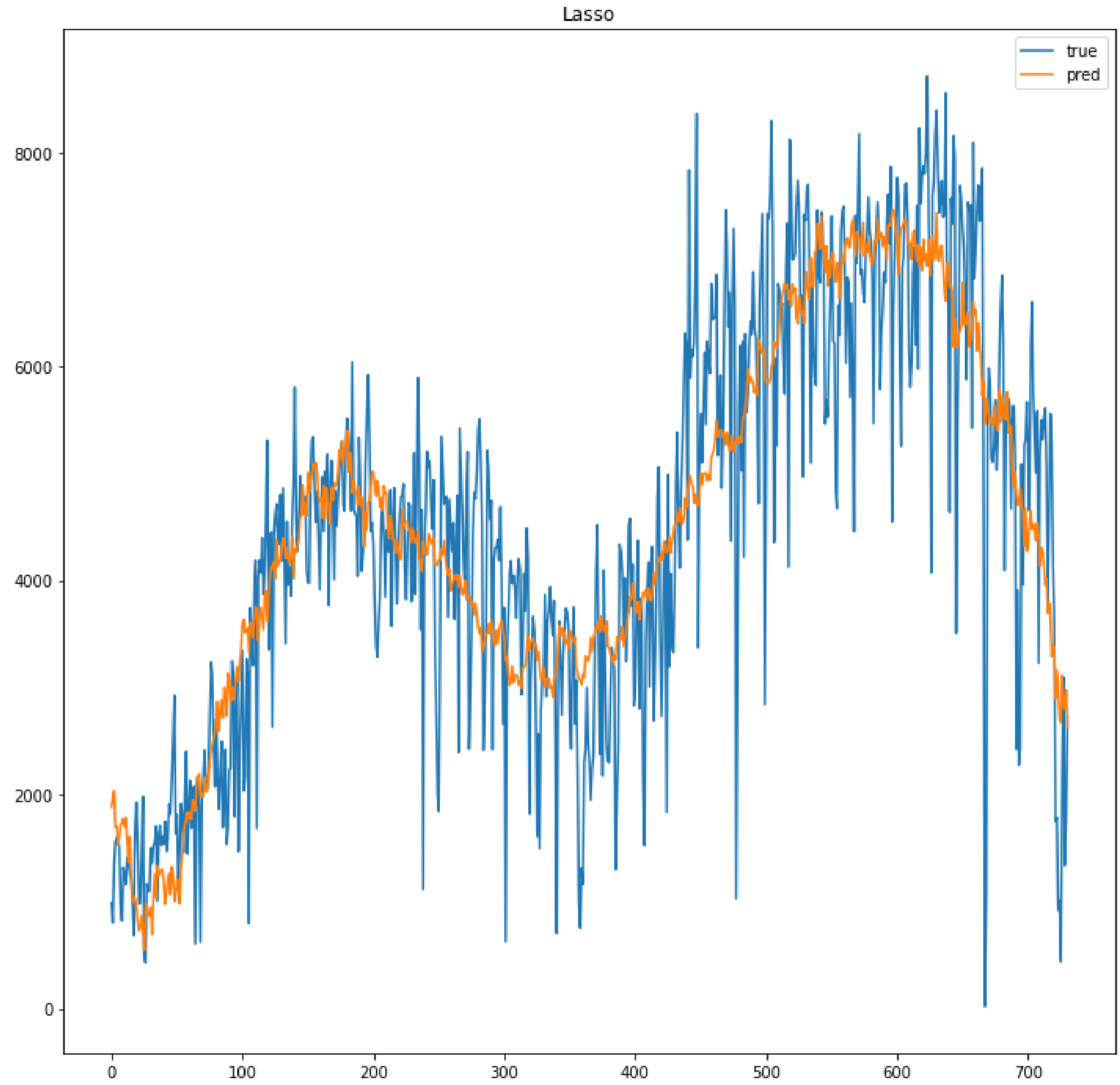
In [48]:
```
regr = Ridge(solver='svd')
train_and_test(X,y,regr)
plot(X,y,regr)
```

```
r2:0.3704749197084104
mse:1969599.5088783351
rmse:1403.424208455282
maxe:3130.2497429977157
med:1074.4418651613585
mae:1196.8196013763795
mape:0.340439335451479
```



In [49]:
```
regr = Lasso(max_iter=10000)
train_and_test(X,y,regr)
plot(X,y,regr)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_coordinate_
descent.py:631: ConvergenceWarning: Objective did not converge. You migh
t want to increase the number of iterations, check the scale of the feat
ures or consider increasing regularisation. Duality gap: 2.661e+08, tole
rance: 2.047e+05
  model = cd_fast.enet_coordinate_descent(
r2:0.7202310407948631
mse:875315.0937125618
rmse:935.5827562073607
maxe:3045.2705556392975
med:575.6724435528786
mae:725.9371733690376
mape:0.200822643746154
```



4.6. Napisz wnioski

TODO 4.6.1

Możesz zestawić dane w tabelce

1. Jakie wyniki r2/MAPE uzyskano dla różnych zestawów danych BASIC, PREPROCESSED i TIMESERIES i algorytmów?
2. Która postać danych najlepsza, jeżeli ograniczyć się wyłacznie do metode liniowej regresji?
3. Czego nie uwzgledniaja dane TIMESERIES, a co uwzględniają pośrednio?
4. Jak oceniasz Ridge i Lasso dla TIMESERIES, skąd taki wynik?

4.6.1.1

BASIC

algorytm	r2	mape
Linear Regression	0.428	0.329
Ridge	0.544	0.287
Lasso	:0.734	0.171
XGBRegressor	0.819	0.146

PREPROCESSED

algorytm	r2	mape
Linear Regression	0.730	0.172
Ridge	0.739	0.170
Lasso	0.733	0.171
XGBRegressor	0.819	0.146

TIMESERIES

algorytm	r2	mape
Linear Regression	0.687	0.203
Ridge	0.370	0.340
Lasso	0.72	0.20
XGBRegressor	0.715	0.187

4.6.1.2

W przypadku regresji liniowej, najlepsze rezultaty otrzymano dla zestawu danych preprocessed. Uzyskany wynik dla r2 wynosil 0.73. W pozostalych przypadkach, wyniki byly na poziomie ~0.4 dla r2.

4.6.1.3

Dane TIMESERIES nie uzwgledniaja sytuacji pogodowych. Natomiast uzgledniaja posrednio zachowania typu oscylacje tygodniowe, miesieczne czy roczne np. w zaleznosci od sytuacji pogodowej.

4.6.1.4

Lasso dla timeseries uzyskal zdecydowanie lepsze wyniki niz Ridge. Wynosily kolejno 0.72 i 0.2 odpowiednio dla r2 i mape.

> Indented block