Relazione

July 5, 2019

1 Progetto di Applicazioni Data Intensive 2018/2019

Orazi Filippo 0000801069
Alesiani Matteo 0000806466

1.1 Descrizione del problema

Il problema da noi analizzato si pone l'obbiettivo di prevedere per conto di una società che effettua bike sharing (ossia una forma di affitto di biciclette automatizzato) il numero di biciclette che saranno affittate durante una giornata che presenta determinate caratteristiche.

Il dataset denominato "Bike Sharing Data Set" è stato scaricato dal sito https://archive.ics.uci.edu/ml e si compone di 731 istanze, 16 diversi attributi di cui 2 identificatori e 3 possibili soggetti di predizione.

[1]:
```
import os.path
import csv
import pandas as pd

if os.path.exists("day.csv"):
    ds = pd.read_csv("day.csv", sep=",")
else:
    print("File non trovato")
```

[2]:
```
len(ds.columns)
```

[2]:
```
16
```

1.1.1 Descrizione variabili

Il dataset scelto presenta variabili strutturate, ovvero i cui valori sono noti.

Vengono di seguito descritte:

- **instant**: indice dei record.
- **dteday**: data.
- **season**: stagione
- **yr**: anno (0: 2011, 1: 2012)
- **mnth**: mese (1 - 12)
- **holiday**: giorno festivo (1: si, 0: no)
- **weekday**: giorno della settimana (0 - 6)

- **workingaday**: se il giorno è festivo o appartiene al weekend 0, altrimenti 1
- **wheathersit**: condizioni meteo generali della giornata:
  - 1: soleggiato, poco nuvoloso
  - 2: nuvoloso, nebbia
  - 3: leggera neve, leggera pioggia
  - 4: neve, pioggia, fulmini
- **temp**: temperatura media giornaliera (C) normalizzata. Valori divisi per 41.
- **atemp**: temperatura perepita (řC) normaizzata. Valori divisi per 50
- **hum**: percentuale di umidità
- **windspeed**: velocità del vento normalizzata, Valori divisi per 67.
- **casual**: numero di utenti casuali
- **registered**: numero di utenti registrati
- **cnt**: numero di utenti totali

La variabile scelta come oggeto di predizione è la variabile "cnt" in quanto a fini di ricerca di mercato è la variabile che più interessa. Vengono quindi esclusi gli attributi "casual" e "registered", di cui "cnt" è la somma, e le relative colonne.

Osserviamo come la variabile da predire sia di tipo continuo. La metodologia utilizzata, per la risoluzione del problema, attua un algoritmo di regressione.

Concludianmo la descrizione delle variabili definendo come indice del dataframe l'attrbuto "dteday" e eliminando "instant", poichè svolge la stessa funzione.

[3]:
```
ds.set_index(["dteday"], inplace=True)
dataset = ds.drop(["casual", "registered","instant"], axis=1)
Y = dataset["cnt"]
```

1.2 Analisi esplorativa

Il compito dell'analisi esplorativa consiste nel identificare nel dataset le caratteristiche degli attributi (feature) che possono influenzare la creazione del modello, ad esempio la presenza di valori nulli comporterebbe l'esigenza di attuare contromisure volte alla loro gestione.

Altro compito dell'analisi sta nel rappresentare l'insieme dei dati attraveso indici di correlazione tra variabili, grafici e descizioni matematiche.

Possiamo ottenere un inisieme di descrittori matematici dei vari attibuti attraverso la funzione *describe*.

[4]:
```
dataset.describe()
```

[4]:

	season	yr	mnth	holiday	weekday	workingday	\
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	
mean	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	
std	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	
50%	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	
75%	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	
max	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	

	weathersit	temp	atemp	hum	windspeed	cnt

```
count  731.000000  731.000000  731.000000  731.000000  731.000000   731.000000
mean     1.395349    0.495385    0.474354    0.627894    0.190486  4504.348837
std      0.544894    0.183051    0.162961    0.142429    0.077498  1937.211452
min      1.000000    0.059130    0.079070    0.000000    0.022392    22.000000
25%      1.000000    0.337083    0.337842    0.520000    0.134950  3152.000000
50%      1.000000    0.498333    0.486733    0.626667    0.180975  4548.000000
75%      2.000000    0.655417    0.608602    0.730209    0.233214  5956.000000
max      3.000000    0.861667    0.840896    0.972500    0.507463  8714.000000
```

La Funzione *info* di *pandas.Dataframe* permette di conoscere un ulteriore insieme di informazioni del dataframe

[5]:
```
dataset.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 731 entries, 2011-01-01 to 2012-12-31
Data columns (total 12 columns):
season        731 non-null int64
yr            731 non-null int64
mnth          731 non-null int64
holiday       731 non-null int64
weekday       731 non-null int64
workingday    731 non-null int64
weathersit    731 non-null int64
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
cnt           731 non-null int64
dtypes: float64(4), int64(8)
memory usage: 116.4 KB
```

Proseguiamo l'analisi del Dataset verificando la correlazione che l'attributo da predire ha in relazione agli altri. La correlazione di due variabili casuali X e Y, è dato dal rapporto tra la loro covarianza σXY e il prodotto delle deviazioni standard σX e σ ρ(X,Y)= $\frac{\sigma XY}{\sigma X \sigma Y}$

Il coefficiente ha un valore compreso tra 1 e -1, dove valori vicini a 1 indicano correlazione diretta (Y cresce al crescere di X) valori vicini a -1 indicano correlazione inversa (Y descresce al crescere di X) valori vicini a 0 indicano assenza di correlazione

Le seguanti funzioni consentono di calcolare la correlazione tra due serie e il grafico relativo:

[6]:
```
import numpy as np
def getCorrelation(feature1, feature2):
    return np.mean((feature1-feature1.mean()) * (feature2-feature2.mean())) /␣
 ↪(feature1.std() * feature2.std())
```

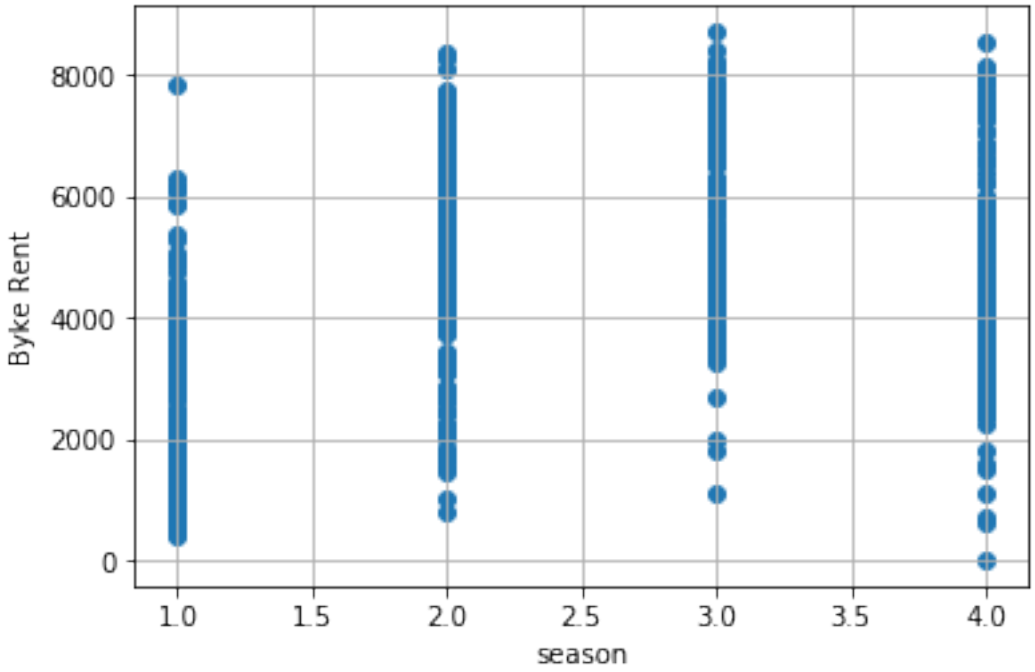
[7]:
```
import matplotlib.pyplot as plot
def plotData(x, y, XAxisName, YAxisName):
    plot.scatter(x, y)
    plot.grid()
    plot.xlabel(XAxisName); plot.ylabel(YAxisName)
```

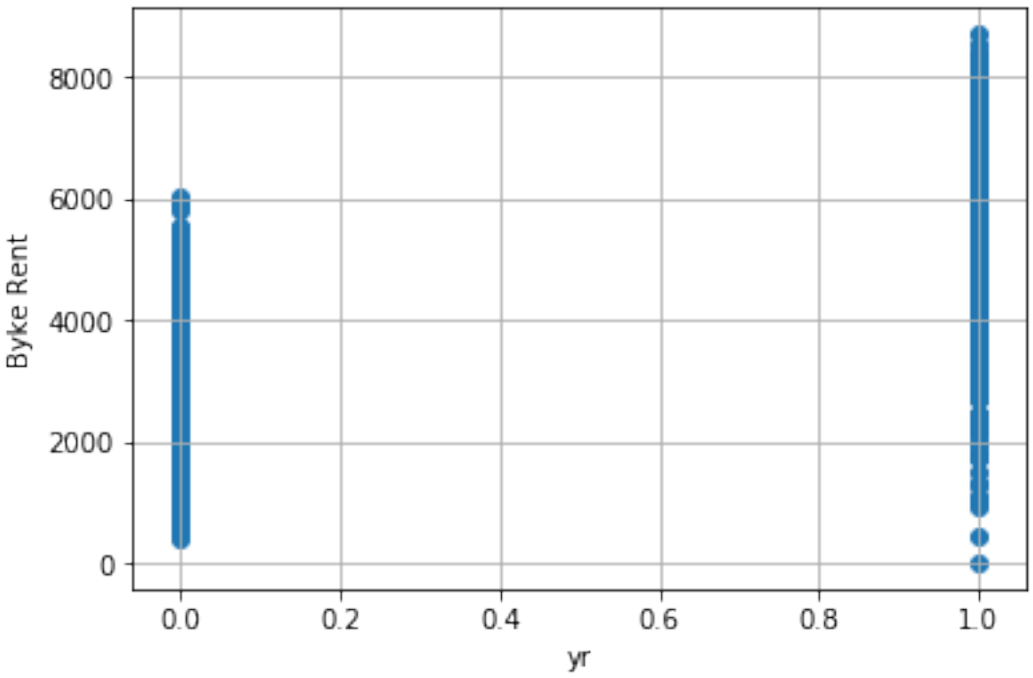
```
    plot.show()
```

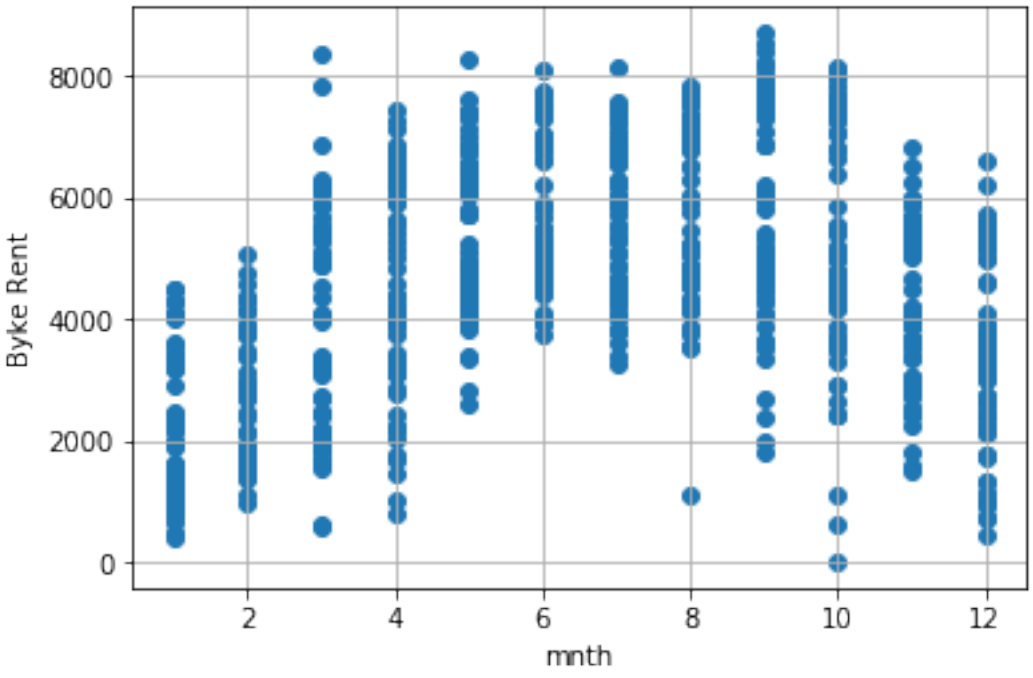
Mentre con la seguente si può ottenere una serie ordinata che indica la correlazione tra "cnt" e il nostro dataset indicizzato su "dtaday" e un grafico a dispersione di ogni feature con l'obbiettivo

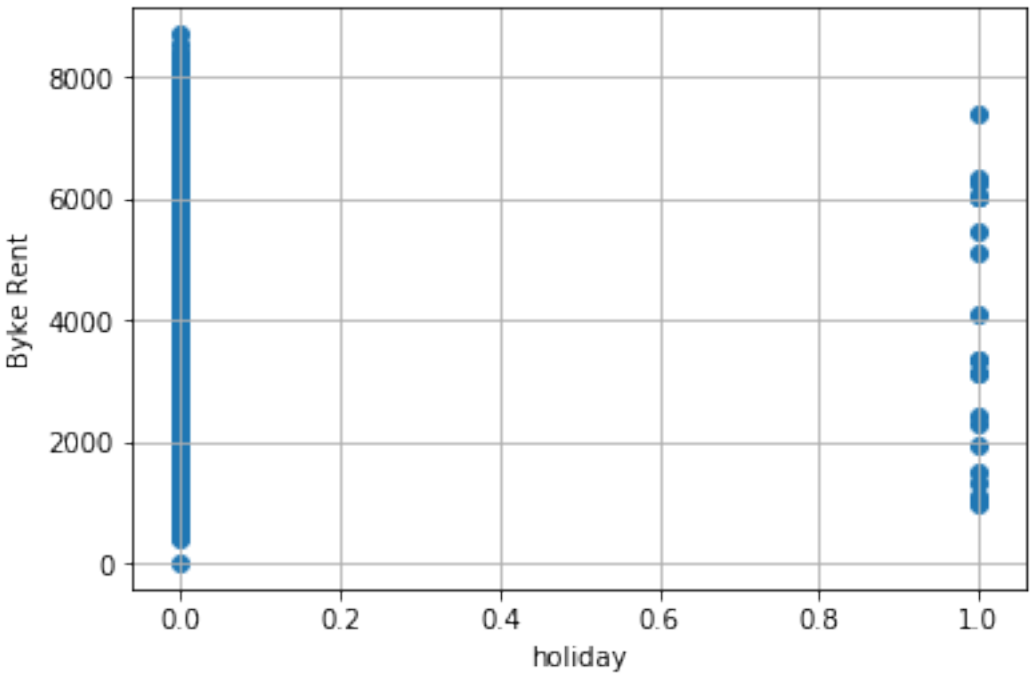
[8]:
```
def correlationRank(dataset, feature):
    correlation = []
    for a in dataset.columns:
        correlation.append(getCorrelation(dataset[a].astype("float"), feature))
        plotData(dataset[a].astype("float"), feature, a, "Byke Rent")
    cor = pd.Series(correlation, dataset.columns)
    cor.sort_values(ascending=False, inplace=True)
    return cor
```

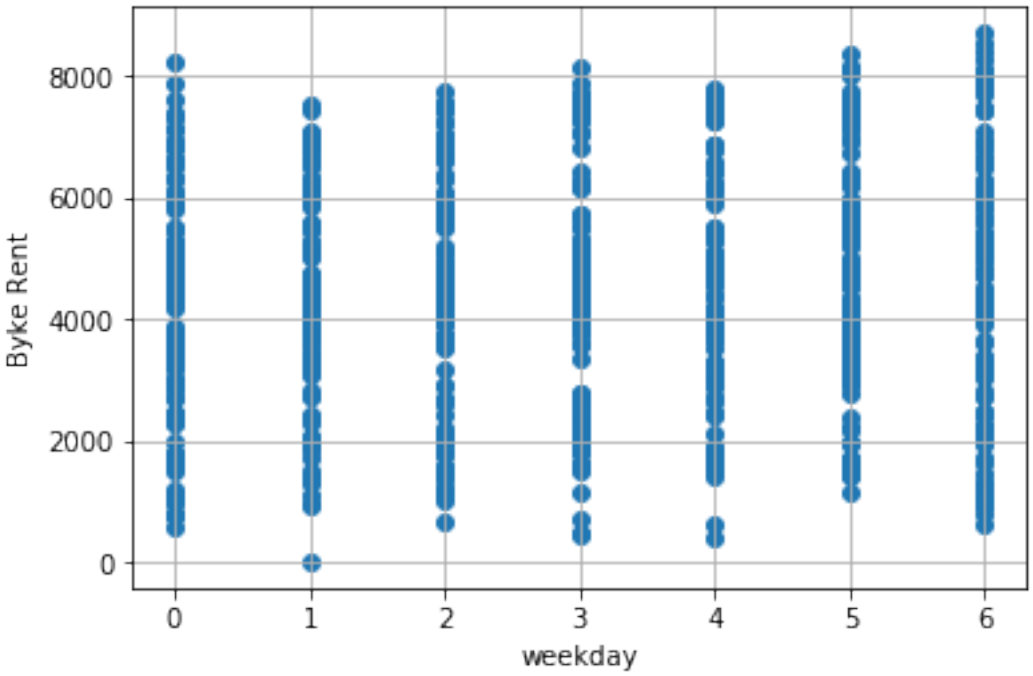
[9]:
```
cor = correlationRank(dataset.drop(["cnt"], axis=1),dataset["cnt"])
```

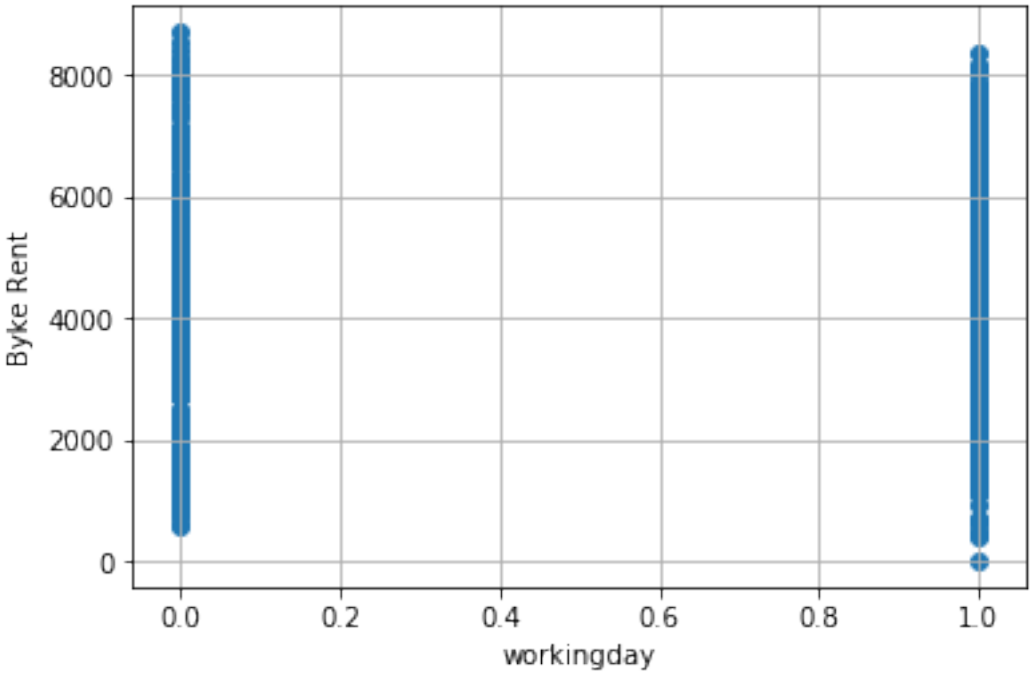


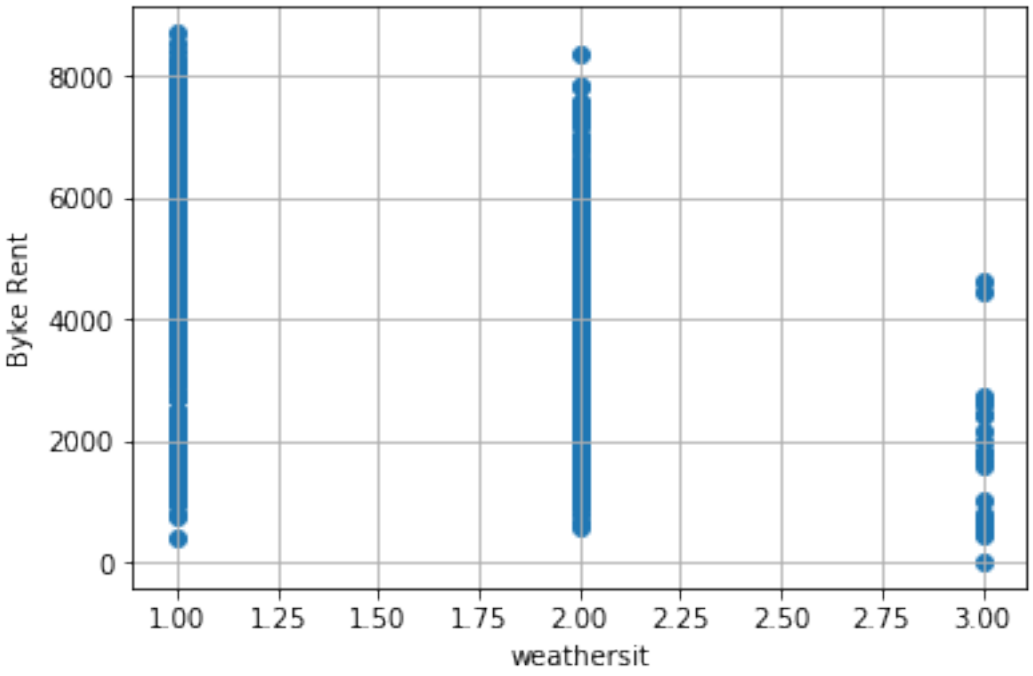

Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
yr

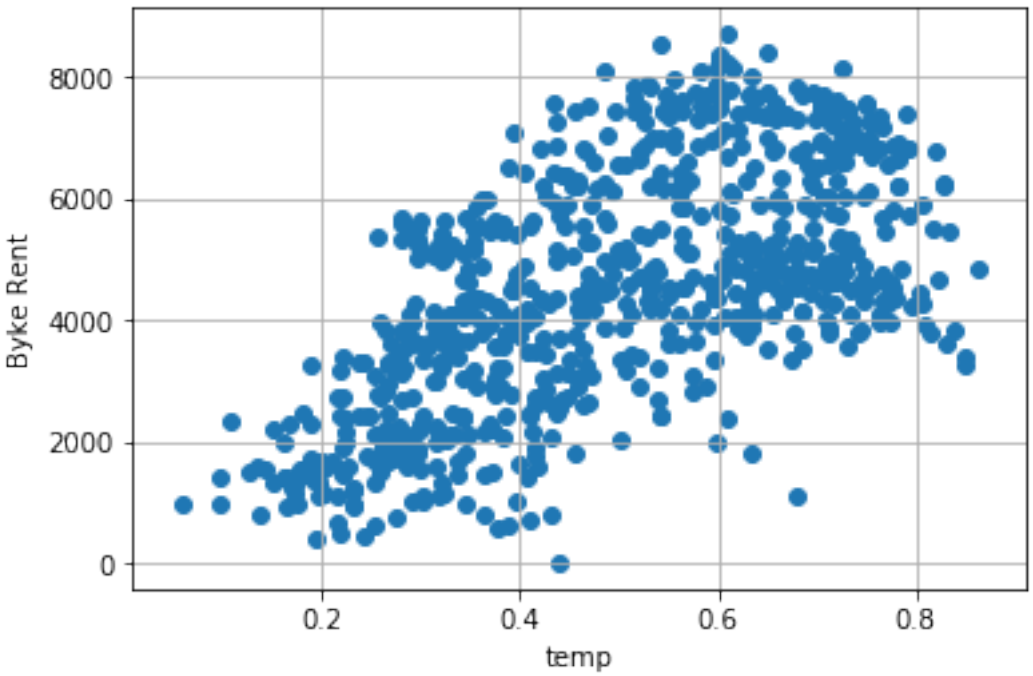

Byke Rent
8000
6000
4000
2000
0
2
4
6
8
10
12
mnth

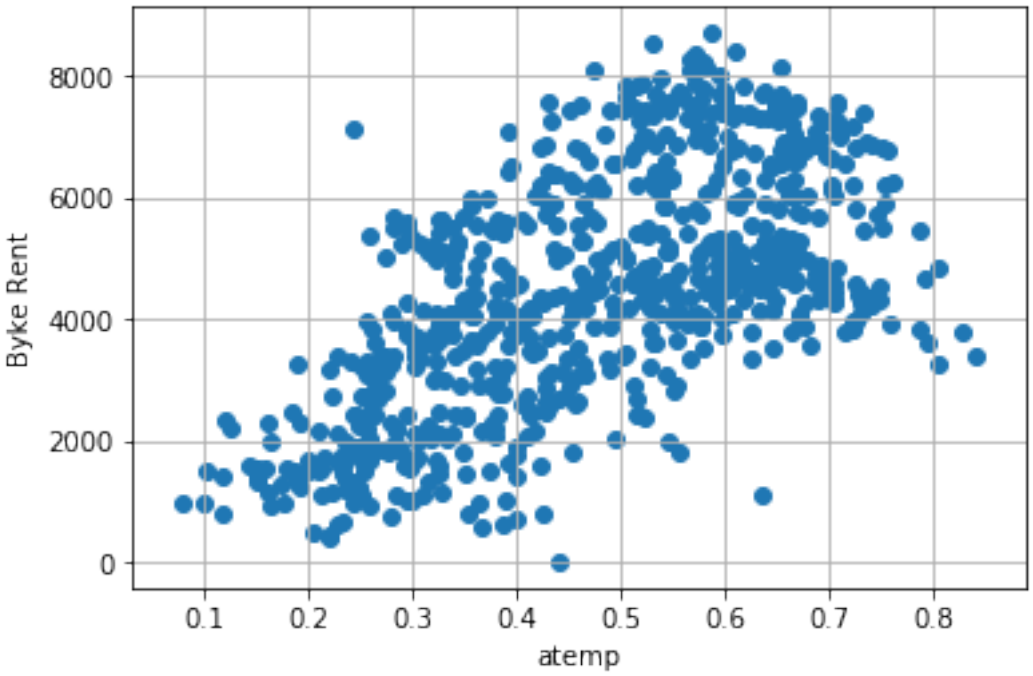

Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
holiday

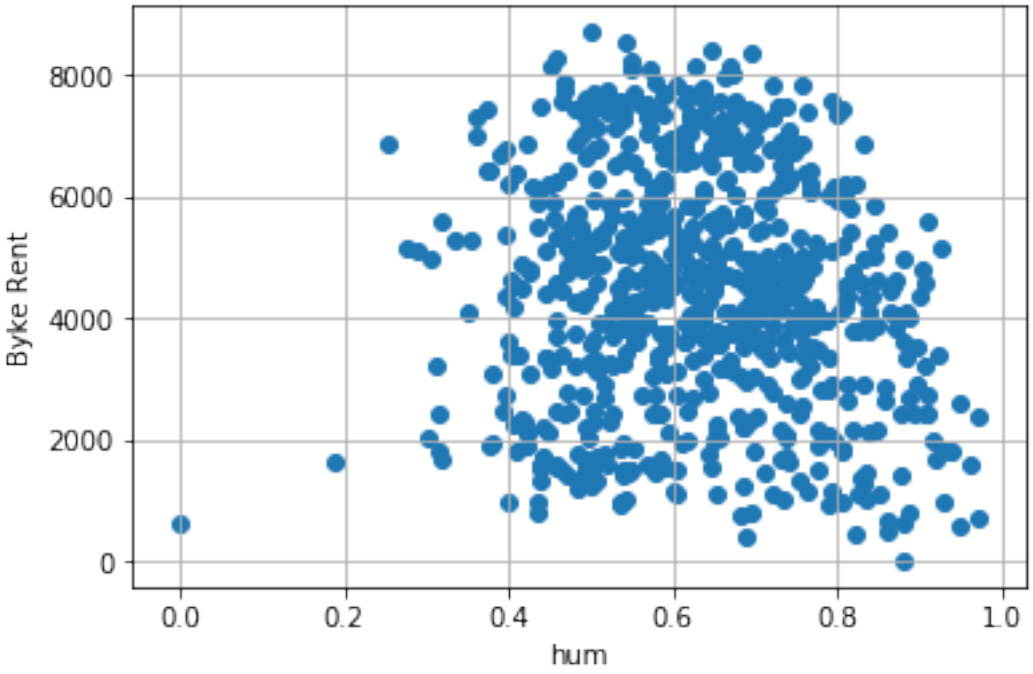

Byke Rent
8000
6000
4000
2000
0
0
1
2
3
4
5
6
weekday

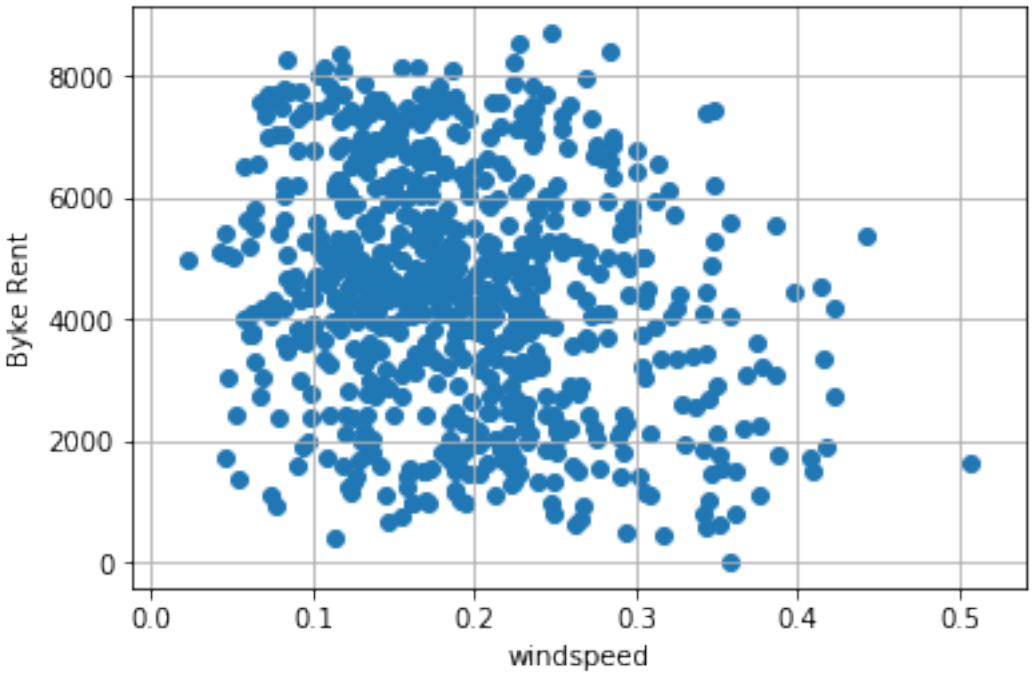

Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
workingday


Byke Rent
8000
6000
4000
2000
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
weathersit


Byke Rent
8000
6000
4000
2000
0
0.2
0.4
0.6
0.8
temp


Byke Rent
8000
6000
4000
2000
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
atemp


Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
hum


Byke Rent
8000
6000
4000
2000
0
0.0
0.1
0.2
0.3
0.4
0.5
windspeed

[10]:
```
print(cor)
```

```
atemp          0.630202
temp           0.626636
yr             0.565934
season         0.405545
mnth           0.279594
weekday        0.067351
workingday     0.061072
holiday       -0.068254
hum           -0.100521
windspeed     -0.234224
weathersit    -0.296984
dtype: float64
```

Dai grafici e dal calcolo della correlazione scopriamo che gli attributi weekday workingday e holiday sono attributi poco importanti per il calcolo. Nonostante ciò essendo la correlazione un calcolo su un coefficente di primo grado questi attributi poco correlati non vengono esclusi dal calcolo in quanto questo si baserà verosimilmente su un algoritmo polinomiale.

1.3 Preaparazione dei dati

Molti dei dati sono già stati standardizzati alla creazione del dataset, i dati che normalmente vengono presentati come categorici sono già forniti in forma numerica

I dati che necessitano di standardidazione verranno elaborati successivamente in ogni Pipeline attraverso la funzione di sklearn *StandardScaler*

Essendo questo un problema di regressione calcoliamo con la norma L1, le feature più rilevanti, ma prima dividiamo il dataset in trainSet e ValidationSet.

- Training set: come si evince dal nome stesso, verrà utilizzato per allenare l'algoritmo di learning
- Validation set: modello utilizzato per verificare l'accuratezza del modello generato.

Spesso è utilizzato un terzo insieme dei dati, definito Test set. Qual'ora i risulati del validation set siano soddisfacenti, è consigliabile testare il modello su valori diversi e non "visti" in precedenza. Ciò permetterà di misurare l'accuratezza a pieno regime.

[11]:
```
from sklearn.model_selection import train_test_split
Y = dataset["cnt"]
X = dataset.drop(["cnt"], axis=1)
XTrain, XVal, YTrain, YVal = train_test_split(X, Y, test_size=0.33,␣
 ↪random_state=73)
```

Il modello Lasso è un modello di regressione lineare che aggiunge la norma L1$ $|\theta|$

Generiamo adesso diversi modelli di learning utilizzando k (nested) cross fold validation applicata ad una grid search.

Il metodo di learning di riferimento, come già detto, sarà la regressione, la quale permette di stimare una data variabile Y in base alla relazione con le altre variabili $X = \{x_1, ..., x_n\}$ con $n > 0$. Il tutto avviene attraverso la discesa del gradiente che, variando i coefficienti θ delle variabili X, permete la diminuzione progressiva dell'errore calcolato tra la funzione di stima e il caso reale.

Per ogni classe di modello costruiamo ora le Pipeline che ci permetteranno di addestrarlo

Il modello Ridge è un modello di regressione lineare che aggiunge la norma L2$ $\|\theta\|$

Valutiamo ora i modelli ricavati nel punto precedente attraverso le metriche gia introdotte di R^2, errore relativo e errore quadratico medio. aggiungiamo alla valutazione una tabella ottenuta attraverso l'attributo *cv_results* di *GridSearchCV* in modo da verificare quali parametri hanno portato un risultato migliore nei vari tipi di regressione.

[25]:
```
def EvalutationTable(results):
    return pd.DataFrame(results.cv_results_).sort_values("mean_test_score",␣
↪ascending=False)
```

[26]:
```
printEvalutation(XVal, YVal, lassoGridSearch)
EvalutationTable(lassoGridSearch)
```

```
Mean squared error    : 4.375e+05
Relative error        : 14.49989%
R-squared coefficient : 0.89501
```

[26]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_linreg__alpha	param_poly__degree	params	split0_test_score
7	2.026778	0.385686	0.242197	0.001691	8	6	{'linreg__alpha': 8, 'poly__degree': 6}	0.853653
4	3.376660	0.478572	0.252330	0.008810	5	6	{'linreg__alpha': 5, 'poly__degree': 6}	0.850507
8	8.837437	0.723542	1.665664	0.110541	8	8	{'linreg__alpha': 8, 'poly__degree': 8}	0.853018
5	13.761474	0.822179	1.509044	0.018292	5	8	{'linreg__alpha': 5, 'poly__degree': 8}	0.845666
1	11.725694	3.374477	0.246336	0.006360	1	6	{'linreg__alpha': 1, 'poly__degree': 6}	0.789166
6	0.006316	0.002048	0.002991	0.001409	8	1	{'linreg__alpha': 8, 'poly__degree': 1}	0.759610
0	0.015626	0.002047	0.005316	0.001693	1	1	{'linreg__alpha': 1, 'poly__degree': 1}	0.756518
3	0.006983	0.002156	0.003330	0.001889	5	1	{'linreg__alpha': 5, 'poly__degree': 1}	0.758797
2	39.967745	10.855220	1.534200	0.011069	1	8	{'linreg__alpha': 1, 'poly__degree': 8}	0.845666

```
2  {'linreg__alpha': 1, 'poly__degree': 8}           0.765142

   split1_test_score  split2_test_score  mean_test_score  std_test_score  \
7           0.912994           0.830353         0.865667        0.034791
4           0.907317           0.829527         0.862450        0.032862
8           0.873217           0.831365         0.852533        0.017089
5           0.834053           0.822341         0.834020        0.009523
1           0.751333           0.759454         0.766651        0.016262
6           0.800192           0.693349         0.751050        0.044036
0           0.802837           0.692739         0.750698        0.045135
3           0.800728           0.692468         0.750664        0.044570
2           0.619144           0.744135         0.709474        0.064446

   rank_test_score
7                1
4                2
8                3
5                4
1                5
6                6
0                7
3                8
2                9
```

[ ]:
```
printEvalutation(XVal, YVal, ridgeGridSearch)
EvalutationTable(ridgeGridSearch)
```

[ ]:
```
printEvalutation(XVal, YVal, NRGridSearch)
EvalutationTable(NRGridSearch)
```

[29]:
```
printEvalutation(XVal, YVal, ENgridSearch)
EvalutationTable(ENgridSearch)
```

```
Mean squared error    : 5.6638e+05
Relative error        : 15.86626%
R-squared coefficient : 0.86408
```

[29]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
14       7.257461      5.981885         0.242590        0.003715
26       0.641817      0.021716         0.239435        0.004211
5        2.204465      1.649292         0.246253        0.001899
2       23.274296     27.598039         0.273652        0.025203
17       0.633198      0.011879         0.266045        0.035211
8        0.649112      0.003907         0.260462        0.022614
11      58.757980      2.736113         0.259638        0.003084
23       7.373223      2.025139         0.240346        0.003738
25       0.008989      0.001641         0.003302        0.000438
20      42.217569      3.074969         0.239356        0.001411
24       0.005983      0.002820         0.002662        0.001695
```

```
15       0.004998      0.000805         0.003647        0.001878
6        0.005673      0.001243         0.002325        0.001237
4        0.010097      0.000982         0.006193        0.000843
1        0.011637      0.002349         0.005987        0.000815
13       0.007303      0.000483         0.005999        0.000822
16       0.014261      0.005285         0.005672        0.002054
10       0.014968      0.009906         0.005977        0.001627
7        0.018190      0.006374         0.006331        0.001245
3        0.005301      0.000444         0.003663        0.001250
22       0.007641      0.001692         0.002990        0.001409
0        0.015469      0.002132         0.004655        0.002487
12       0.006650      0.002050         0.002659        0.000940
19       0.009320      0.001673         0.002833        0.000253
9        0.007979      0.002821         0.001990        0.000007
21       0.006822      0.002250         0.002327        0.001243
18       0.006002      0.002830         0.002646        0.001701

   param_linreg__alpha param_linreg__l1_ratio param_poly__degree  \
14                   2                     0.5                  6
26                   8                       1                  6
5                    1                     0.5                  6
2                    1                     0.1                  6
17                   2                       1                  6
8                    1                       1                  6
11                   2                     0.1                  6
23                   8                     0.5                  6
25                   8                       1                  2
20                   8                     0.1                  6
24                   8                       1                  1
15                   2                       1                  1
6                    1                       1                  1
4                    1                     0.5                  2
1                    1                     0.1                  2
13                   2                     0.5                  2
16                   2                       1                  2
10                   2                     0.1                  2
7                    1                       1                  2
3                    1                     0.5                  1
22                   8                     0.5                  2
0                    1                     0.1                  1
12                   2                     0.5                  1
19                   8                     0.1                  2
9                    2                     0.1                  1
21                   8                     0.5                  1
18                   8                     0.1                  1

                                                    params  split0_test_score  \
```

```
14  {'linreg__alpha': 2, 'linreg__l1_ratio': 0.5, ...          0.831632
26  {'linreg__alpha': 8, 'linreg__l1_ratio': 1.0, ...          0.835492
5   {'linreg__alpha': 1, 'linreg__l1_ratio': 0.5, ...          0.824544
2   {'linreg__alpha': 1, 'linreg__l1_ratio': 0.1, ...          0.822241
17  {'linreg__alpha': 2, 'linreg__l1_ratio': 1.0, ...          0.815963
8   {'linreg__alpha': 1, 'linreg__l1_ratio': 1.0, ...          0.821826
11  {'linreg__alpha': 2, 'linreg__l1_ratio': 0.1, ...          0.783801
23  {'linreg__alpha': 8, 'linreg__l1_ratio': 0.5, ...          0.789340
25  {'linreg__alpha': 8, 'linreg__l1_ratio': 1.0, ...          0.791523
20  {'linreg__alpha': 8, 'linreg__l1_ratio': 0.1, ...          0.764985
24  {'linreg__alpha': 8, 'linreg__l1_ratio': 1.0, ...          0.759374
15  {'linreg__alpha': 2, 'linreg__l1_ratio': 1.0, ...          0.757909
6   {'linreg__alpha': 1, 'linreg__l1_ratio': 1.0, ...          0.757316
4   {'linreg__alpha': 1, 'linreg__l1_ratio': 0.5, ...          0.762480
1   {'linreg__alpha': 1, 'linreg__l1_ratio': 0.1, ...          0.755575
13  {'linreg__alpha': 2, 'linreg__l1_ratio': 0.5, ...          0.754112
16  {'linreg__alpha': 2, 'linreg__l1_ratio': 1.0, ...          0.771459
10  {'linreg__alpha': 2, 'linreg__l1_ratio': 0.1, ...          0.742031
7   {'linreg__alpha': 1, 'linreg__l1_ratio': 1.0, ...          0.758047
3   {'linreg__alpha': 1, 'linreg__l1_ratio': 0.5, ...          0.734897
22  {'linreg__alpha': 8, 'linreg__l1_ratio': 0.5, ...          0.699377
0   {'linreg__alpha': 1, 'linreg__l1_ratio': 0.1, ...          0.682686
12  {'linreg__alpha': 2, 'linreg__l1_ratio': 0.5, ...          0.669059
19  {'linreg__alpha': 8, 'linreg__l1_ratio': 0.1, ...          0.634090
9   {'linreg__alpha': 2, 'linreg__l1_ratio': 0.1, ...          0.573319
21  {'linreg__alpha': 8, 'linreg__l1_ratio': 0.5, ...          0.401212
18  {'linreg__alpha': 8, 'linreg__l1_ratio': 0.1, ...          0.278843

    split1_test_score  split2_test_score  mean_test_score  std_test_score  \
14           0.901721           0.803112         0.845488        0.041432
26           0.913035           0.783830         0.844119        0.053099
5            0.908612           0.776951         0.836703        0.054434
2            0.901453           0.769070         0.830921        0.054392
17           0.899629           0.755812         0.823801        0.058974
8            0.892369           0.753376         0.822523        0.056746
11           0.889907           0.781267         0.818325        0.050627
23           0.863721           0.740863         0.797975        0.050527
25           0.846855           0.705940         0.781439        0.057969
20           0.837847           0.709756         0.770863        0.052458
24           0.799857           0.694766         0.751332        0.043278
15           0.801139           0.692804         0.750617        0.044527
6            0.802127           0.691817         0.750420        0.045297
4            0.799522           0.687821         0.749941        0.046456
1            0.784114           0.681905         0.740531        0.043061
13           0.780266           0.680744         0.738374        0.042126
16           0.864472           0.571240         0.735724        0.122349
10           0.755659           0.672322         0.723338        0.036500
```

```
7            0.866946          0.529924          0.718306          0.140429
3            0.719898          0.658703          0.704499          0.032956
22           0.690472          0.641400          0.677083          0.025492
0            0.651893          0.615933          0.650171          0.027279
12           0.635776          0.604826          0.636554          0.026229
19           0.610714          0.588004          0.610936          0.018815
9            0.531653          0.524333          0.543102          0.021575
21           0.361374          0.374033          0.378873          0.016620
18           0.247209          0.263030          0.263028          0.012915

    rank_test_score
14                1
26                2
5                 3
2                 4
17                5
8                 6
11                7
23                8
25                9
20               10
24               11
15               12
6                13
4                14
1                15
13               16
16               17
10               18
7                19
3                20
22               21
0                22
12               23
19               24
9                25
21               26
18               27
```

definiamo come modelli migliori i seguenti: * regressione con lasso di grado 6 e con $\lambda = 8$ * regressione con lasso di grado 6 e con $\lambda = 5$ * regressione con Elastic net di grado 6 con $\lambda = 2$ e $\alpha = 0.5$

che presentano rispettivamente le seguenti metriche di giudizio

[34]:
```
LassoModel1 = Pipeline([("poly", PolynomialFeatures(degree=6,␣
 ↪include_bias=False)),
                        ("scale",  StandardScaler()),
                        ("linreg", Lasso(alpha=8, max_iter=6000, tol=0.005))])
```

```
print("LassoModel1")
LassoModel1.fit(XTrain, YTrain)
count = 0
for a in LassoModel1.named_steps["linreg"].coef_ :
    if a != 0:
        count += 1
print("number of non zero param: " + str(count))
printEvalutation(XVal, YVal, LassoModel1)

print("\nLassoModel2")
LassoModel2 = Pipeline([("poly", PolynomialFeatures(degree=6,␣
 ↪include_bias=False)),
                        ("scale",  StandardScaler()),
                        ("linreg", Lasso(alpha=5, max_iter=6000, tol=0.005))])
LassoModel2.fit(XTrain, YTrain)
count = 0
for a in LassoModel2.named_steps["linreg"].coef_ :
    if a != 0:
        count += 1
print("number of non zero param: " + str(count))

printEvalutation(XVal, YVal, LassoModel2)

print("\nENModel")
ENModel = Pipeline([("poly",   PolynomialFeatures(degree = 6,␣
 ↪include_bias=False)),
                    ("scale",  StandardScaler()),
                    ("linreg",  ElasticNet(alpha=2, l1_ratio=0.5, tol = 0.05,␣
 ↪max_iter = 6000))])
ENModel.fit(XTrain, YTrain)
count = 0
for a in ENModel.named_steps["linreg"].coef_ :
    if a != 0:
        count += 1
print("number of non zero param: " +  str(count))

printEvalutation(XVal, YVal, ENModel)
```

```
LassoModel1
number of non zero param: 167
Mean squared error    : 4.375e+05
Relative error        : 14.49989%
R-squared coefficient : 0.89501

LassoModel2
number of non zero param: 215
Mean squared error    : 4.7139e+05
```

```
Relative error        : 15.29182%
R-squared coefficient : 0.88688

ENModel
number of non zero param: 7877
Mean squared error    : 5.6638e+05
Relative error        : 15.86626%
R-squared coefficient : 0.86408
```

Dei tre modelli complessivamente molto buoni, viene scartato il modello con ElasticNet perchè nel complesso presenta risultati peggiori.

Dei due modelli che utilizzano il Lasso viene designato come migliore quello che presenta grado 6 e con $\lambda = 8$ perchè ottiene prestazioni migliori in tutte le metriche di giudizio.
Inoltre compie la propria elaborazione con un numero di coefficienti minore, ne consegue una elaborazione più efficente rispetto al carico computazionale.

Infine è importante considerare che la diminuzione del numero di coefficienti può portare al fenomeno dell'overfitting, ossia la costruzione di un modello troppo aderente al training set con un apparente aumento della qualità della previsione. Un Modello così sviluppato potrebbe non risultare descrittivo quando applicato ad un dataset ignoto.