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.

[51]:
```
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")
```

[52]:
```
len(ds.columns)
```

[52]:
```
16
```

1.1.1 Descrizione variabili

- **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 che abbiamo scelto per effettuare la predizione è la variabile "cnt" i quanto a fini di ricerca di mercato è la variabile che più interessa. escludendo dagli attributi le colonne "casual" e "registered" che di cui "cnt" è la somma.

[53]:
```
ds.set_index(["dteday"], inplace=True)
dataset = ds.drop(["casual", "registered","instant"], axis=1)
Y = dataset["cnt"]
```

1.2 Analisi esplorativa

Una analisi dei vari attibuti si può avere attraverso la funzione *describe*

[54]:
```
dataset.describe()
```

[54]:

	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

[55]:
```
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
```

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

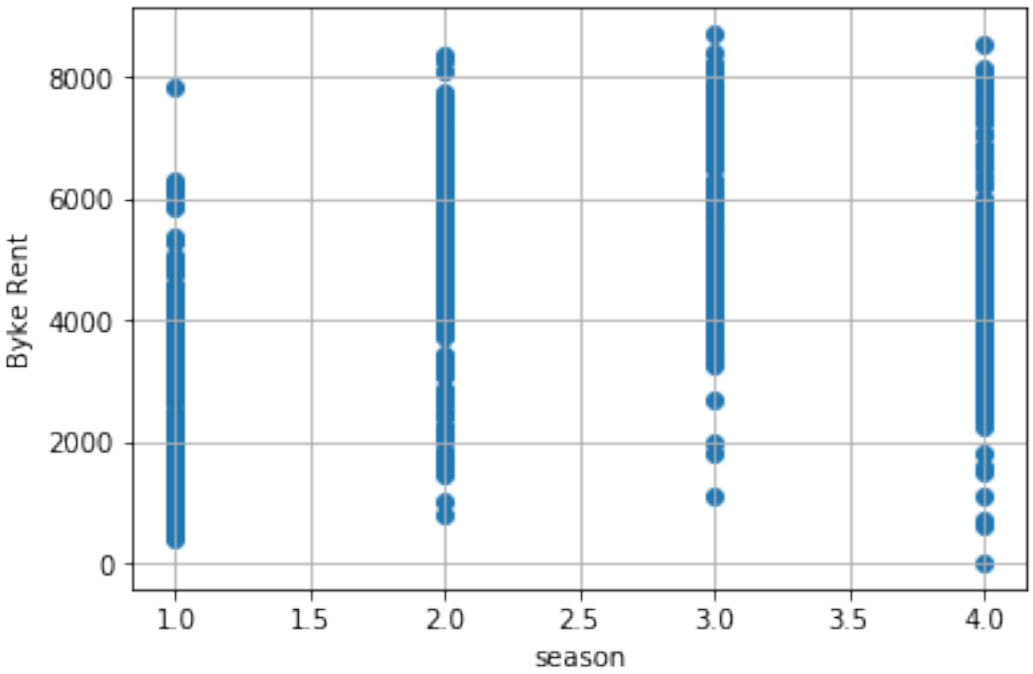
[56]:
```
import numpy as np
def getCorrelation(feature1, feature2):
    return np.mean((feature1-feature1.mean()) * (feature2-feature2.mean())) /␣
 ↪(feature1.std() * feature2.std())
```

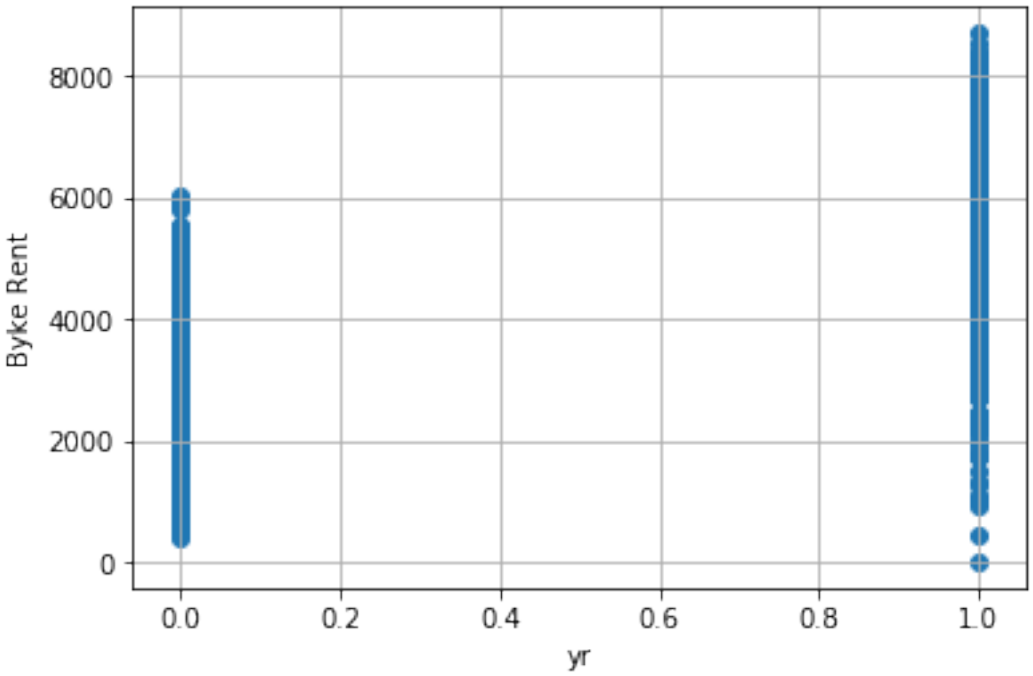
[57]:
```
import matplotlib.pyplot as plot
def plotData(x, y, XAxisName, YAxisName):
    plot.scatter(x, y)
    plot.grid()
    plot.xlabel(XAxisName); plot.ylabel(YAxisName)
    plot.show()
```

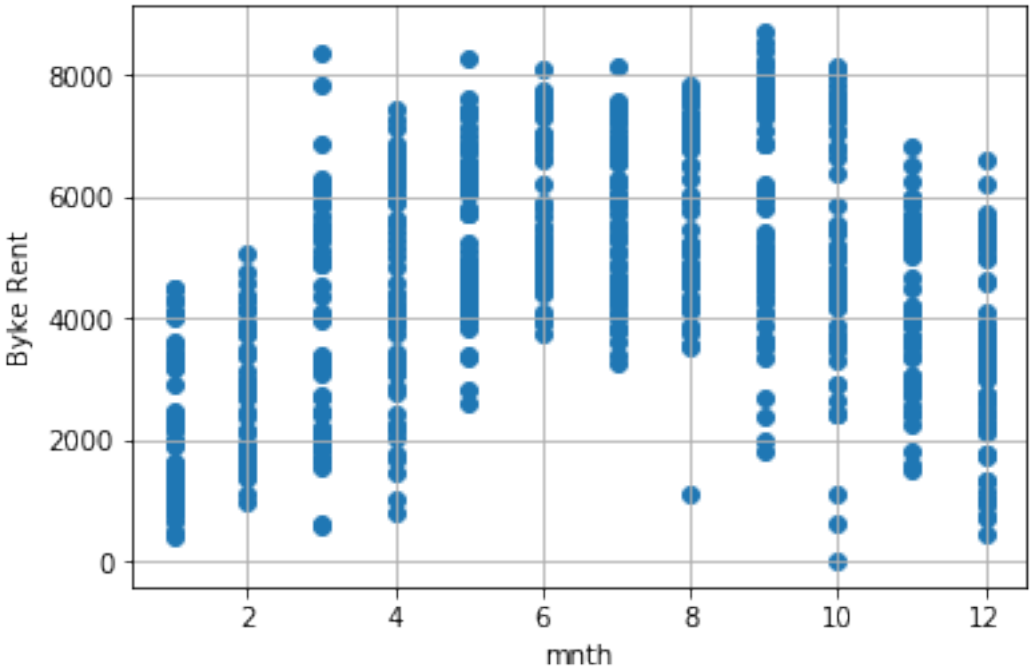
Mentre con la seguente si può ottenere una serie ordinata che indica la correlazione tra "cnt" e il nostro dataset indicizzato su "dtaday", un histogramma di ogni attributo, e un grafico a dispersione di ogni feature con l'obbiettivo

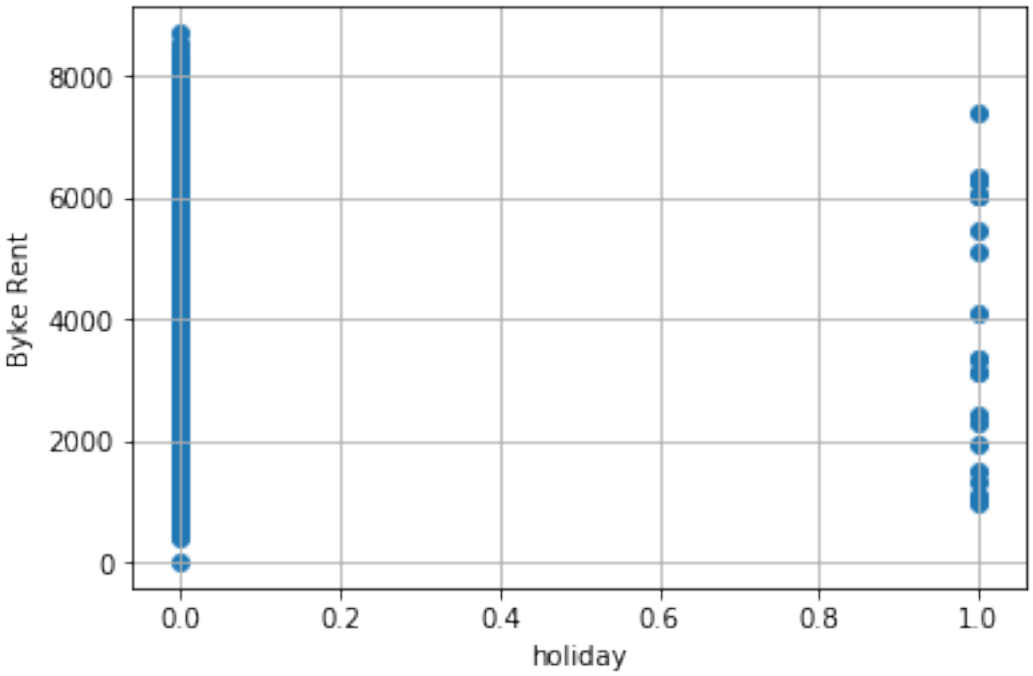
[58]:
```
def correlationRank(dataset, feature):
    correlation = []
    for a in dataset.columns:
        correlation.append(getCorrelation(dataset[a].astype("float"), feature))
        plotData(dataset[a].astype("float"), feature, a, "Byke Rent")
    cor = pd.Series(correlation, dataset.columns)
    cor.sort_values(ascending=False, inplace=True)
    return cor
```

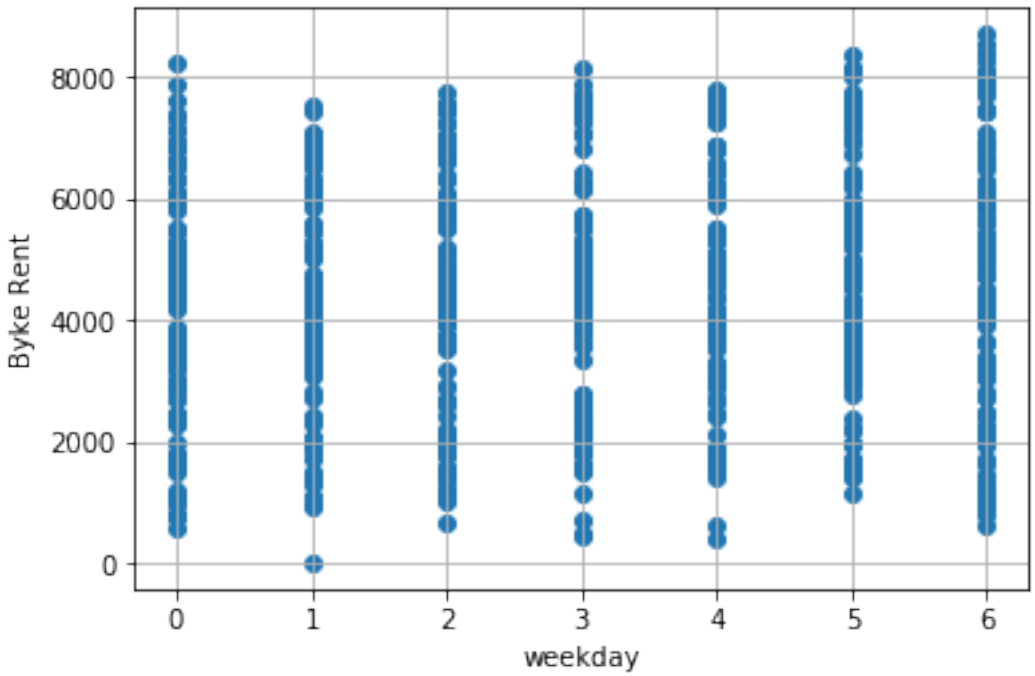
[59]:
```
cor = correlationRank(dataset.drop(["cnt"], axis=1),dataset["cnt"])
```

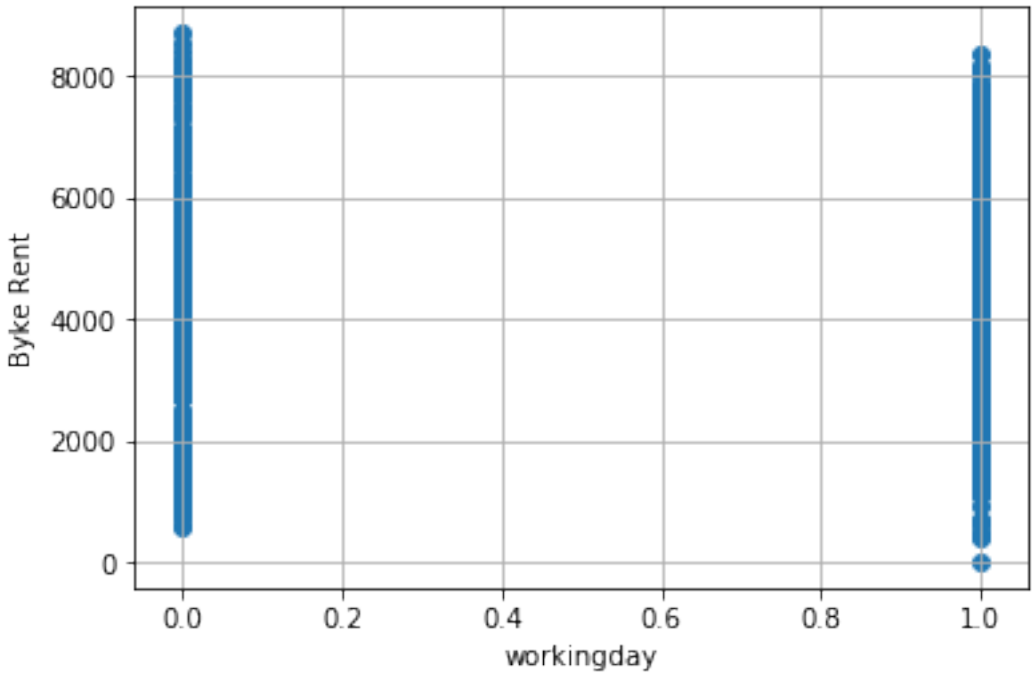

Byke Rent
8000
6000
4000
2000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
season

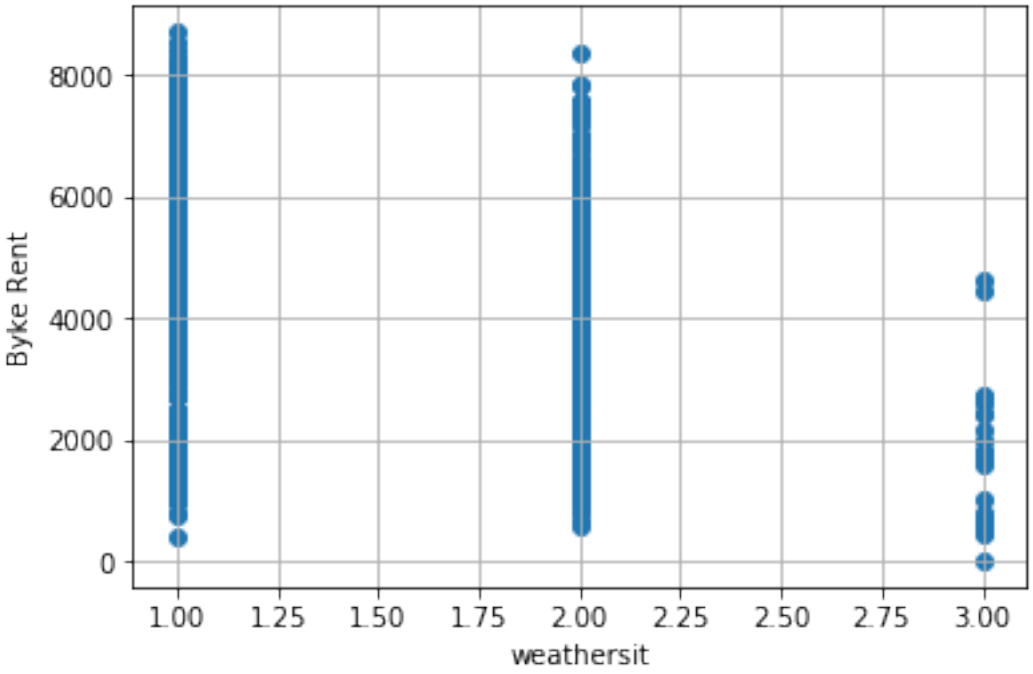

Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
yr

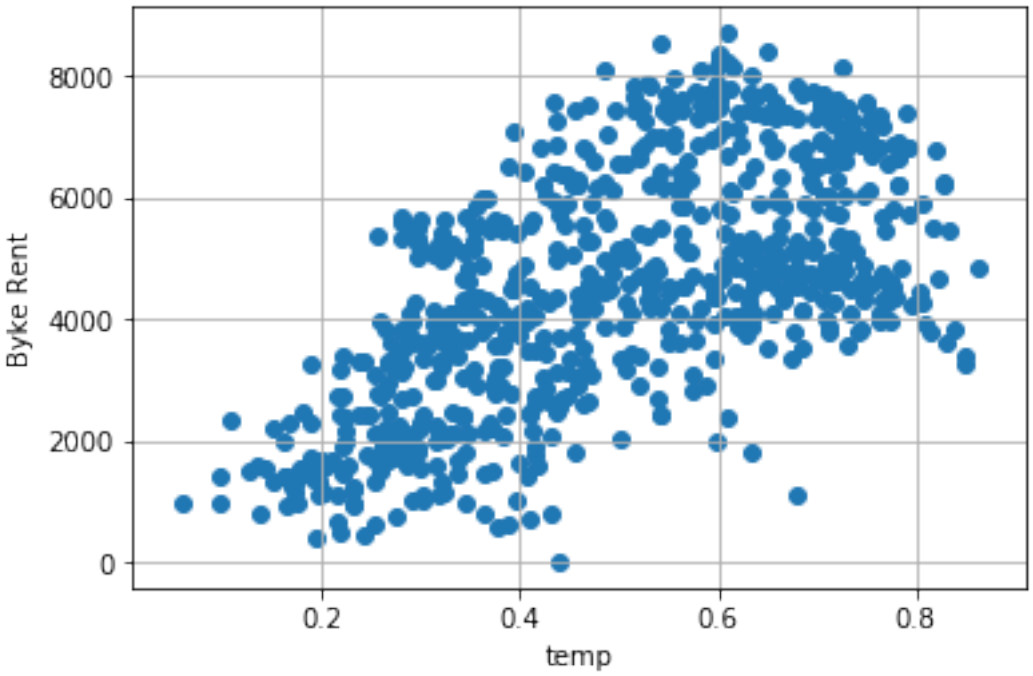

Byke Rent
8000
6000
4000
2000
0
2
4
6
8
10
12
mnth

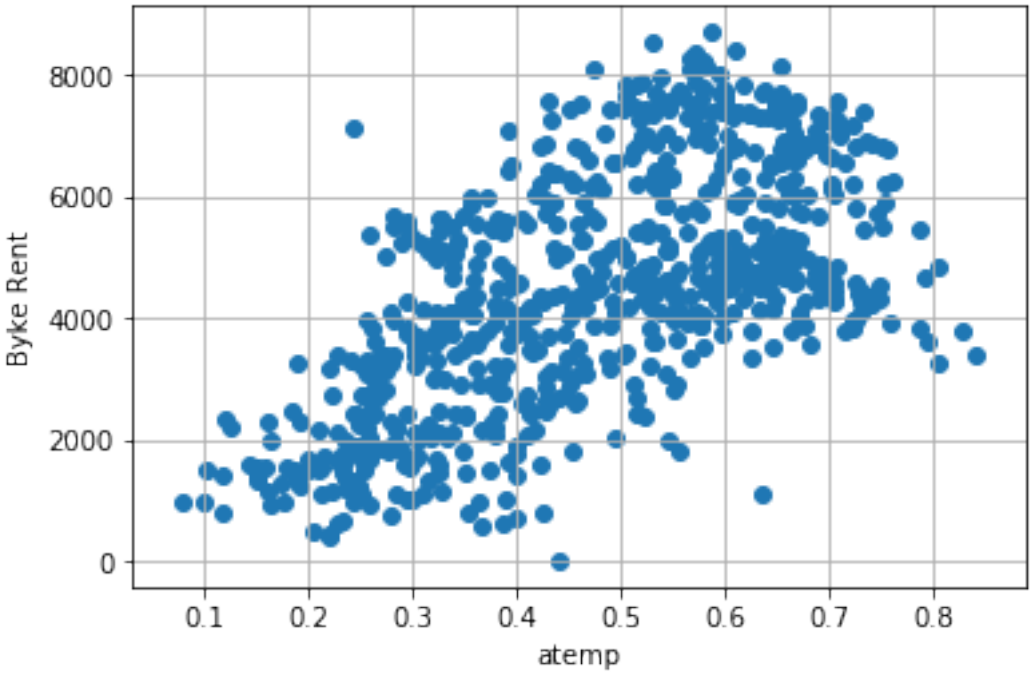

Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
holiday

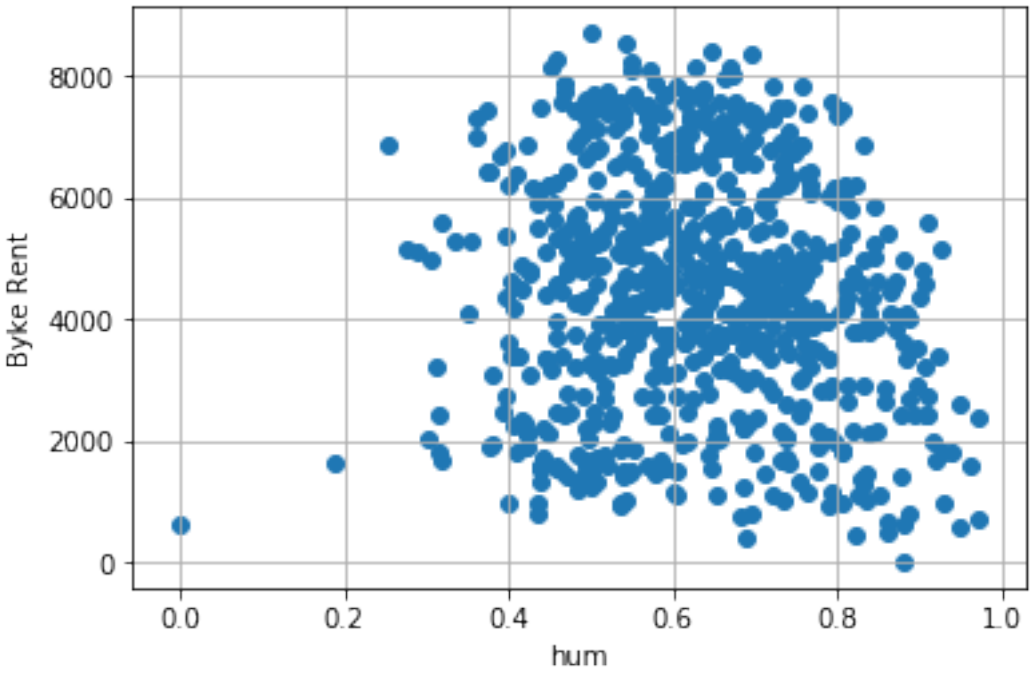

Byke Rent
8000
6000
4000
2000
0
0
1
2
3
4
5
6
weekday

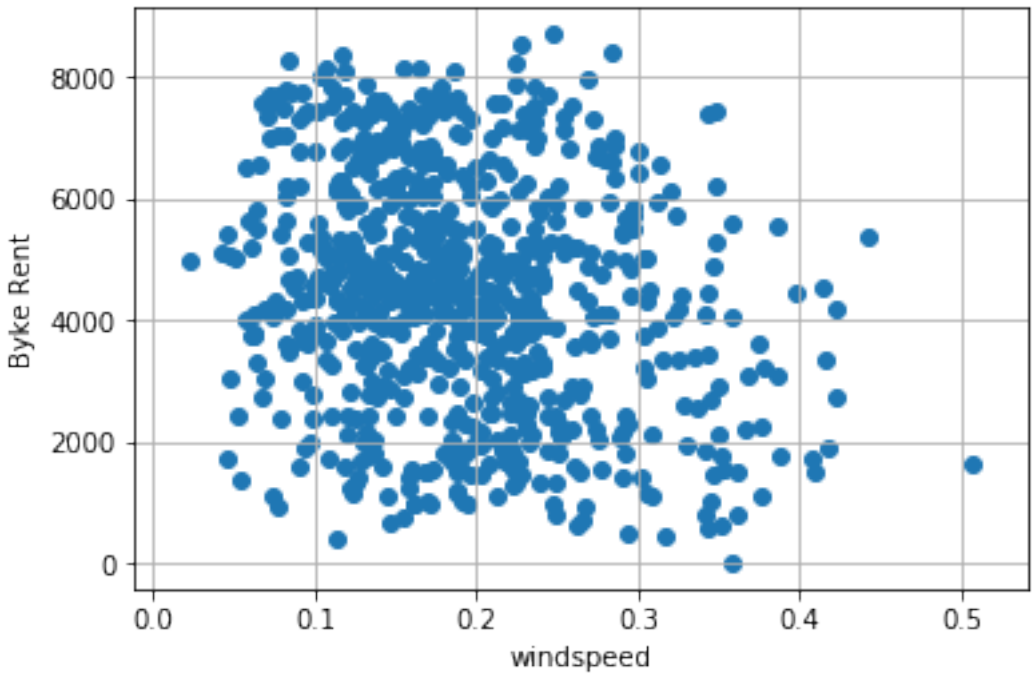

Byke Rent
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
workingday


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



[60]:
```
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

[ ]:

[61]:
```
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)
```

[62]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Lasso

def elaborationWithLasso(degeePipe=1, alphaPipe=0):
    return Pipeline([("poly", PolynomialFeatures(degree=degeePipe,␣
↪include_bias=False)),
                     ("scale",  StandardScaler()),
                     ("linreg", Lasso(alpha=alphaPipe, max_iter=6000, tol=0.
↪005))])
```

[63]:
```
def showZerosFeatures(XTrain, YTrain):
    model = elaborationWithLasso(1, 2)
    model.fit(XTrain, YTrain)
    tmp = pd.Series(model.named_steps["linreg"].coef_, XTrain.columns)
    print(tmp)
    a = []
    for row in tmp.index:
        if(tmp[row]==0):
            a.append(row)
    print(a)
```

[64]:
```
showZerosFeatures(XTrain, YTrain)
```

```
season         543.090176
yr            1029.734537
mnth          -119.037291
holiday        -79.024799
weekday        136.482490
workingday      42.310286
weathersit    -355.379160
temp           511.234902
atemp          368.312876
hum            -92.656684
windspeed     -175.260109
dtype: float64
[]
```

Prima di decidere se mantenere o no la colonna "temp" valutiamo la bontà del modello attraverso il calcolo del coefficiente R^2, dell' errore relativo e dell'errore quadratico medio.

[65]:
```
def relativeError(YTrue, YPred):
    return np.mean(np.abs((YTrue - YPred) / YTrue))
```

[37]:
```
from sklearn.metrics import mean_squared_error
def printEvalutation(X, Y, model):
    print("Mean squared error    : {:.5}".format(mean_squared_error(model.
 ↪predict(X), Y)))
    print("Relative error        : {:.5%}".format(relativeError(model.
 ↪predict(X), Y)))
    print("R-squared coefficient : {:.5}".format(model.score(X, Y)))
```

[38]:
```
model = elaborationWithLasso(6, 8)
model.fit(XTrain, YTrain)
printEvalutation(XVal, YVal, model)
#model.named_steps["linreg"].coef_
```

```
Mean squared error    : 0.056597
Relative error        : 37.47671%
R-squared coefficient : -0.026112
```

Il risultato è buono ma sono possibili miglioramenti per cui non andremo ad escludere manualmente gli attributi che la norma L1 azzera ma piuttosto riuseremo la norma successivamente.

1.4 Generazione modelli di learning

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

Definiamo innanzitutto le pipeline di ogni modello.

Il modello Ridge è un modello di regressione lineare che applica $ $\|\theta\|$

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.

[81]:
```
def EvalutationTable(results):
    return pd.DataFrame(results.cv_results_).sort_values("mean_test_score",␣
 ↪ascending=False)
```

[97]:
```
printEvalutation(XVal, YVal, lassoGridSearch)
EvalutationTable(lassoGridSearch)
```

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

[97]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
7	1.371462	0.269155	0.170868	0.002059	
4	2.392622	0.204151	0.247671	0.055406	
8	5.944362	0.641371	1.139963	0.131013	

```
5      10.428777      1.305873         1.184167        0.137482
1       7.563224      2.084867         0.175528        0.010685
6       0.003985      0.000005         0.001995        0.000814
0       0.006639      0.002357         0.003321        0.000938
3       0.005317      0.000471         0.002331        0.000469
2      25.923007      7.181931         1.166880        0.112282

  param_linreg__alpha param_poly__degree  \
7                   8                   6
4                   5                   6
8                   8                   8
5                   5                   8
1                   1                   6
6                   8                   1
0                   1                   1
3                   5                   1
2                   1                   8

                                    params  split0_test_score  \
7  {'linreg__alpha': 8, 'poly__degree': 6}           0.853653
4  {'linreg__alpha': 5, 'poly__degree': 6}           0.850507
8  {'linreg__alpha': 8, 'poly__degree': 8}           0.853018
5  {'linreg__alpha': 5, 'poly__degree': 8}           0.845666
1  {'linreg__alpha': 1, 'poly__degree': 6}           0.789166
6  {'linreg__alpha': 8, 'poly__degree': 1}           0.759610
0  {'linreg__alpha': 1, 'poly__degree': 1}           0.756518
3  {'linreg__alpha': 5, 'poly__degree': 1}           0.758797
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
```

[83]:
```
printEvalutation(XVal, YVal, ridgeGridSearch)
EvalutationTable(ridgeGridSearch)
```

```
Mean squared error    : 6.8278e+05
Relative error        : 18.41982%
R-squared coefficient : 0.83615
```

[83]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
6       0.003989  7.867412e-07         0.001662        0.000470
3       0.003668  4.797091e-04         0.001662        0.000470
0       0.013290  8.221573e-03         0.002330        0.000942
7       0.432275  2.534582e-03         0.164879        0.004098
4       0.431044  1.201211e-02         0.170195        0.010092
1       0.441285  8.501083e-03         0.175199        0.008238
8       2.713074  8.197481e-03         1.160564        0.065962
5       2.723050  4.973643e-02         1.028583        0.002616
2       2.785268  7.849298e-02         1.080777        0.039904

  param_linreg__alpha param_poly__degree  \
6                   6                  1
3                   2                  1
0                   1                  1
7                   6                  6
4                   2                  6
1                   1                  6
8                   6                  8
5                   2                  8
2                   1                  8

                                  params  split0_test_score  \
6  {'linreg__alpha': 6, 'poly__degree': 1}           0.759824
3  {'linreg__alpha': 2, 'poly__degree': 1}           0.757911
0  {'linreg__alpha': 1, 'poly__degree': 1}           0.757071
7  {'linreg__alpha': 6, 'poly__degree': 6}           0.770281
4  {'linreg__alpha': 2, 'poly__degree': 6}           0.654038
1  {'linreg__alpha': 1, 'poly__degree': 6}           0.519560
8  {'linreg__alpha': 6, 'poly__degree': 8}           0.196457
5  {'linreg__alpha': 2, 'poly__degree': 8}          -0.248420
2  {'linreg__alpha': 1, 'poly__degree': 8}          -0.620303

   split1_test_score  split2_test_score  mean_test_score  std_test_score  \
6           0.801570           0.691764         0.751053        0.045255
3           0.802726           0.692420         0.751019        0.045295
0           0.803070           0.692696         0.750946        0.045268
```

```
7          0.647910          0.716620         0.711604        0.050084
4          0.457669          0.592960         0.568222        0.082054
1          0.332098          0.480846         0.444168        0.080806
8          0.144853          0.437809         0.259706        0.127688
5          0.022533          0.192883        -0.011002        0.181715
2         -0.018037          0.008956        -0.209795        0.290482

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

[84]:
```
printEvalutation(XVal, YVal, NRGridSearch)
EvalutationTable(NRGridSearch)
```

```
Mean squared error    : 6.9471e+05
Relative error        : 26.48701%
R-squared coefficient : 0.83329
```

[84]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.007647      0.000470         0.001995        0.000002
1       0.011599      0.001692         0.003359        0.000471

  param_poly__degree              params  split0_test_score  \
0                  1  {'poly__degree': 1}           0.754600
1                  2  {'poly__degree': 2}           0.731238

   split1_test_score  split2_test_score  mean_test_score  std_test_score  \
0           0.803568           0.693151         0.750439        0.045173
1           0.839222           0.335742         0.635401        0.216428

   rank_test_score
0                1
1                2
```

[85]:
```
printEvalutation(XVal, YVal, gs)
EvalutationTable(gs)
```

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

[85]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	\
14	5.194003	4.204705e+00	0.174201	1.575638e-02	
26	0.449697	1.377411e-02	0.167882	4.177777e-03	
5	1.473052	1.076288e+00	0.162898	2.049241e-03	
2	14.287216	1.666043e+01	0.165226	3.083492e-03	
17	0.436931	1.272212e-03	0.170212	1.222562e-02	
8	0.436830	6.361488e-03	0.162235	4.713707e-04	
11	37.298311	4.227240e-01	0.163563	8.141988e-04	
23	4.973026	1.385614e+00	0.162899	1.242987e-03	
25	0.005966	8.030556e-04	0.002324	4.734664e-04	
20	29.926910	2.056698e+00	0.164217	3.067716e-03	
24	0.003666	4.766547e-04	0.001994	1.877310e-06	
15	0.003988	7.867412e-07	0.001330	4.701903e-04	
6	0.004332	4.846888e-04	0.001662	4.700779e-04	
4	0.008300	4.783025e-04	0.002327	4.704151e-04	
1	0.007300	4.518705e-04	0.002670	4.778656e-04	
13	0.008635	1.699869e-03	0.003320	1.238675e-03	
16	0.010788	2.356491e-03	0.002328	4.716513e-04	
10	0.007098	8.257996e-04	0.002662	4.715951e-04	
7	0.012659	3.914850e-03	0.002328	4.695754e-04	
3	0.003999	9.218167e-06	0.001660	4.687292e-04	
22	0.005638	4.604813e-04	0.002007	1.585837e-05	
0	0.004323	4.697429e-04	0.001993	2.099648e-06	
12	0.003325	4.700217e-04	0.001995	1.946680e-07	
19	0.006139	2.083184e-04	0.002004	9.882155e-06	
9	0.004009	1.259388e-05	0.001662	4.709234e-04	
21	0.003989	2.404313e-05	0.001663	4.699093e-04	
18	0.004321	4.689539e-04	0.000998	8.485379e-07	

	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

[99]:
```
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(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(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(count)
```

```
printEvalutation(XVal, YVal, ENModel)
```

```
LassoModel1
167
Mean squared error    : 4.375e+05
Relative error        : 14.49989%
R-squared coefficient : 0.89501

LassoModel2
215
Mean squared error    : 4.7139e+05
Relative error        : 15.29182%
R-squared coefficient : 0.88688

ENModel
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.

[ ]:

[ ]: