

Linear model in WDE

```
In [1]: #WDE dataset
WDE_path="C:/Users/aliba/OneDrive/Desktop/UNIVERSITA/TESI/DATASET/WalkingDistanceEstimation-master/dataset/"
classi=['armhand', 'pocket', 'calling', 'swing', 'handheld']
n_elem=500
n_left=50

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import pandas as pd

#model
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split, GridSearchCV

#regression
from sklearn.model_selection import KFold
from sklearn import linear_model
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.metrics import make_scorer, r2_score

#visualization
from yellowbrick.regressor import PredictionError
from yellowbrick.regressor import ResidualsPlot
from yellowbrick.features import rank1d, rank2d
from yellowbrick.model_selection import FeatureImportances
from yellowbrick.model_selection import LearningCurve

from ipynb.fs.full.functioncollection import trueeqWDE,error_rate, importWDE, filtWDE, f_ext_WDE,makeeqWDE,full
```

Import all WDE

```
In [2]: DATASET, stop_list =importWDE()
```

```

PDR_Raw_2019-03-20-09-10-12 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 288}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 12}

PDR_Raw_2019-03-20-09-21-02 {'armhand': 0, 'pocket': 0, 'calling': 284, 'swing': 0, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 15, 'swing': 0, 'handheld': 0}

PDR_Raw_2019-03-20-09-29-55 {'armhand': 0, 'pocket': 0, 'calling': 34, 'swing': 0, 'handheld': 45}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 3, 'swing': 0, 'handheld': 1}

PDR_Raw_2019-03-21-08-32-39 {'armhand': 196, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 0}
Outliers eliminati          {'armhand': 26, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 0}

PDR_Raw_2019-03-21-09-07-51 {'armhand': 527, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 0}
Outliers eliminati          {'armhand': 203, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 0}

PDR_Raw_2019-03-21-11-57-56 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 197, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 151, 'handheld': 0}

PDR_Raw_2019-03-24-11-12-21 {'armhand': 0, 'pocket': 142, 'calling': 0, 'swing': 139, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 8, 'calling': 0, 'swing': 10, 'handheld': 0}

PDR_Raw_2019-03-28-11-50-11 {'armhand': 0, 'pocket': 0, 'calling': 275, 'swing': 429, 'handheld': 425}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 16, 'swing': 42, 'handheld': 121}

PDR_Raw_2019-03-29-07-37-22 {'armhand': 0, 'pocket': 0, 'calling': 171, 'swing': 0, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 64, 'swing': 0, 'handheld': 0}

PDR_Raw_2019-03-29-08-30-54 {'armhand': 0, 'pocket': 157, 'calling': 0, 'swing': 0, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 116, 'calling': 0, 'swing': 0, 'handheld': 0}

PDR_Raw_2019-03-30-11-29-16 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 897}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 167}

PDR_Raw_2019-03-31-01-23-59 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 1726}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 202}

PDR_Raw_2019-03-31-10-04-54 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 385, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 156, 'handheld': 0}

PDR_Raw_2019-03-31-10-33-25 {'armhand': 0, 'pocket': 386, 'calling': 0, 'swing': 3, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 45, 'calling': 36, 'swing': 2, 'handheld': 0}

PDR_Raw_2019-03-31-12-03-05 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 232, 'handheld': 0}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 21, 'handheld': 0}

PDR_Raw_2019-03-31-12-29-51 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 568}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 205}

PDR_Raw_2019-04-01-10-45-07 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 1214}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 29}

PDR_Raw_2019-04-02-08-44-50 {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 664}
Outliers eliminati          {'armhand': 0, 'pocket': 0, 'calling': 0, 'swing': 0, 'handheld': 60}

```

In totale=> armhand:723, pocket:685, calling:764, swing:1385, handheld:5827, -->9384 stride

```

In [3]: stop_list["swing"].insert(2,255)
print(stop_list)

{'armhand': [0, 196, 723], 'pocket': [0, 142, 299, 685], 'calling': [0, 284, 318, 593, 764], 'swing': [0, 197,
255, 336, 765, 1150, 1153, 1385], 'handheld': [0, 288, 333, 758, 1655, 3381, 3949, 5163, 5827]}

```

```

In [4]: #select only one mode
mode="swing"
for c in classi:
    if c!=mode:
        del DATASET[c]
        del stop_list[c]

```

"DATASET" è la variabile in cui importiamo il nostro dataset WDE, è un dizionario che ha come chiavi le cinque modalità, come valori ha una lista. La lista è una lista di stride, dove ogni stride è un dizionario che rappresenta le misrazioni dello stride

```

In [5]: print(type(DATASET))
for c,v in DATASET.items():
    print(c,type(v),len(v),type(v[0]),v[0].keys())

<class 'dict'>
swing <class 'list'> 1385 <class 'dict'> dict_keys(['target', 'Acc_X', 'Acc_Y', 'Acc_Z', 'Gyr_X', 'Gyr_Y', 'Gyr_Z', 'SensorTimestamp'])

```

Applichiamo il butterworth filter di primo ordine con cutoff frequency di 3Hz ad ogni stride

```

In [6]: filt=WDF(DATASET):

```

```
Filtering:##  
Done!
```

"**Feature_DS**" è un dizionario con chiavi le classi, e valori dizionari con chiavi 'feature' che contiene la list di feature dello stride e 'target' che contiene il float della lunghezza dello stride.

```
In [7]: Feature_DS=f_ext_WDE(DATASET)
```

```
Extracting swing:#####
```

```
In [8]: for k,v in Feature_DS.items():  
        print("\n",k,type(v),v.keys(),len(v['feature']),end=" ")  
        if k==mode:  
            print(len(v['feature'][0]),len(v['target']))
```

```
armhand <class 'dict'> dict_keys(['feature', 'target']) 0  
pocket <class 'dict'> dict_keys(['feature', 'target']) 0  
calling <class 'dict'> dict_keys(['feature', 'target']) 0  
swing <class 'dict'> dict_keys(['feature', 'target']) 1385 92 1385
```

```
handheld <class 'dict'> dict_keys(['feature', 'target']) 0
```

- **DS_train** è il dataset equilibrato di train prendendo i primi 500 elementi per ogni modalità
- **DS_test** è il dataset di test equilibrato prendendo gli *ultimi* 50 elementi per ogni modalità

```
In [9]: #nt=int(len(Feature_DS[mode]['target']/10)  
DS_train,DS_test = trueeqWDE(Feature_DS,stop_list,n_train=600,n_test=60)
```

```
In [10]: for k,v in DS_train.items():  
         print("\n",k,type(v),v.keys(),len(v['feature']),end=" ")  
         if k==mode:  
             print(len(v['feature'][0]),len(v['target']))
```

```
armhand <class 'dict'> dict_keys(['feature', 'target']) 0  
pocket <class 'dict'> dict_keys(['feature', 'target']) 0  
calling <class 'dict'> dict_keys(['feature', 'target']) 0  
swing <class 'dict'> dict_keys(['feature', 'target']) 600 92 600
```

```
handheld <class 'dict'> dict_keys(['feature', 'target']) 0
```

```
In [11]: for k,v in DS_test.items():  
         print("\n",k,type(v),v.keys(),len(v['feature']),end=" ")  
         if k==mode:  
             print(len(v['feature'][0]),len(v['target']))
```

```
armhand <class 'dict'> dict_keys(['feature', 'target']) 0  
pocket <class 'dict'> dict_keys(['feature', 'target']) 0  
calling <class 'dict'> dict_keys(['feature', 'target']) 0  
swing <class 'dict'> dict_keys(['feature', 'target']) 60 92 60
```

```
handheld <class 'dict'> dict_keys(['feature', 'target']) 0
```

- **regr_dataset_train** è il dataset formattato per la regressione per il train, non tiene in considerazione le modalità ed unisce tutte le feature in una lista e così tutti i target. E' un dizionario con chiavi 'feature' e 'target'
- **regr_dataset_test** è lo stesso, ma con i dati di test

```
In [12]: regr_dataset_train, regr_dataset_test = full_regr_dataset(DS_train,DS_test)
```

```
In [13]: for k,v in regr_dataset_train.items():  
         print(k,type(v),len(v),type(v[0]))  
  
feature <class 'list'> 600 <class 'list'>  
target <class 'list'> 600 <class 'float'>
```

```
In [14]: for k,v in regr_dataset_test.items():  
         print(k,type(v),len(v),type(v[0]))  
  
feature <class 'list'> 60 <class 'list'>  
target <class 'list'> 60 <class 'float'>
```

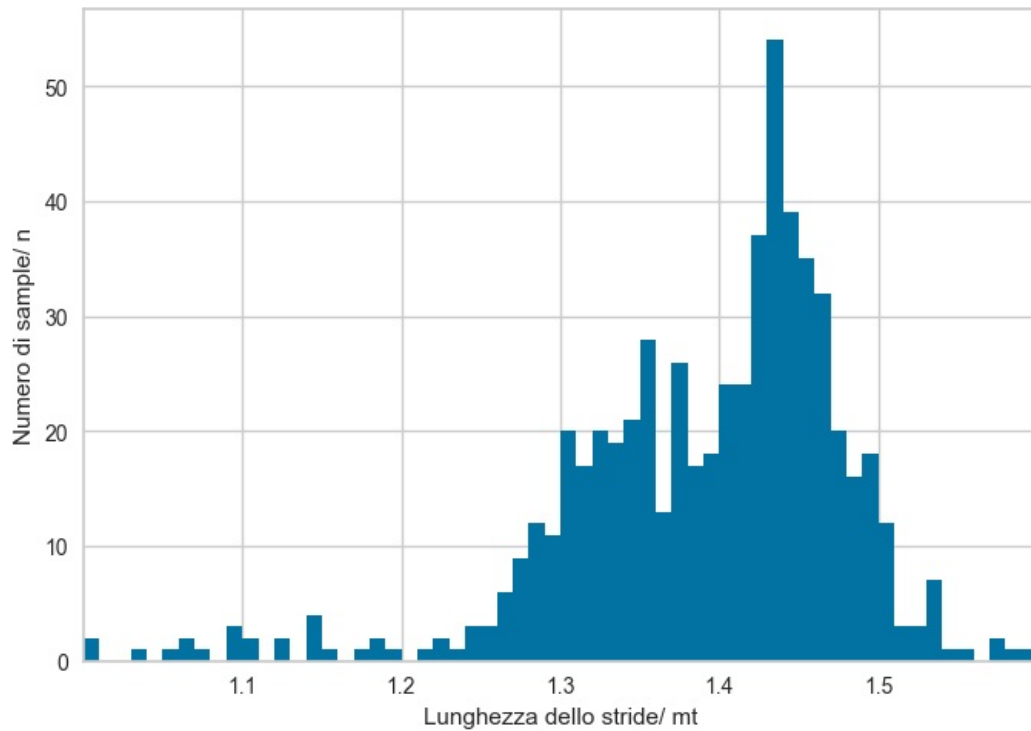
Isoliamo i singoli **F_x,F_y** rispettivamente feature e target del train e **F_x_test,F_y_test** rispettivamente feature e target di test

```
In [15]: F_x=np.array(regr_dataset_train["feature"])  
         F_y=np.array(regr_dataset_train["target"])  
         F_x_test=np.array(regr_dataset_test["feature"])  
         F_y_test=np.array(regr_dataset_test["target"])
```

Vediamo come sono distribuite le lunghezze registrate degli stride

```
In [16]: #istogramma dei target F_y  
ist_stride_lenght(F_y)
```

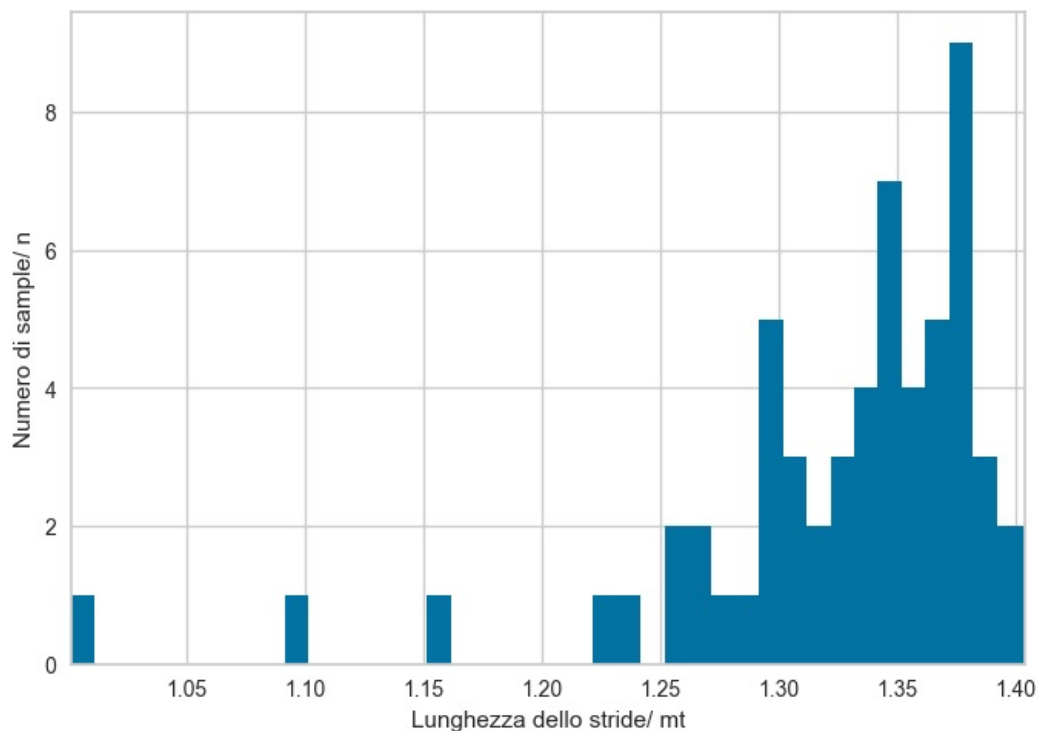
Registered Stride lenght: Min: 1.000034177353732 , Max: 1.5996465737789718
Media= 1.39142200729519; Deviazione standard= 0.08871998079193971.



Out[16]: 0

```
In [17]: #istogramma dei target di test  
ist_stride_lenght(F_y_test)
```

Registered Stride lenght: Min: 1.0016373859967411 , Max: 1.4036484769762416
Media= 1.3257855349341763; Deviazione standard= 0.07088304702057831.



Out[17]: 0

-----Feature Analysis-----

Salviamo la lista dei nomi delle feature.

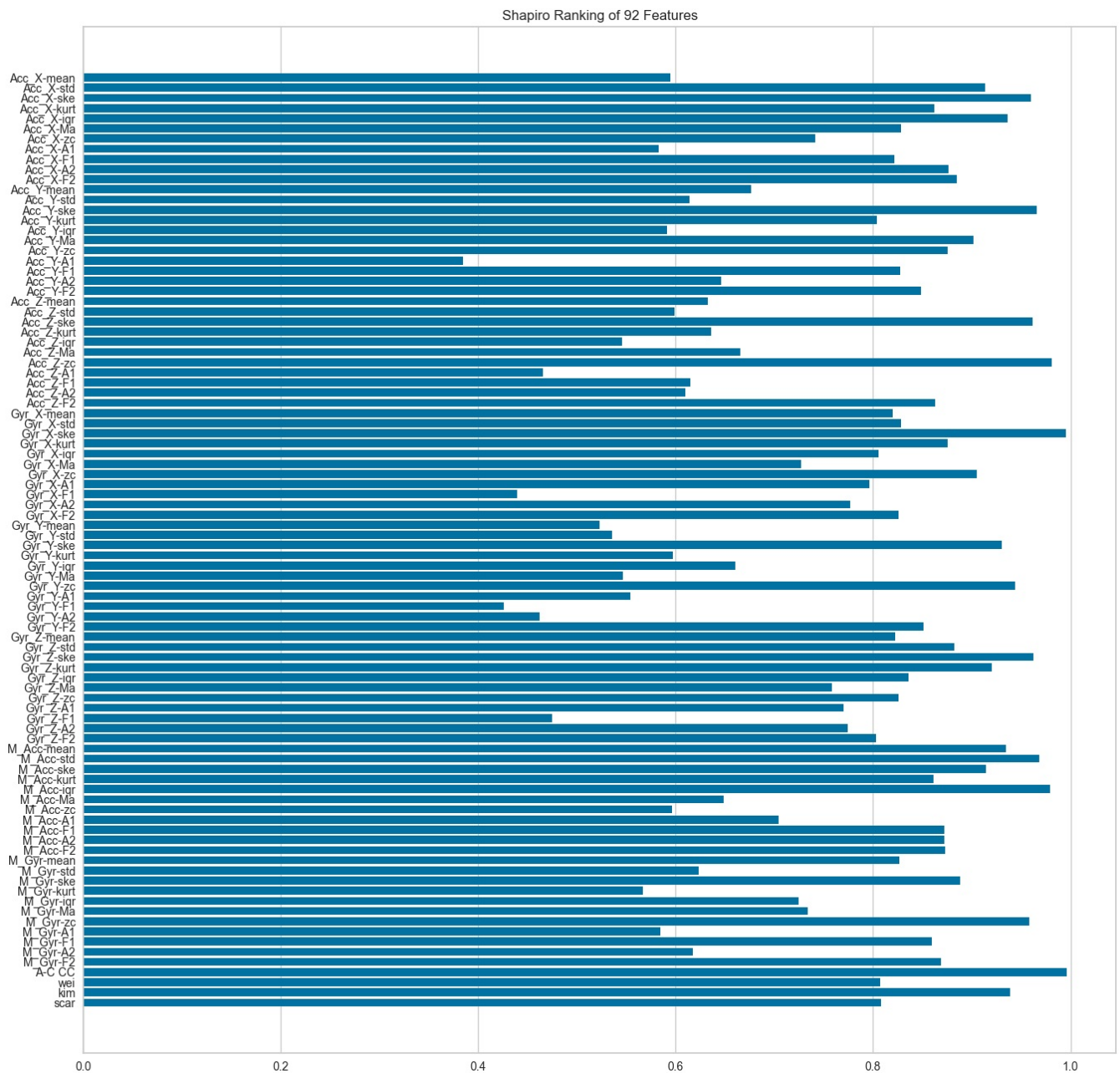
```
In [18]: feature_name=['Acc_X-mean', 'Acc_X-std', 'Acc_X-ske', 'Acc_X-kurt', 'Acc_X-igr', 'Acc_X-Ma', 'Acc_X-zc', 'Acc_X  
print(f"Abbiamo: {len(feature_name)} feature.")
```

Abbiamo: 92 feature.

rank1d e **rank2d** offrono la possibilità di visualizzare in modo semplice ed intuitivo il ranking delle feature. In particolare:

- **rank1d** utilizza shapiro da scipy.stats per calcolare lo score di ogni feature e poi plotta lo score in un barchart di matplotlib.
- **rank2d** valuta lo score per ogni coppia di feature, le funzioni di scoring supportate sono <"pearson","covariance","spearman","kendalltau"> che misurano quanto le feature sono legate tra di loro; noi utilizziamo "pearson".

```
In [19]: #rank1d
_, axes = plt.subplots(ncols=1, figsize=(16,16))
rank1d(F_x,F_y,features=feature_name,ax=axes , show=False)
plt.show()
```



```
In [20]: #####rank2d

##Per visualizzare cosa sceglierà f_regression
#devo fare in modo che tra le feature ci sia anche il target #poi vedere solo la linea riferita al target e ved

HP_x=np.hstack((F_x,np.reshape(F_y,(-1,1))))

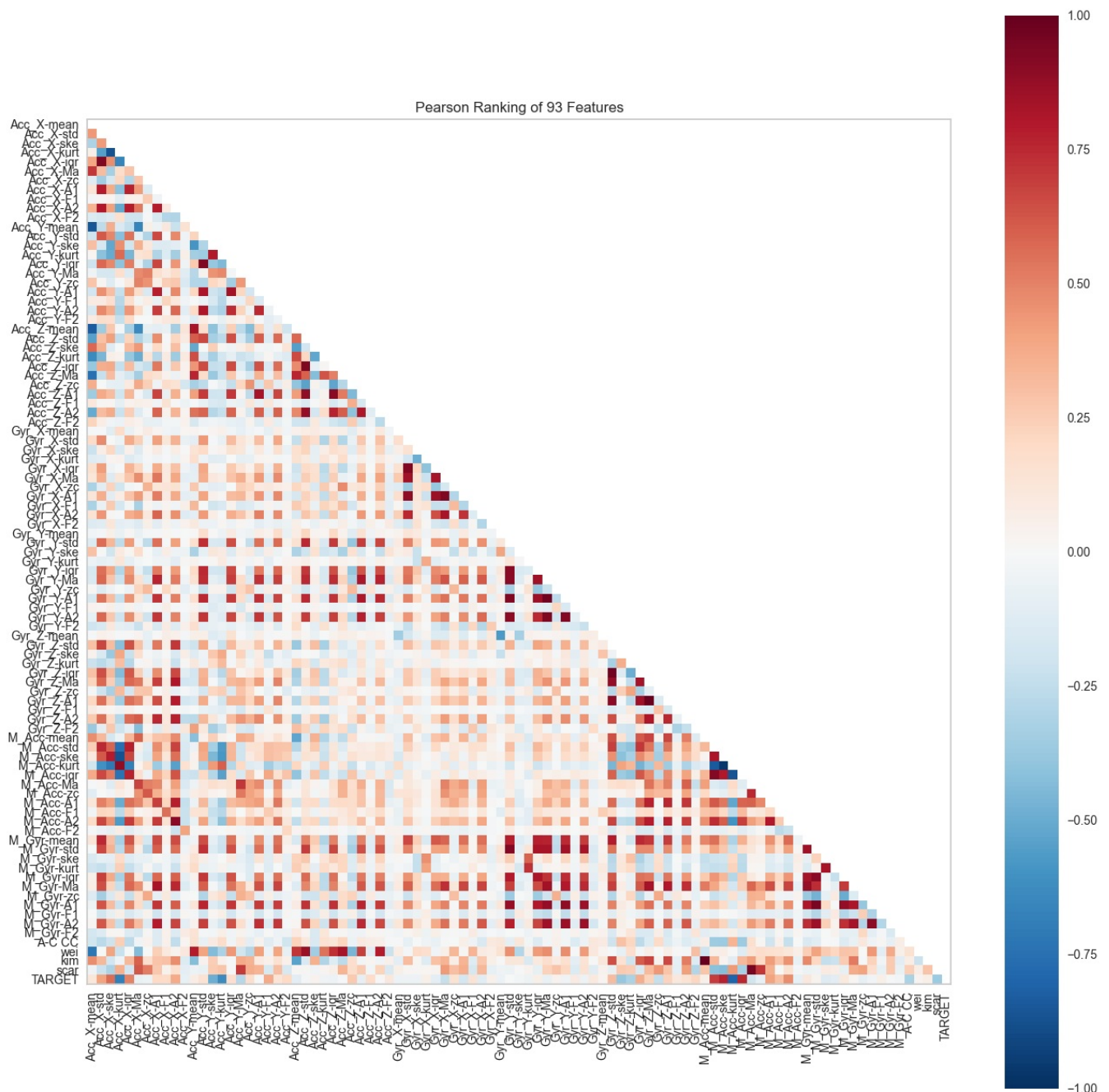
HP_featurename=feature_name+["TARGET"]

_, axes = plt.subplots(ncols=1, figsize=(16,16))

rank2d(HP_x,features=HP_featurename,ax=axes , show=False)

plt.show()

#Si vede sia ogni coppia di feature come è relazionata, sia il pearson score con il target di ogni feature!
```



Linear Regression

Descrizione:

- **input** : F_x ed F_y che sono rispettivamente il vettore di vettori di feature ed il vettore di target.
- **perf** è un oggetto scorer costruito dalla funzione che noi desideriamo usare comemetro di valutazione per le performance, nel nostro caso `error_rate`.
- **kf** è l'oggetto che crea gli indici per dividere i nostri input in k (10) fold da usare per la cross validation.
- **pipe** è l'oggetto che rappresenta la pipeline degli estimatori, ogni step fa il fit con il predict del precedente; in ordine l'input viene ridotto selezionando le feature (`SelectKbest` con `f_regression`) -> le feature vengono scalate (`StandardScaler`) -> il modello lineare viene fittato (`LinearRegression`) `f_regression` usa come metodo di ranking il pearson tra la singola feature ed il target.
- **est** è l'istanza `GridSearchCV` che permette di eseguire la cross validation tramite gli indici di `kf` (facendo quindi in tutto k (10) train usando ogni volta un fold diverso per il test) e di volta in volta testa una nuova combinazione di parametri presenti nel dizionario `param` (nel nostro caso solo quante feature selezionare, da 1 a 92). Per ogni parametro esegue una 10-fold cross-validation e calcola la media dello score calcolato con `perf`. Infine restituisce come `.best_params_` i parametri con cui si ha avuto la migliore media, con `.best_estimator_` restituisce un estimatore fittato su tutti i dati di train con il `best_param_`.

Salvo quindi il miglior estimatore in `best` che rappresenta quindi l'estimatore che ha il minor `error_rate` medio nei k fold con i migliori parametri, fittato su tutto il set di dati di train senza la divisione in fold.

```
In [21]: #F_x feature #F_y target
```

```
#Creiamo l'oggetto per valutare le performance tramite la nostra funzione
perf=make_scorer(error_rate, greater_is_better=False)
```

```
#Creiamo oggetto cross-validator tramite KFold
kf = KFold(n_splits=10, shuffle= True, random_state=0)#con lo shuffle= False assicuriamo che i blocchi siano co

#####GRIDSEARCH CV
#select = SelectKBest(score_func=f_regression)
#scaler=StandardScaler()
#regr = linear_model.LinearRegression()

#pipeline di estimatori
pipe = Pipeline([('select',SelectKBest(score_func=f_regression)),('scaler', StandardScaler()), ('regr', linear_
print("La pipeline è composta da: ", pipe)

#Parametri da testare:
param=[{'select__k':[x+1 for x in range(F_x.shape[1])]}]

#Creiamo l'oggetto per la ricerca dei parametri
est=GridSearchCV(pipe,param_grid= param, cv=kf,verbose=1,scoring="r2", return_train_score= True) #Fittiamo sul
est.fit(F_x,F_y)

#Ricaviamo il miglior set di parametri
best=est.best_estimator_
print(f"\n\nIl miglior estimatore si ha col parametro {est.best_params_} ottenuto all'indice: {est.best_index_}")
```

```
La pipeline è composta da: Pipeline(steps=[('select',
      SelectKBest(score_func=<function f_regression at 0x000001DEED66A8C0>)),
      ('scaler', StandardScaler()), ('regr', LinearRegression())])
Fitting 10 folds for each of 92 candidates, totalling 920 fits
```

Il miglior estimatore si ha col parametro {'select__k': 37} ottenuto all'indice: 36 .

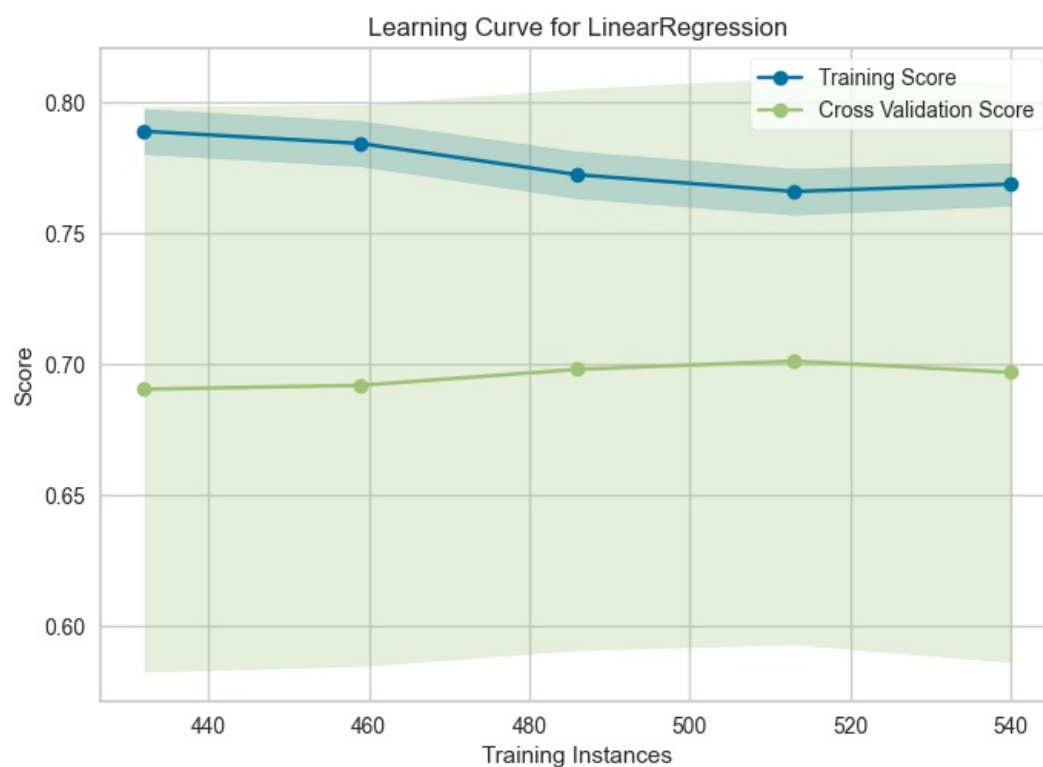
L'R2-score medio è di 0.697 !

-----MODEL EVALUATION-----

LearningCurve permette di vedere l'evoluzione delle performance del modello facendo la media dello score nella cross validation (asse y) in base al numero di train samples (asse x). Permette di valutare:

- Se il modello necessita di maggiori dati di training (se converge vuol dire che aumentare il numero di sample migliora le performance, se converge in un valore troppo alto vuol dire che necessita più sample in training)
- Se l'errore dipende di più dalla media o dalla varianza

```
In [22]: viz = LearningCurve(best, scoring="r2",cv=kf,train_sizes=np.linspace(0.8,1.0,5))
viz.fit(F_x,F_y) # Fit the data to the visualizer
viz.show()
```



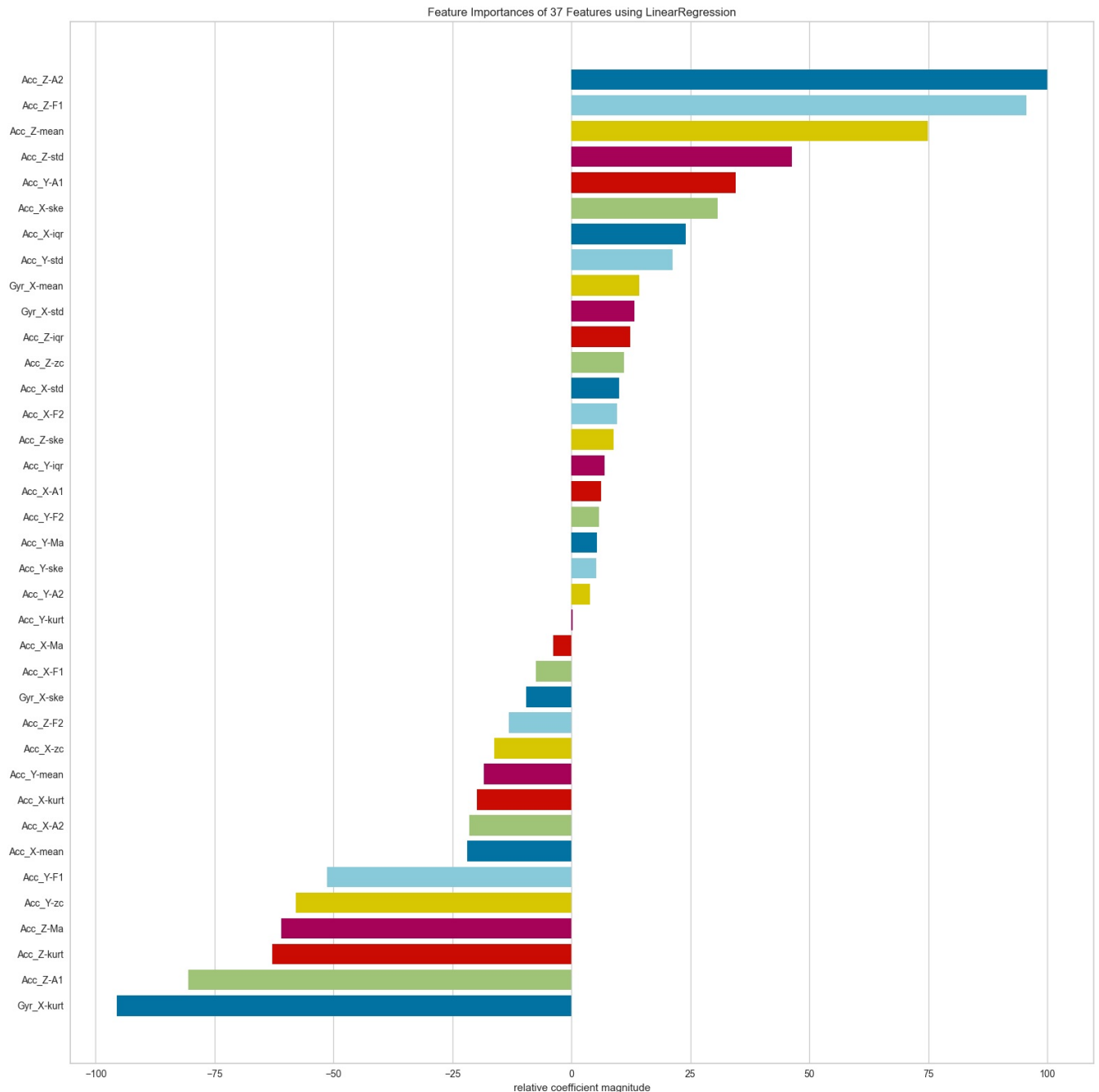
```
Out[22]: <AxesSubplot:title={'center':'Learning Curve for LinearRegression'}, xlabel='Training Instances', ylabel='Score'>
```

-----FEATURE IMPORTANCE-----

FeatureImportances permette di plottare le feature del best estimator ordinate in ordine decrescente secondo la loro importanza!

```
In [23]: #per la feature importance dobbiamo scomporre la pipeline, ci serve solo l'ultimo step, il regressore nominato
##Attenzione FeatureImportance.fit trasforma il regressore, quindi creiamone uno nuovo
kk=est.best_params_['select_k']
skb=SelectKBest(score_func=f_regression, k=kk)
temp_X=skb.fit_transform(F_x,F_y)
stdsc=StandardScaler()
temp_X=stdsc.fit_transform(temp_X)
linmod=linear_model.LinearRegression()

_, axes = plt.subplots(ncols=1, figsize=(16,16))
viz = FeatureImportances(linmod,ax=axes,labels=feature_name)
viz.fit(temp_X,F_y)
viz.show()
```



```
Out[23]: <AxesSubplot:title={'center':'Feature Importances of 37 Features using LinearRegression'}, xlabel='relative coefficient magnitude'>
```

Visualizziamo tutti i risultati del GridSearchCV

Qui possiamo vedere ogni split con ogni parametro che score ha ottenuto, sia per il train (9 fold) che per il test (1 fold). Es: split3 è la cross validation usando come train fold (0..2,4..10) e come test fold 3 .

Le righe corrispondono alle performance dell'estimatore con un determinato parametro (92 righe come il numero di parametri testati); le colonne rappresentano gli il determinato score per ogni parametro.

```
In [24]: risultati=pd.DataFrame(est.cv_results_)
display(risultati)
```


	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_select_k	params	split0_test_score	split1_test_score	split2_test_score
0	0.004216	0.005859	0.000798	0.002395	1	{'select__k': 1}	0.558988	0.709693	0
1	0.001251	0.003150	0.001601	0.004802	2	{'select__k': 2}	0.559031	0.709429	0
2	0.004687	0.007160	0.000000	0.000000	3	{'select__k': 3}	0.589979	0.668282	0
3	0.004173	0.005046	0.000801	0.002404	4	{'select__k': 4}	0.615971	0.657130	0
4	0.003210	0.003931	0.000800	0.002399	5	{'select__k': 5}	0.616112	0.649893	0
...
87	0.033061	0.006023	0.001564	0.004692	88	{'select__k': 88}	0.618037	0.770075	-1
88	0.033869	0.006219	0.001562	0.004686	89	{'select__k': 89}	0.614688	0.779204	-1
89	0.033209	0.006114	0.001564	0.004693	90	{'select__k': 90}	0.607260	0.774012	-1
90	0.032983	0.003416	0.000000	0.000000	91	{'select__k': 91}	0.609957	0.775318	-1
91	0.034471	0.006700	0.000000	0.000000	92	{'select__k': 92}	0.610004	0.771586	-0

92 rows × 31 columns

Selezioniamo solo i dati relativi all'errore negli split di train e test Per poter costruire i seguenti boxplot.

```
In [25]: #lable di tutte le colonne
column_names=list(risultati)
#lable di ciò che non ci interessa nei nuovi dataframe
column_names=[x for x in column_names if x[0:5]!="split"]
#creiamo un nuovo dataframe eliminando queste colonne
data=risultati.drop(column_names,axis=1)

#creiamo due nuovi dataframe con solo split test e split train
column_names=list(data)
data_train=data.drop(column_names[:int(len(column_names)/2)],axis = 1)
data_test=data.drop(column_names[int(len(column_names)/2):],axis = 1)
display(data_train.iloc[[0,1, -1]])
display(data_test.iloc[[0,1, -1]])
```

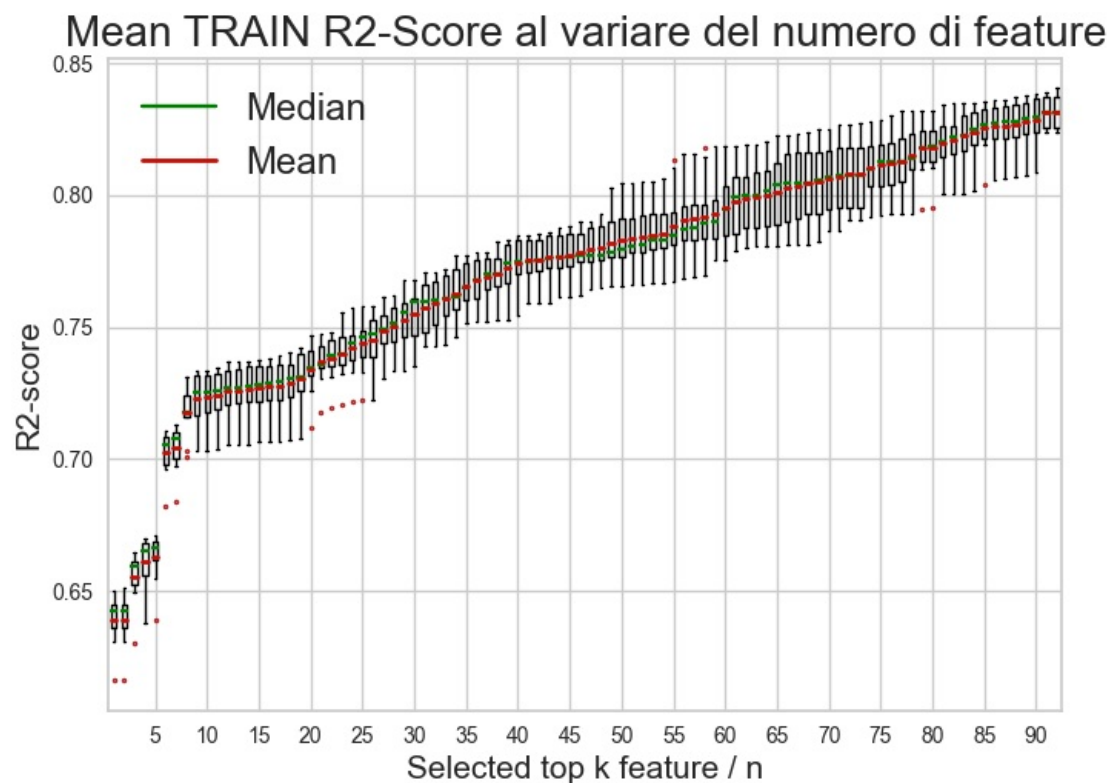
	split0_train_score	split1_train_score	split2_train_score	split3_train_score	split4_train_score	split5_train_score	split6_train_score	split7_train_score
0	0.648089	0.630859	0.641399	0.635630	0.615925	0.645136	0.642998	
1	0.648145	0.630861	0.641417	0.635796	0.615938	0.645138	0.643208	
91	0.838737	0.823903	0.840424	0.827870	0.824804	0.824530	0.832576	

	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	split5_test_score	split6_test_score	split7_test_score
0	0.558988	0.709693	0.609385	0.669434	0.808373	0.583152	0.563823	0.596
1	0.559031	0.709429	0.609792	0.664213	0.808627	0.583285	0.560891	0.596
91	0.610004	0.771586	-0.665292	0.723450	0.777670	0.786408	0.665089	0.624

Plottiamo le performance al variare del parametro k Nell'asse x il numero di feature selezionate, nell'asse y il relativo error_rate per il nostro dataset.

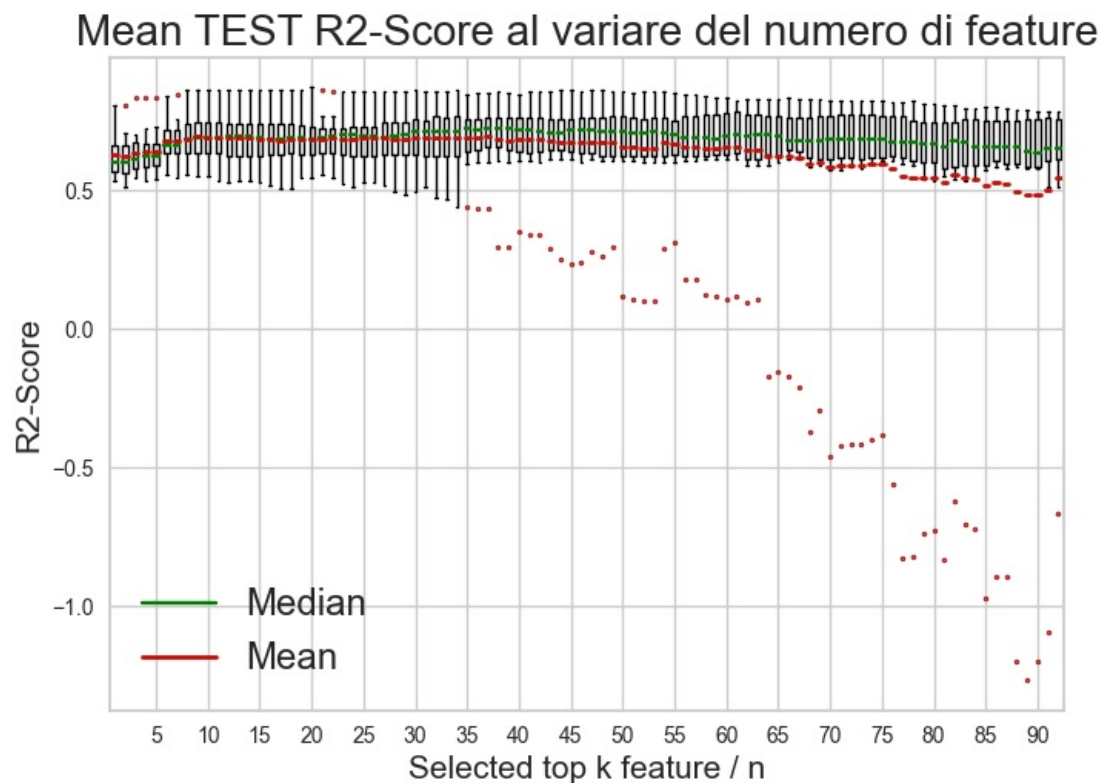
```
In [26]: #plot train
boxdata=data_train.values
flierprops = dict(marker='.',markededgecolor='firebrick', markersize=3, linestyle='none')
medianprops = dict(linestyle='-', linewidth=1.75, color='green')
meanprops = dict(linestyle='-', linewidth=2,color='r')

bp=plt.boxplot(boxdata.tolist(), flierprops=flierprops, medianprops=medianprops, showmeans= True, meanline = True)
plt.xticks([x for x in range(1,len(risultati["mean_train_score"])+1) if x%5==0],[str(x) for x in param[0]["select__k"]])
plt.xlabel("Selected top k feature / n",fontsize=15)
plt.ylabel("R2-score", fontsize=15)
plt.grid(True)
plt.legend([bp['medians'][0], bp['means'][0]], ['Median', 'Mean'], fontsize='x-large')
plt.title("Mean TRAIN R2-Score al variare del numero di feature",fontsize=20)
plt.show()
```



```
In [27]: #plot test
boxdata=data_test.values
flierprops = dict(marker='.',markeredgecolor='firebrick', markersize=3, linestyle='none')
medianprops = dict(linestyle='-', linewidth=1.75, color='green')
meanprops = dict(linestyle='-', linewidth=2,color='r')

bp=plt.boxplot(boxdata.tolist(), flierprops=flierprops, medianprops=medianprops, showmeans=True, meanline = True)
plt.xticks([x for x in range(1,len(risultati["mean_test_score"])+1) if x%5==0],[str(x) for x in param[0]["selected_top_k_feature"]])
plt.xlabel("Selected top k feature / n",fontsize=15)
plt.ylabel("R2-Score",fontsize=15)
plt.grid(True)
plt.legend([bp['medians'][0], bp['means'][0]], ['Median', 'Mean'], fontsize='x-large')
plt.title("Mean TEST R2-Score al variare del numero di feature",fontsize=20)
plt.show()
```



Alla luce dei seguenti dati potrebbe essere conveniente non utilizzare il miglior estimatore, in quel caso basterebbe creare una pipeline allo stesso modo, utilizzando come parametro il parametro desiderato, senza l'utilizzo di GridSearchCV

Es (k=20):

```
best = Pipeline([('select',SelectKBest(score_func=f_regression,k=20)),('scaler', StandardScaler()), ('regr',  
linear_model.LinearRegression())])
```

Calcoliamo Akaike Information Criterion (**AIC**) per il modello con featre da 1-92 e vediamo il contenuto informativo

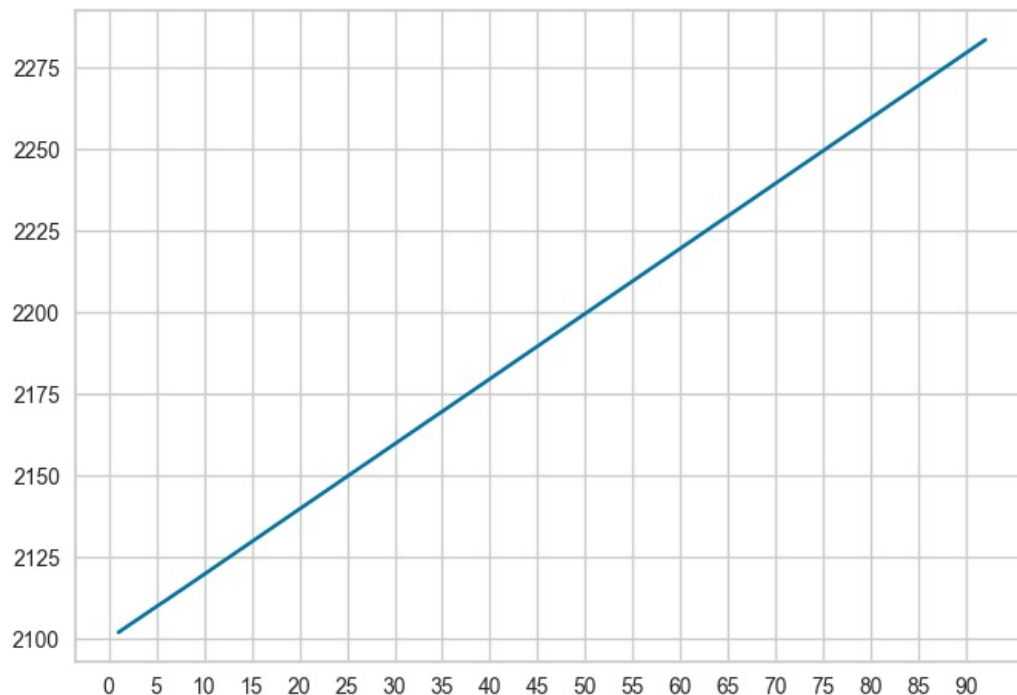
```
In [28]: import statsmodels.api as sm
```

```
AICs=[]  
print("Fitting model:",end=" ")  
for sk in [x+1 for x in range(92)]:  
  
    print(f"{sk}. ",end="")  
    SKB=SelectKBest(score_func=f_regression, k=sk)  
    t_X=SKB.fit_transform(F_x,F_y)  
    SSC=StandardScaler()  
    t_X=SSC.fit_transform(t_X)  
  
    #fit regression model  
    m_a = sm.OLS(F_y, t_X).fit()  
  
    AICs.append(m_a.aic)
```

Fitting model: 1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.
36.37.38.39.40.41.42.43.44.45.46.47.48.49.50.51.52.53.54.55.56.57.58.59.60.61.62.63.64.65.66.67.68.69.70.71.72.
73.74.75.76.77.78.79.80.81.82.83.84.85.86.87.88.89.90.91.92.

Plottiamo

```
In [29]: plt.plot([x+1 for x in range(92)],AICs)  
plt.xticks([x for x in range(92) if x%5==0])  
plt.grid(True)  
plt.show()  
print(np.array(AICs))
```



```
[2101.9976135 2103.99747767 2105.95883318 2107.94233705 2109.9398863  
2111.8443702 2113.83974368 2115.799476 2117.79396716 2119.7939548  
2121.79225957 2123.78857769 2125.78630377 2127.78626686 2129.78593527  
2131.78472079 2133.78455855 2135.78455839 2137.7829384 2139.77148204  
2141.76107997 2143.75963947 2145.75692412 2147.75368733 2149.75307177  
2151.7370421 2153.73531193 2155.72805312 2157.7279262 2159.72387947  
2161.72385473 2163.71296973 2165.71108573 2167.70189471 2169.69005501  
2171.6871169 2173.68691383 2175.67466407 2177.67293008 2179.67279793  
2181.67149657 2183.67123677 2185.6711374 2187.67074305 2189.66430497  
2191.66428543 2193.66426159 2195.66350867 2197.65854688 2199.65802686  
2201.65694161 2203.65532048 2205.65502246 2207.65410376 2209.64916544  
2211.64913397 2213.64507271 2215.64471844 2217.64397542 2219.64314697  
2221.63351278 2223.63268806 2225.63232116 2227.63113818 2229.59185487  
2231.59069609 2233.5850123 2235.58175532 2237.5816242 2239.58152972  
2241.57871475 2243.57626367 2245.5752058 2247.57519597 2249.57336202  
2251.57287587 2253.57233901 2255.57173555 2257.57081415 2259.57063266  
2261.56815373 2263.56792855 2265.56346009 2267.56159249 2269.56149801  
2271.55278728 2273.54575249 2275.54493718 2277.54419791 2279.54404249  
2281.5438508 2283.54274766]
```

Essendo il modello molto leggero il tempo di fit è sempre uguale, quindi è ovvio che avere più feature sia più informativo e quindi un AIC score migliore

Visualize regression with yellowbrick

y = true value

\hat{y} = predicted value

Residuals = $y_{\text{pred}} - y_{\text{true}}$

Purtroppo non è possibile cambiare lo scoring method della figura, quindi nella legenda comparirà R^2 invece del nostro error rate, che viene quindi calcolato separatamente.

----- USE BEST PREDICTOR ON **TRAIN** DATA -----

CALCOLIAMO l'error_rate medio nel train.

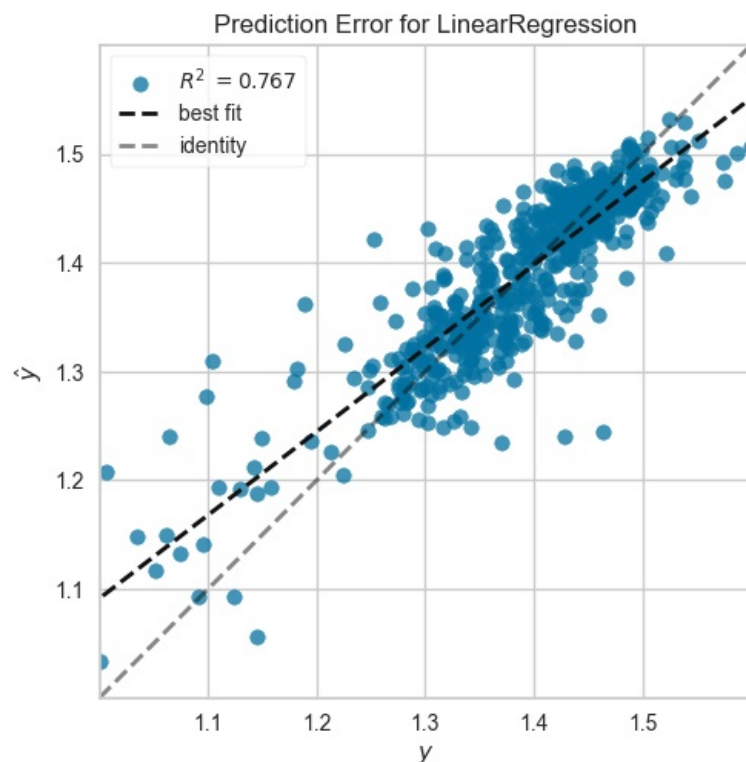
```
In [30]: y_pred=best.predict(F_x)
print(f"\nErrore medio del dataset: {error_rate(F_y,y_pred):.3f}% !")
```

Errore medio del dataset: 2.221% !

```
In [31]: #F_x feature 92 #F_y target

visualizer= PredictionError(best,bestfit=True)

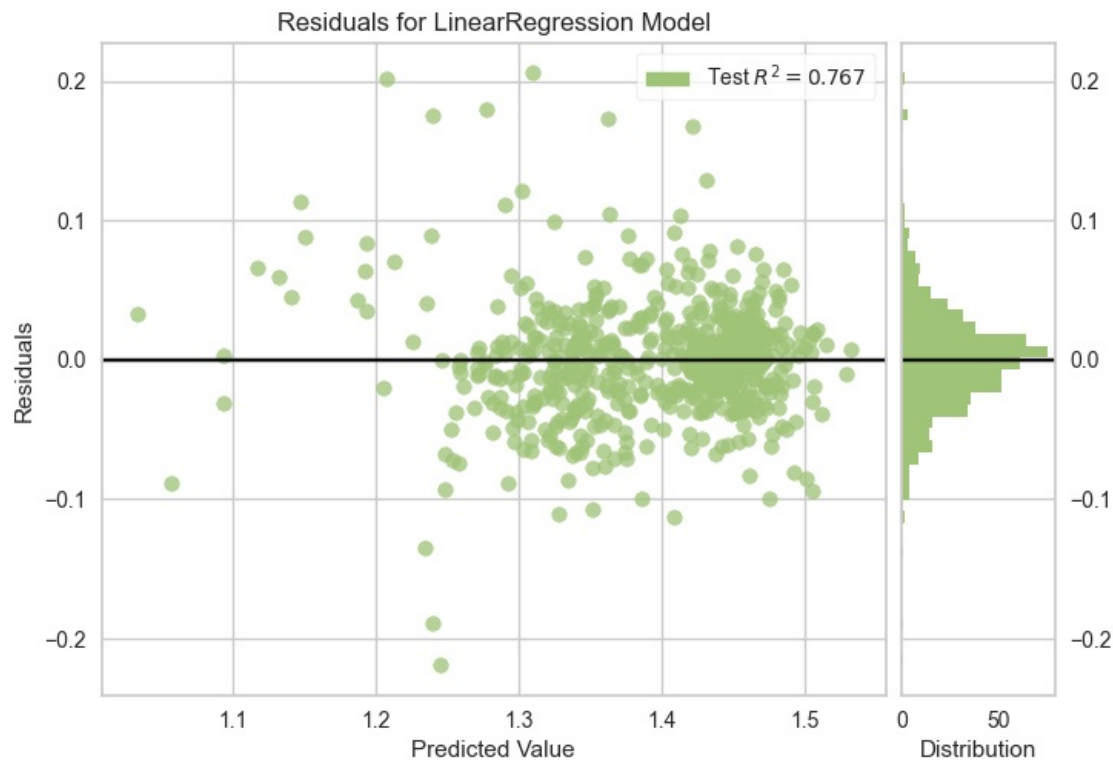
#visualizer.fit(F_x, F_y) # Fit the training data in the visualizer # best is already fitted on F_x F_y!
visualizer.score(F_x, F_y)#train # Evaluate the model on data (train or test)
visualizer.show() # Finalize and render the figure ;
```



```
Out[31]: <AxesSubplot:title={'center':'Prediction Error for LinearRegression'}, xlabel='$y$', ylabel='$\hat{y}$'>
```

```
In [32]: visualizer= ResidualsPlot(best)

#visualizer.fit(F_x, F_y)# Fit the training data in the visualizer # best is already fitted on F_x F_y!
visualizer.score(F_x, F_y)#train # Evaluate the model on data (train or test)
visualizer.show() # Finalize and render the figure ;
#facendo il fit questo grafico restituisce anche il train, ma nel nostro caso train e test sono gli stessi, #st.
```



Out[32]: <AxesSubplot:title={'center': 'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

Dall'istogramma laterale vediamo anche la distribuzione dell'errore in metri!

----- USE BEST PREDICTOR ON TEST DATA -----

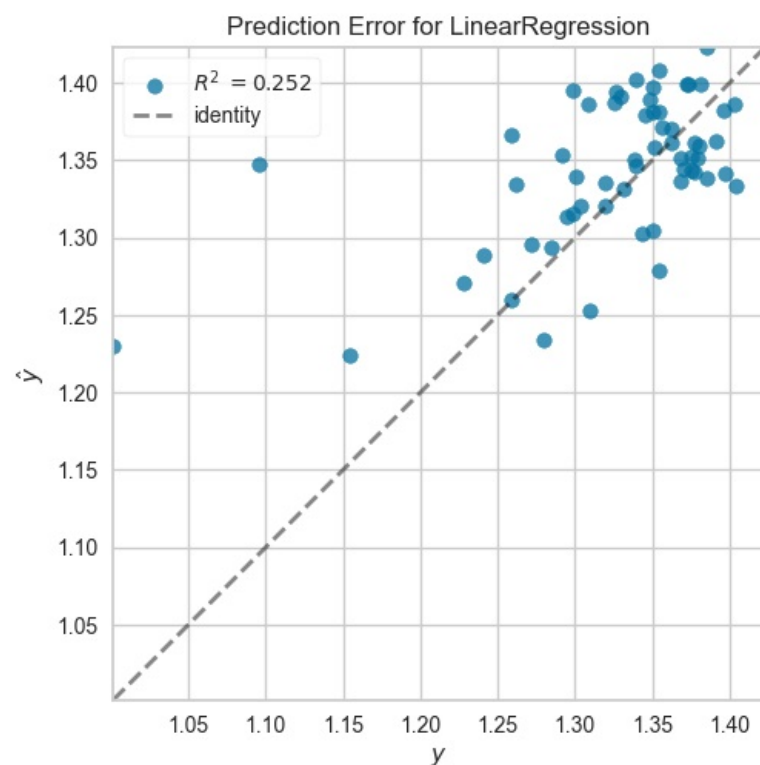
```
In [33]: #predict with best predictor
b_pred=best.predict(F_x_test)

#evaluate error_rate
print(f"Error rate: {error_rate(F_y_test,b_pred):.3f}% ")

Error rate: 3.389%
```

```
In [34]: visualizer= PredictionError(best,bestfit=False)

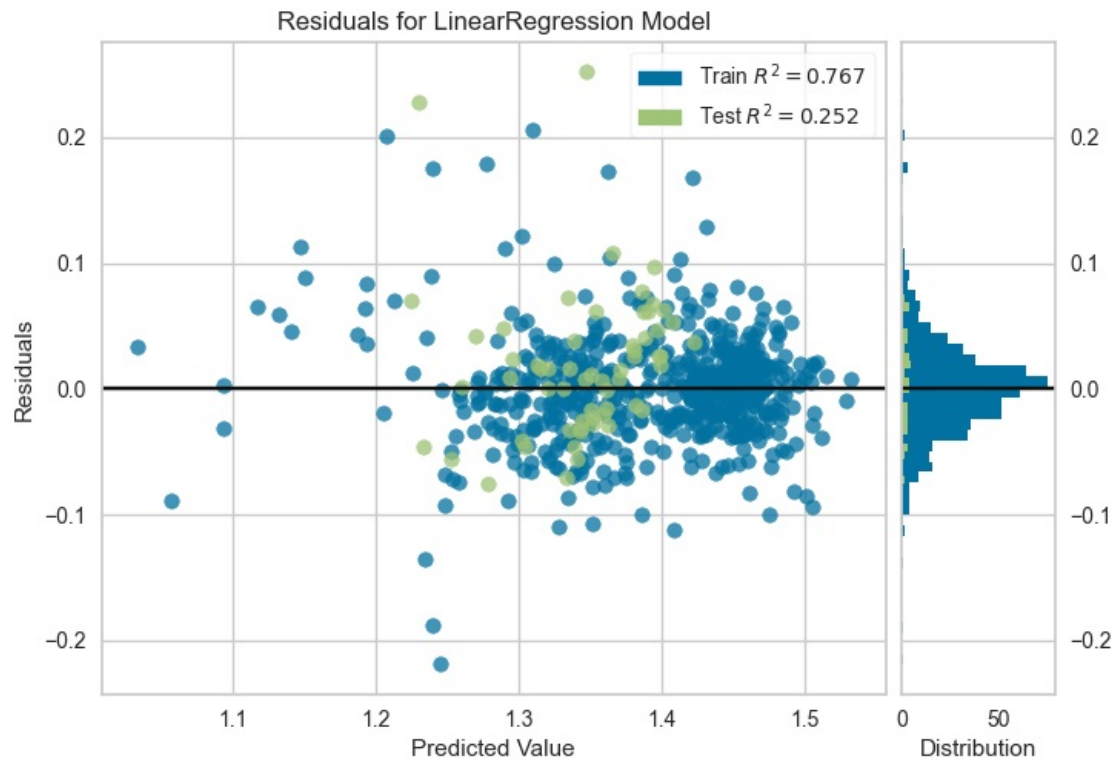
#visualizer.fit(F_x, F_y) # Fit the training data in the visualizer # best is already fitted on F_x F_y!
visualizer.score(F_x_test, F_y_test)#test # Evaluate the model on data (train or test)
visualizer.show() # Finalize and render the figure
```



Out[34]: <AxesSubplot:title={'center': 'Prediction Error for LinearRegression'}, xlabel='y', ylabel='y-hat'>

```
In [35]: visualizer= ResidualsPlot(best)
```

```
visualizer.fit(F_x, F_y) # Fit the training data in the visualizer # best is already fitted on F_x F_y!  
visualizer.score(F_x_test, F_y_test)#test # Evaluate the model on data (train or test)  
visualizer.show() # Finalize and render the figure
```



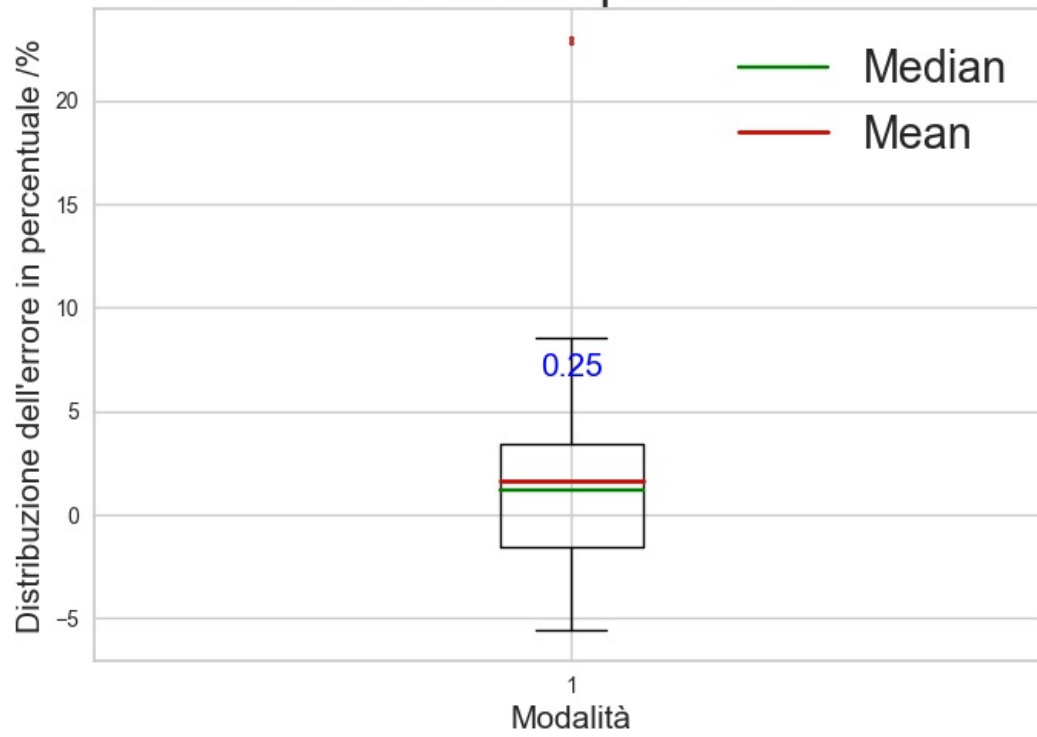
```
Out[35]: <AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>
```

Dall'istogramma laterale vediamo anche la distribuzione dell'errore in metri!

Vediamo l' R^2 score e plottiamolo sopra il boxplot che rappresenta la distribuzione dell'**ERRORE IN METRI**

```
In [36]: #predict  
b_pred=best.predict(F_x_test)  
mt_err_dist=((b_pred-F_y_test)/F_y_test)*100  
r2_scores=r2_score(F_y_test,b_pred)  
  
flierprops = dict(marker='.',markedgecolor='firebrick', markersize=3, linestyle='none')  
medianprops = dict(linestyle='-', linewidth=1.75, color='green')  
meanprops = dict(linestyle='-', linewidth=2, color='r')  
  
bp1=plt.boxplot(mt_err_dist.tolist(), flierprops=flierprops, medianprops=medianprops, showmeans=True, meanline  
plt.text(1,np.mean(mt_err_dist)+np.std(mt_err_dist),int(r2_scores*100)/100,horizontalalignment="center",color="r")  
plt.xlabel("Modalità",fontsize=15)  
plt.ylabel("Distribuzione dell'errore in percentuale %",fontsize=15)  
plt.grid(True)  
plt.legend([bp1['medians'][0], bp1['means'][0]], ['Median', 'Mean'], fontsize='xx-large')  
plt.title("Mean TEST Score per le 5 modalità",fontsize=25)  
plt.show()
```

Mean TEST Score per le 5 modalità



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js