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,ful
```

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]:
```
print(stop_list)
```

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

"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 [4]:
```
print(type(DATASET))
for c,v in DATASET.items():
    print(c,type(v),len(v),type(v[0]),v[0].keys())
```

```
<class 'dict'>
armhand <class 'list'> 723 <class 'dict'> dict_keys(['target', 'Acc_X', 'Acc_Y', 'Acc_Z', 'Gyr_X', 'Gyr_Y', 'Gy
r_Z', 'SensorTimestamp'])
pocket <class 'list'> 685 <class 'dict'> dict_keys(['target', 'Acc_X', 'Acc_Y', 'Acc_Z', 'Gyr_X', 'Gyr_Y', 'Gyr
_Z', 'SensorTimestamp'])
calling <class 'list'> 764 <class 'dict'> dict_keys(['target', 'Acc_X', 'Acc_Y', 'Acc_Z', 'Gyr_X', 'Gyr_Y', 'Gy
r_Z', 'SensorTimestamp'])
swing <class 'list'> 1385 <class 'dict'> dict_keys(['target', 'Acc_X', 'Acc_Y', 'Acc_Z', 'Gyr_X', 'Gyr_Y', 'Gyr
_Z', 'SensorTimestamp'])
handheld <class 'list'> 5827 <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 [5]:
```
filtWDE(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 [6]:
```
Feature_DS=f_ext_WDE(DATASET)
```

```
Extracting armhand:#########
Extracting pocket:#########
Extracting calling:#########
Extracting swing:#########
Extracting handheld:#########
```

In [7]:
```
for k,v in Feature_DS.items():
    print(k,type(v),v.keys(),len(v['feature']),len(v['feature'][0]),len(v['target']))
```

```
armhand <class 'dict'> dict_keys(['feature', 'target']) 723 92 723
pocket <class 'dict'> dict_keys(['feature', 'target']) 685 92 685
calling <class 'dict'> dict_keys(['feature', 'target']) 764 92 764
swing <class 'dict'> dict_keys(['feature', 'target']) 1385 92 1385
handheld <class 'dict'> dict_keys(['feature', 'target']) 5827 92 5827
```

- **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 [8]:
```
DS_train,DS_test = trueeqWDE(Feature_DS,stop_list)
```

In [9]:
```
for k,v in DS_train.items():
    print(k,type(v),v.keys(),len(v['feature']),len(v['feature'][0]),len(v['target']))
```

```
armhand <class 'dict'> dict_keys(['feature', 'target']) 500 92 500
pocket <class 'dict'> dict_keys(['feature', 'target']) 500 92 500
calling <class 'dict'> dict_keys(['feature', 'target']) 500 92 500
swing <class 'dict'> dict_keys(['feature', 'target']) 500 92 500
handheld <class 'dict'> dict_keys(['feature', 'target']) 500 92 500
```

In [10]:
```
for k,v in DS_test.items():
    print(k,type(v),v.keys(),len(v['feature']),len(v['feature'][0]),len(v['target']))
```

```
armhand <class 'dict'> dict_keys(['feature', 'target']) 50 92 50
pocket <class 'dict'> dict_keys(['feature', 'target']) 50 92 50
calling <class 'dict'> dict_keys(['feature', 'target']) 50 92 50
swing <class 'dict'> dict_keys(['feature', 'target']) 50 92 50
handheld <class 'dict'> dict_keys(['feature', 'target']) 50 92 50
```

- **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 [11]:
```
regr_dataset_train, regr_dataset_test = full_regr_dataset(DS_train,DS_test)
```

In [12]:
```
for k,v in regr_dataset_train.items():
    print(k,type(v),len(v),type(v[0]))
```

```
feature <class 'list'> 2500 <class 'list'>
target <class 'list'> 2500 <class 'float'>
```

In [13]:
```
for k,v in regr_dataset_test.items():
    print(k,type(v),len(v),type(v[0]))
```

```
feature <class 'list'> 250 <class 'list'>
target <class 'list'> 250 <class 'float'>
```

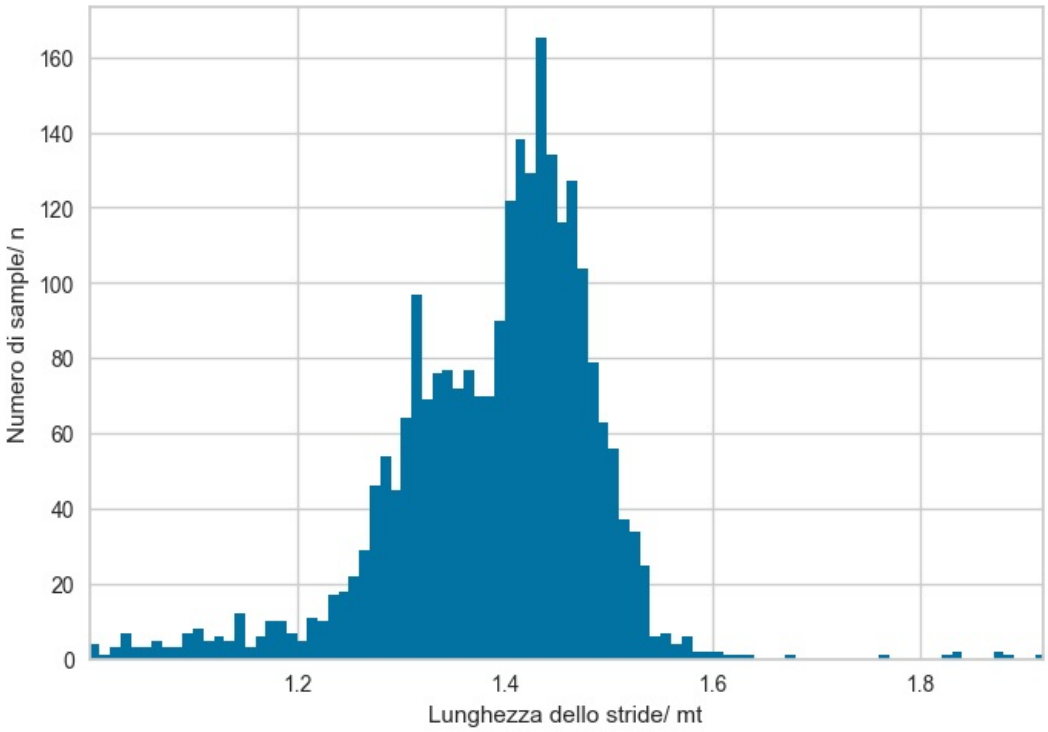
Isoliamo i singoli **F_x,F_y** rispettivamete feature e target del train e **F_x_test,F_y_test** rispettivamente feature e target di test

In [14]:
```
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 distibuite le lunghezze registrate degli stride

In [15]:
```
#istogramma dei target F_y
ist_stride_lenght(F_y)
```

```
Registered Stride lenght: Min:  1.000034177353732 , Max:  1.917364683119593
Media= 1.3906684693896598; Deviazione standard= 0.09820942152593022.
```



Out[15]: 0

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

Salviamo la lista dei nomi delle feature.

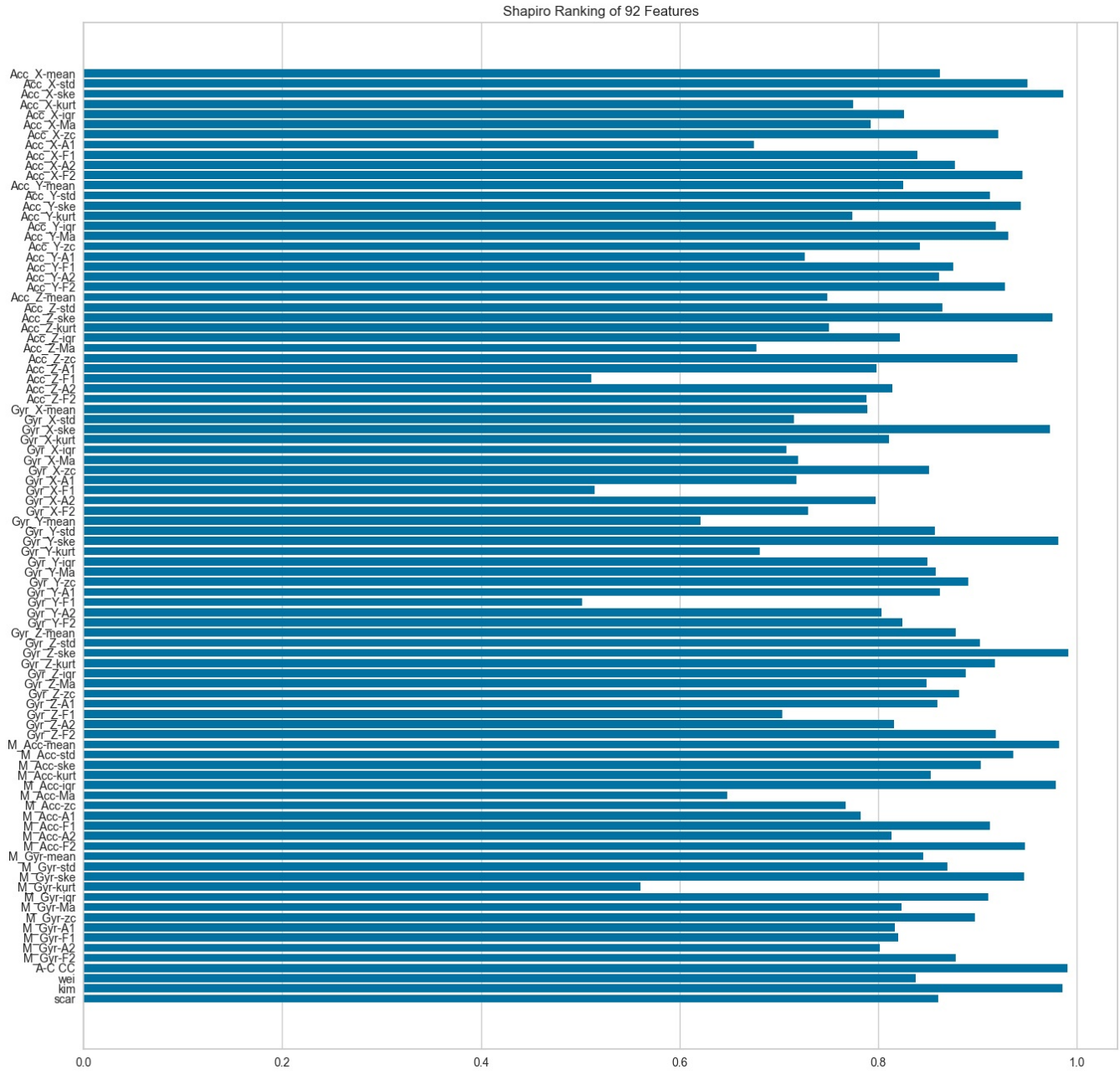
In [16]:
```
feature_name=['Acc_X-mean', 'Acc_X-std', 'Acc_X-ske', 'Acc_X-kurt', 'Acc_X-iqr', 'Acc_X-Ma', 'Acc_X-zc', 'Acc_X
print(f"Abbiamo: {len(feature_name)} feature.")
```

```
Abbiamo: 92 feature.
```

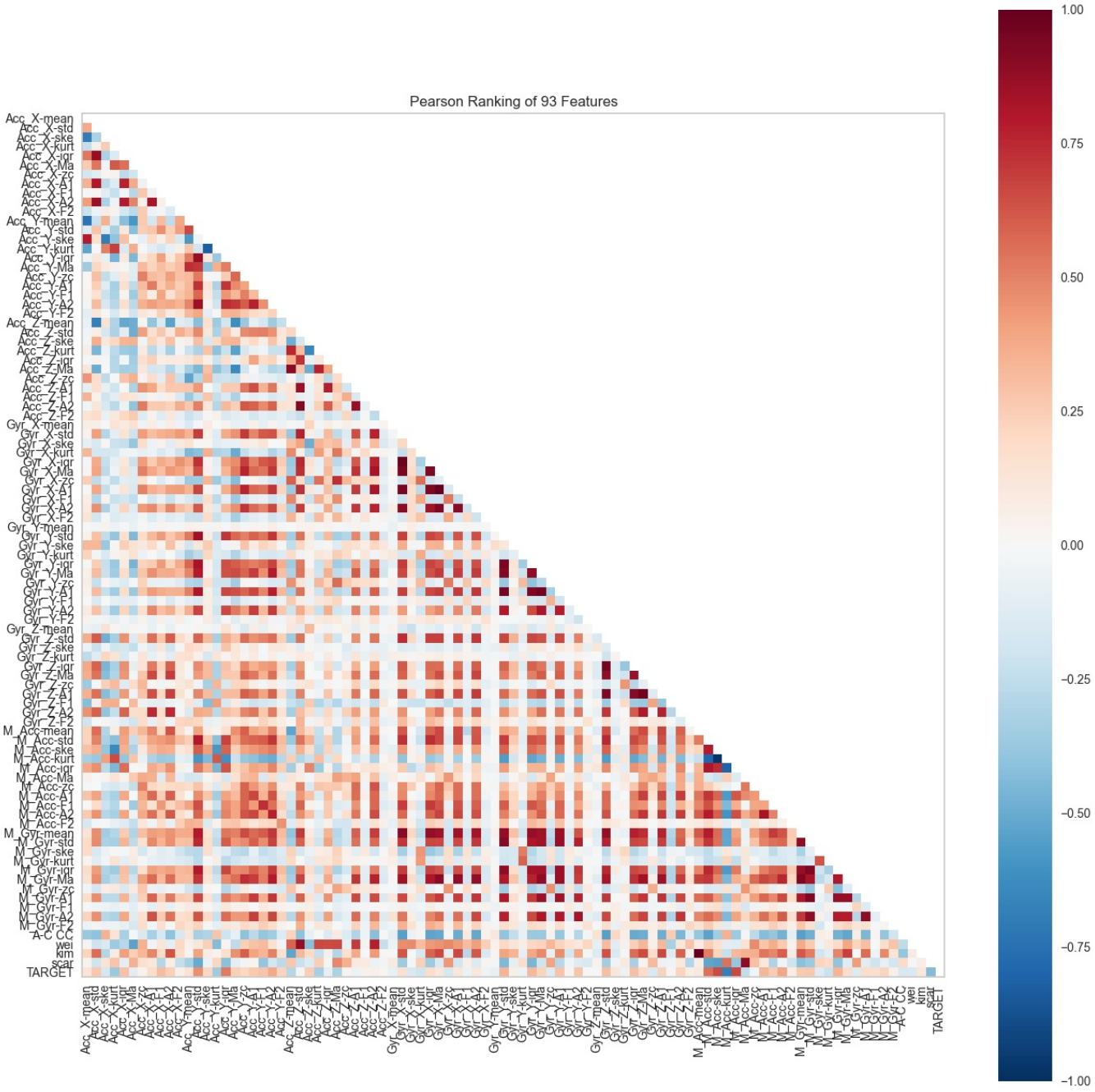
rank1d e rank2d offrono la possibilità di visualizzare in modo semplice ed intuitivo il ranking delle feature. In particolare:

- **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 [17]:
```
#rank1d
_, axes = plt.subplots(ncols=1, figsize=(16,16))
rank1d(F_x,F_y,features=feature_name,ax=axes , show=False)
plt.show()
```



In [18]:
```
####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 vede

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 [19]:
```
#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_}
```

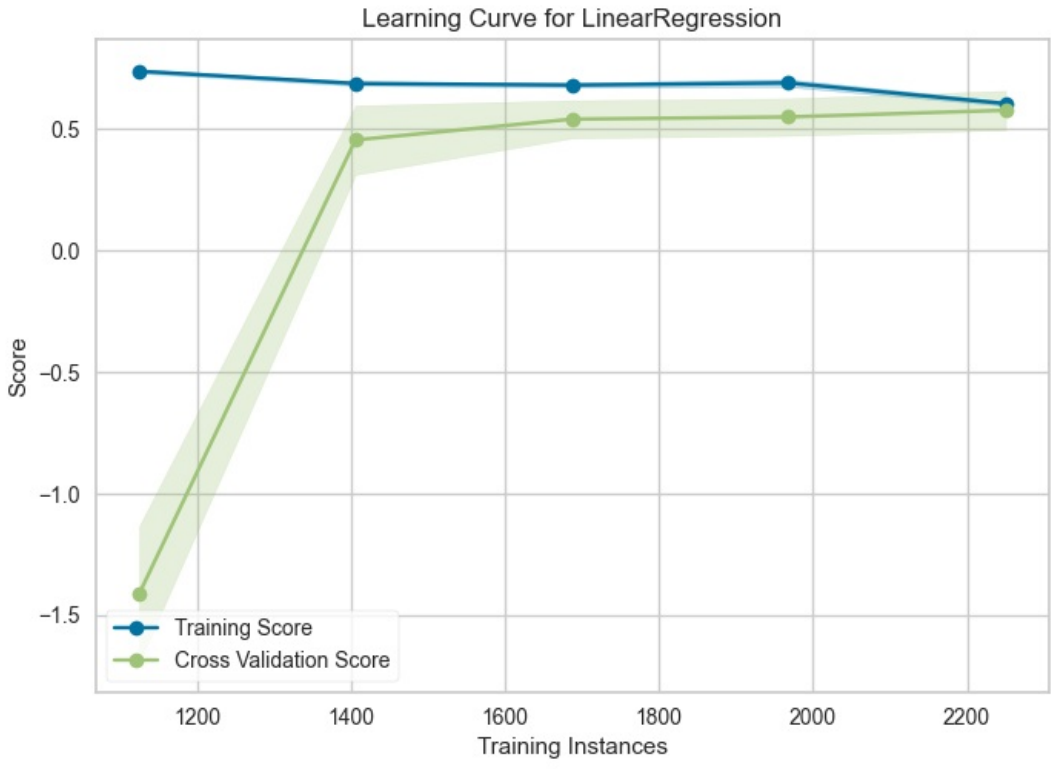
```
La pipeline è composta da:  Pipeline(steps=[('select',
                 SelectKBest(score_func=<function f_regression at 0x00000276C53168C0>)),
                ('scaler', StandardScaler()), ('regr', LinearRegression())])
Fitting 10 folds for each of 92 candidates, totalling 920 fits


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

 L'R2-score medio è di 0.578 !
```

-----MODEL EVALUETION-----

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 convege 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 [20]:
```
viz = LearningCurve(best, scoring="r2",cv=kf,train_sizes=np.linspace(0.5,1.0,5))

viz.fit(F_x,F_y) # Fit the data to the visualizer
viz.show()
```

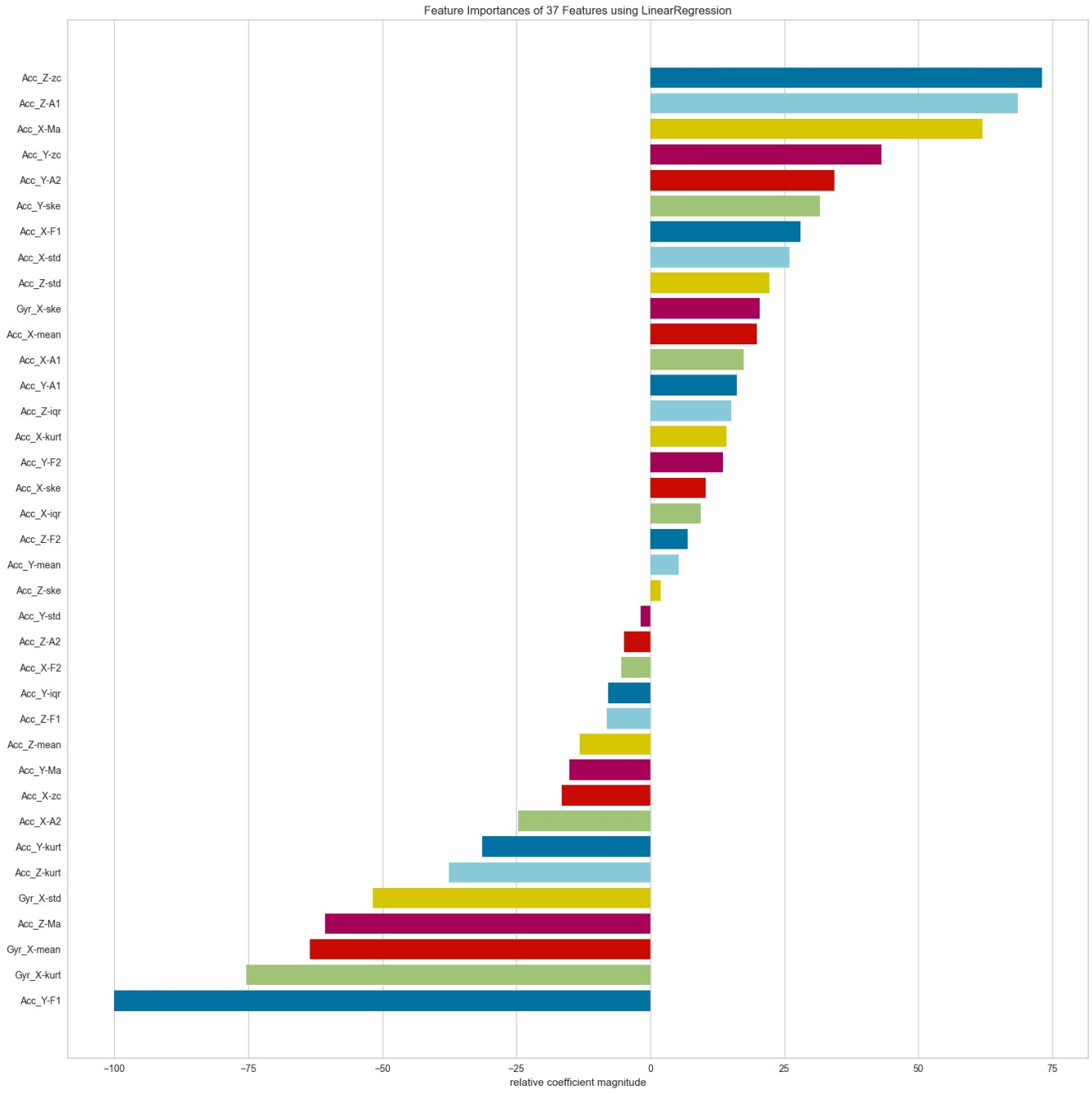


Out[20]:
```
<AxesSubplot:title={'center':'Learning Curve for LinearRegression'}, xlabel='Training Instances', ylabel='Score
'>
```

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

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

In [21]:
```
#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[21]:
```
<AxesSubplot:title={'center':'Feature Importances of 37 Features using LinearRegression'}, xlabel='relative coe
fficient 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 [22]:
```
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_tes
0	0.004838	0.005282	0.001602	0.003204	1	{'select__k': 1}	0.450022	0.390146	0.
1	0.004383	0.006744	0.001599	0.004798	2	{'select__k': 2}	0.478918	0.438418	0.
2	0.008367	0.007658	0.000000	0.000000	3	{'select__k': 3}	0.476778	0.438588	0.
3	0.007890	0.004950	0.000000	0.000000	4	{'select__k': 4}	0.480186	0.447377	0.
4	0.005345	0.007007	0.000000	0.000000	5	{'select__k': 5}	0.480787	0.441324	0.
...	...	...	...	...	...	...	...	...	
87	0.040354	0.007491	0.001563	0.004690	88	{'select__k': 88}	0.575756	0.405362	0.
88	0.043446	0.004662	0.000800	0.002401	89	{'select__k': 89}	0.575789	0.406029	0.
89	0.041914	0.006290	0.000800	0.002401	90	{'select__k': 90}	0.574553	0.410649	0.
90	0.040855	0.006245	0.002366	0.005029	91	{'select__k': 91}	0.573920	0.411818	0.
91	0.041539	0.005880	0.000000	0.000000	92	{'select__k': 92}	0.574465	0.413756	0.

92 rows × 31 columns

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

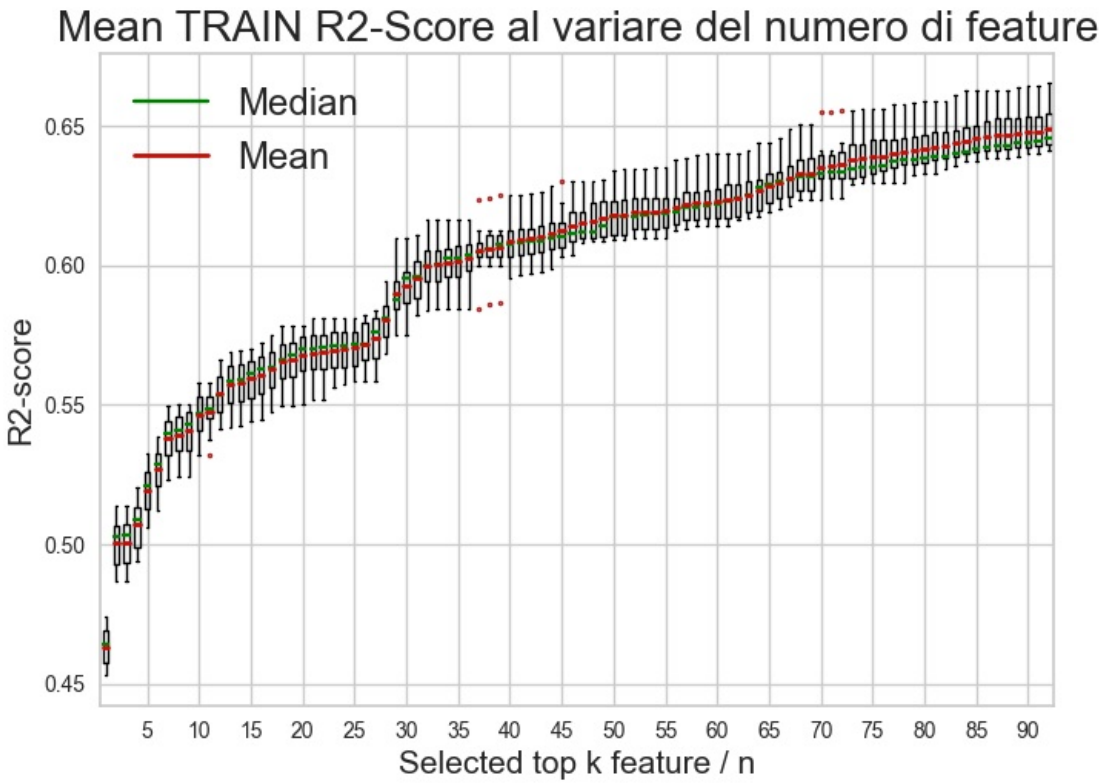
In [23]:
```
#lable di tutte le colonne
column_names=list(risultati)
#lable di ciò che non ci interessa nei nuovi dataframe
column_names=[x for x in column_names if x[0:5]!="split"]
#creiamo un nuovo dataframe eliminando queste colonne
data=risultati.drop(column_names,axis=1)

#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_t
0	0.464714	0.470476	0.473822	0.457785	0.463398	0.457532	0.469205	
1	0.503039	0.506588	0.513599	0.494560	0.502548	0.492224	0.507090	
91	0.649462	0.665213	0.646298	0.644582	0.657402	0.643295	0.644155	

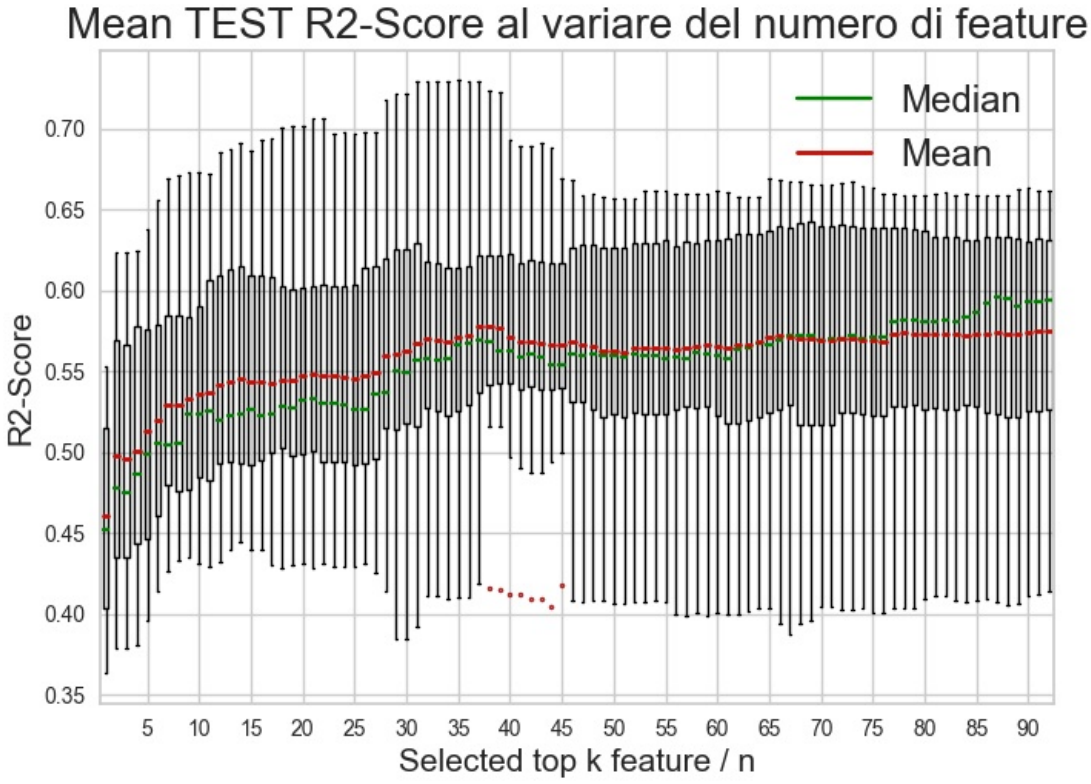
	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	split5_test_score	split6_test_score	split7_test_s
0	0.450022	0.390146	0.362915	0.515766	0.454145	0.513000	0.400416	0.552
1	0.478918	0.438418	0.378827	0.558289	0.476921	0.573329	0.432778	0.623
91	0.574465	0.413756	0.585166	0.632264	0.494389	0.626886	0.602954	0.649

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 [24]:
```
#plot train
boxdata=data_train.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 = Tr
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]["sele
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 [25]:
```
#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 = Tr
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]["selec
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())])

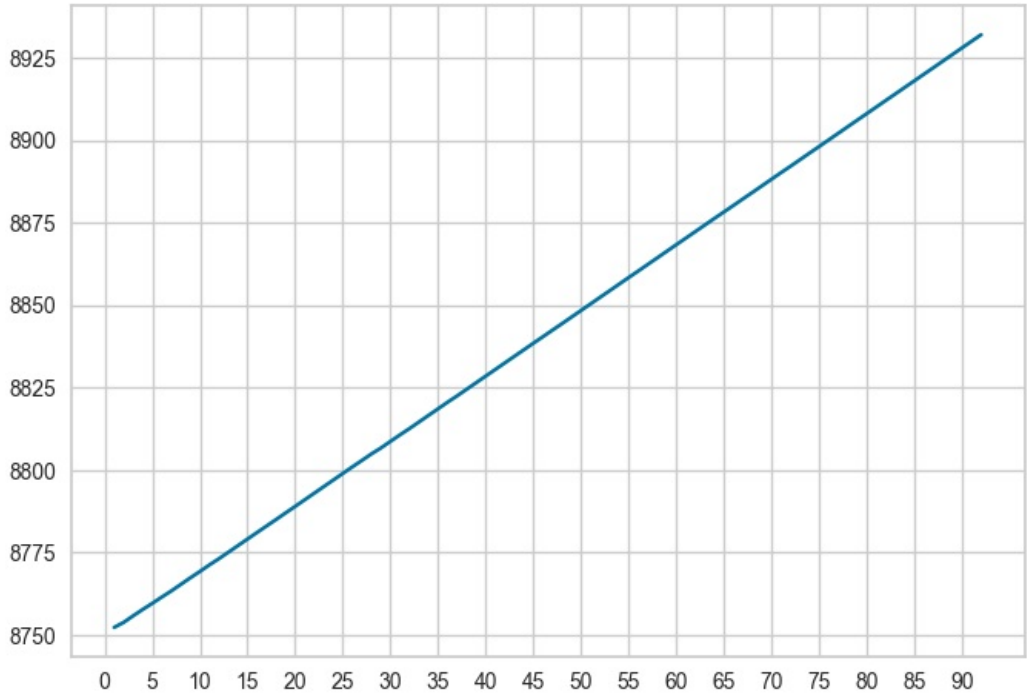
Calcoliamo Akaike Information Criterion (**AIC**) per il modello con featre da 1-92 e vediamo il contenuto informativo

In [26]:
```
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 [27]:
```
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))
```




```
[8752.30020533 8753.83647445 8755.83578534 8757.75646515 8759.60342103
 8761.50968194 8763.37410363 8765.3597055  8767.35525148 8769.27026313
 8771.27025977 8773.14544736 8775.13524917 8777.12189318 8779.11541703
 8781.09746457 8783.06634025 8785.03287415 8787.03016838 8789.00346309
 8791.00173873 8793.00148322 8795.00145256 8796.99371932 8798.99263193
 8800.98979215 8802.93049313 8804.91297659 8806.72028933 8808.719514
 8810.69932185 8812.61074351 8814.61061656 8816.60816252 8818.60801152
 8820.60742965 8822.52222618 8824.52089839 8826.51599179 8828.51585391
 8830.50940907 8832.50509104 8834.49591739 8836.48314426 8838.48200985
 8840.44490242 8842.44441282 8844.40681643 8846.40119434 8848.40117035
 8850.39996268 8852.3982899  8854.38896089 8856.38777849 8858.38766939
 8860.35864663 8862.35651425 8864.35628882 8866.35347494 8868.34995388
 8870.33726367 8872.33650419 8874.32436057 8876.31787785 8878.28224806
 8880.25252225 8882.25183205 8884.22521453 8886.22217171 8888.21695654
 8890.21458328 8892.15991177 8894.15984801 8896.15980132 8898.13388826
 8900.13336833 8902.13293694 8904.13076677 8906.12644885 8908.11264204
 8910.11165086 8912.10825292 8914.0893169  8916.08667481 8918.06973331
 8920.06760939 8922.05557742 8924.05548266 8926.05496475 8928.0531129
 8930.04294708 8932.03002509]
```

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_pred - y_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.

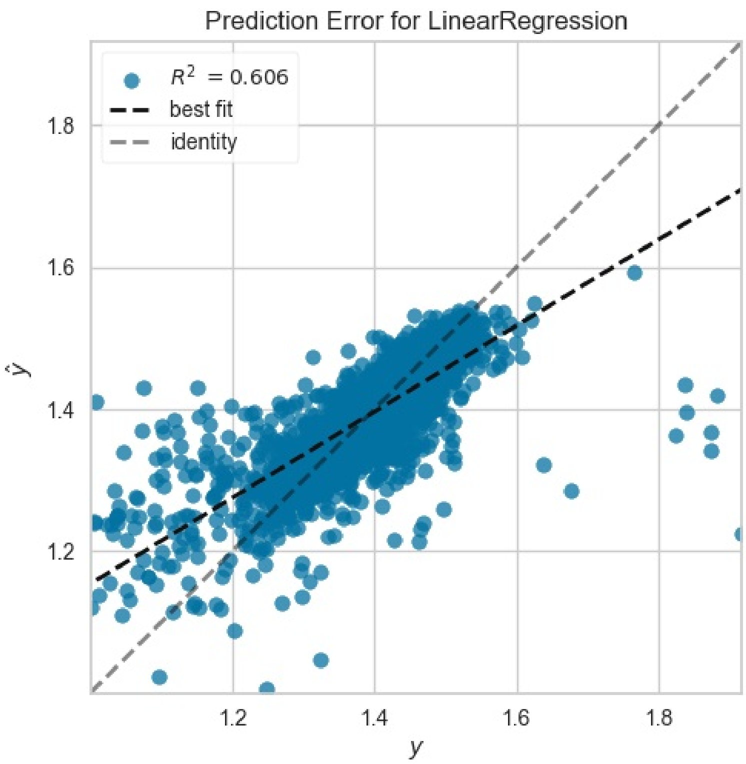
------------------------ USE BEST PREDICTOR ON **TRAIN** DATA ------------------------

CALCOLIAMO l'error_rate medio nel train.

In [28]:
```
y_pred=best.predict(F_x)
print(f"\nErrore medio del dataset: {error_rate(F_y,y_pred):.3f}% !")
```

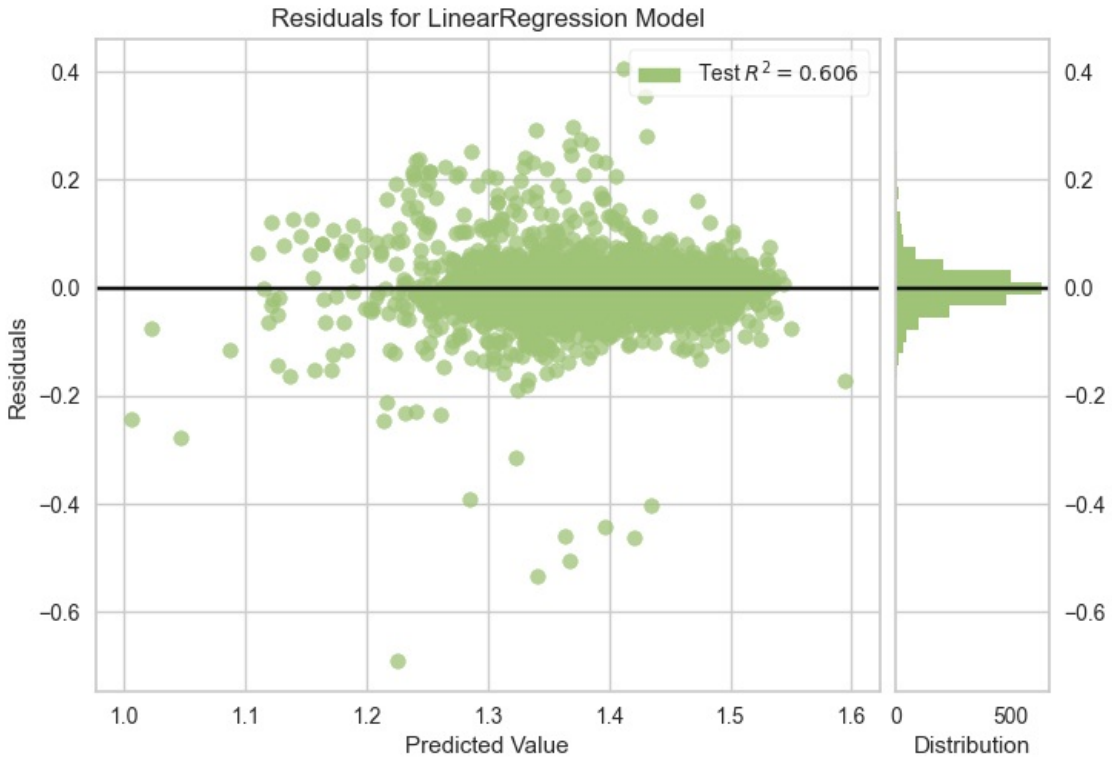
```
Errore medio del dataset: 2.761% !
```

In [29]:
```
#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[29]:
```
<AxesSubplot:title={'center':'Prediction Error for LinearRegression'}, xlabel='$y$', ylabel='$\\hat{y}$'>
```

In [30]:
```
visualizer= ResidualsPlot(best)

#visualizer.fit(F_x, F_y)# Fit the training data in the visualizer # best is alredy 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[30]: <AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

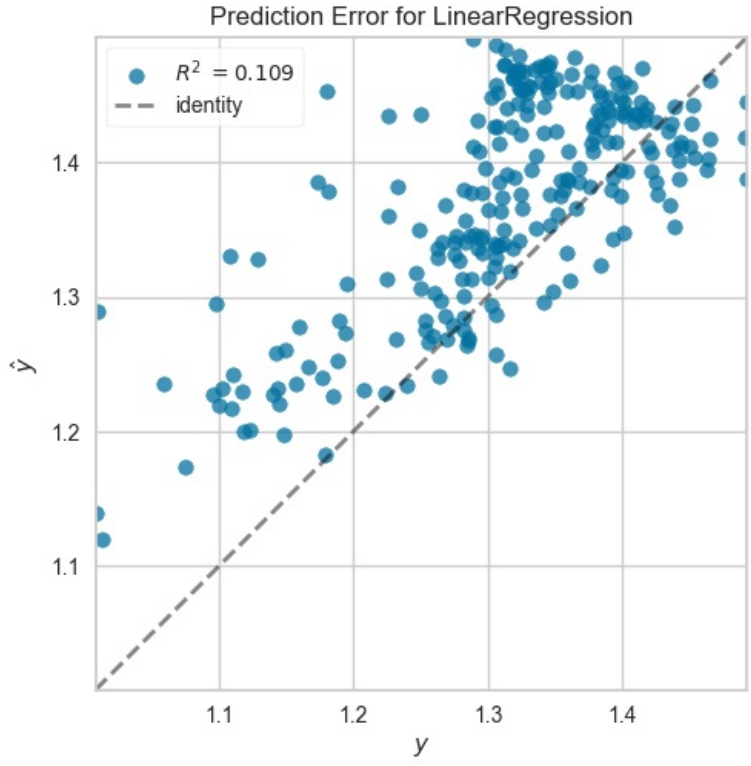
Dall'istogramma laterale vediamo anche la distribuzione dell'errore in metri!

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

In [31]:
```
#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}% ")
```

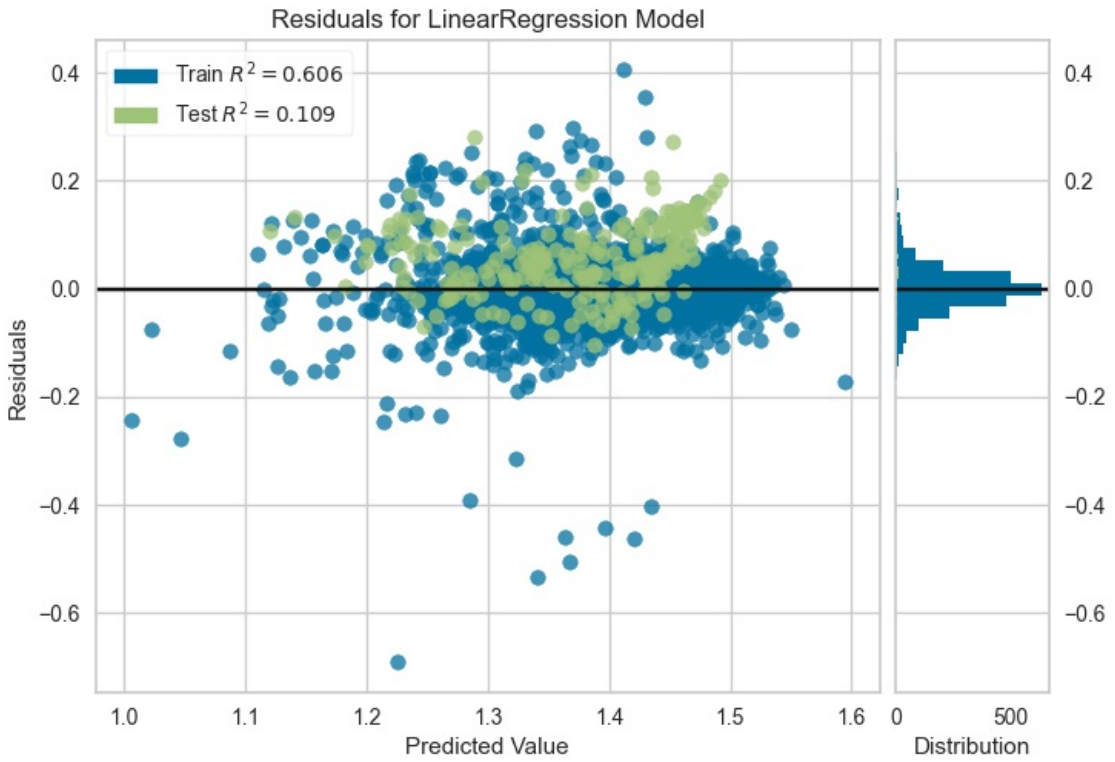
```
Error rate: 5.633%
```

In [32]:
```
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[32]: <AxesSubplot:title={'center':'Prediction Error for LinearRegression'}, xlabel='y', ylabel='$\\hat{y}$'>

In [33]:
```
visualizer= ResidualsPlot(best)

visualizer.fit(F_x, F_y) # Fit the training data in the visualizer # best is alredy 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
```

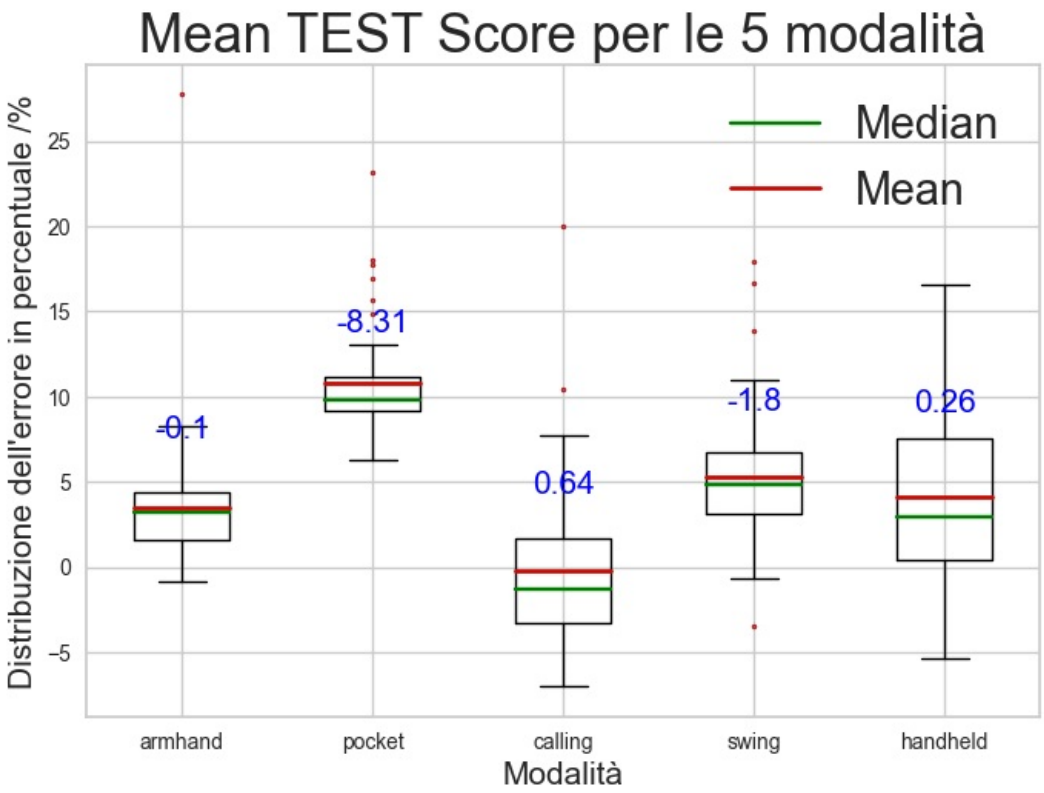


Out[33]:
```
<AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residua
ls'>
```

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

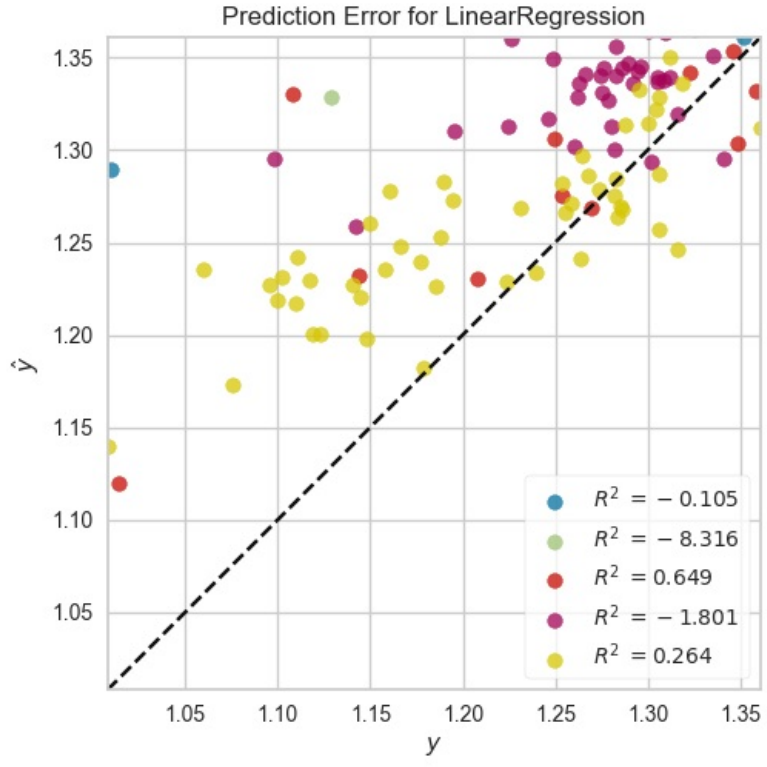
Vediamo l'R2 score per ogni singola modalità e plottiamolo sopra il boxplot che rappresenta la distribuzione dell' ERRORE IN METRI

In [34]:
```
#plot residuals distribution for every mode
mt_err_dist=[[],[],[],[],[]]
r2_scores=[]

start=0;
for stop in range(n_left,len(F_x_test)+n_left,n_left):
    file_feature = F_x_test[start : stop]
    file_target = F_y_test[start : stop]

    #predict
    b_pred=best.predict(file_feature)
    mt_err_dist[int(stop/n_left)-1]=((b_pred-file_target)/file_target)*100
    r2_scores.append(r2_score(file_target,b_pred))

    start=stop
mt_err_dist=np.array(mt_err_dist)

flierprops = dict(marker='.',markeredgecolor='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
for i in range(len(r2_scores)):
    plt.text(i+1,np.mean(mt_err_dist[i])+np.std(mt_err_dist[i]),int(r2_scores[i]*100)/100,horizontalalignment="
plt.xticks([x+1 for x in range(len(classi))],classi)
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()
```



In [35]:
```
vizz=PredictionError(best,bestfit=False,identity=False)
print(f"Label in ordine dall'alto verso il basso sono: {classi}")
st=0
sp=n_left
for i in range(len(classi)):
    if sp==n_left*6:
        break
    vizz.score(F_x_test[st:sp], F_y_test[st:sp])
    st=sp
    sp+=n_left
    vizz.finalize()
plt.plot([1,2],[1,2],"k--")
vizz.show()
```

```
Label in ordine dall'alto verso il basso sono: ['armhand', 'pocket', 'calling', 'swing', 'handheld']
```



Out[35]:
```
<AxesSubplot:title={'center':'Prediction Error for LinearRegression'}, xlabel='$y$', ylabel='$\\hat{y}$'>
```

Vediamo rispetto al modello fittato senza distinzione lo score R2 per ogni modalità ed anche la distribuzione delle varie modalità nel piano.