CapstoneNotebook

0.1 Importing necessary libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math as m
from sympy import *
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import *
from catboost import CatBoostRegressor
```

0.2 Task 1 : Data Wrangling

0.2.1 As load_boston is no longer available in sklearn, data is directly imported as a csv from kaggle into pandas. Differences could exist in the data.

[2]:
```
df = pd.read_csv('boston_house_prices.csv', header=1)
```

[3]:
```
df.head()
```

[3]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        B  LSTAT  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

[4]:
```
df.columns
```

[4]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

0.2.2 Analysing the data with info() method. Since the method already shows the shape of 506x14 and all columns with 506 non-null rows (or no missing data), there is no need for any other method.

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

0.2.3 Scaling the data

[6]:
```
scaler = StandardScaler().fit(df).transform(df)
df_scaled = pd.DataFrame(scaler, columns = df.columns)
df_scaled
```

[6]:
```
         CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
0   -0.419782  0.284830 -1.287909 -0.272599 -0.144217  0.413672 -0.120013
1   -0.417339 -0.487722 -0.593381 -0.272599 -0.740262  0.194274  0.367166
2   -0.417342 -0.487722 -0.593381 -0.272599 -0.740262  1.282714 -0.265812
3   -0.416750 -0.487722 -1.306878 -0.272599 -0.835284  1.016303 -0.809889
4   -0.412482 -0.487722 -1.306878 -0.272599 -0.835284  1.228577 -0.511180
..        ...       ...       ...       ...       ...       ...       ...
501 -0.413229 -0.487722  0.115738 -0.272599  0.158124  0.439316  0.018673
```

```
502 -0.415249 -0.487722  0.115738 -0.272599  0.158124 -0.234548  0.288933
503 -0.413447 -0.487722  0.115738 -0.272599  0.158124  0.984960  0.797449
504 -0.407764 -0.487722  0.115738 -0.272599  0.158124  0.725672  0.736996
505 -0.415000 -0.487722  0.115738 -0.272599  0.158124 -0.362767  0.434732

          DIS       RAD       TAX   PTRATIO         B     LSTAT      MEDV
0    0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562  0.159686
1    0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439 -0.101524
2    0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727  1.324247
3    1.077737 -0.752922 -1.106115  0.113032  0.416163 -1.361517  1.182758
4    1.077737 -0.752922 -1.106115  0.113032  0.441052 -1.026501  1.487503
..        ...       ...       ...       ...       ...       ...       ...
501 -0.625796 -0.982843 -0.803212  1.176466  0.387217 -0.418147 -0.014454
502 -0.716639 -0.982843 -0.803212  1.176466  0.441052 -0.500850 -0.210362
503 -0.773684 -0.982843 -0.803212  1.176466  0.441052 -0.983048  0.148802
504 -0.668437 -0.982843 -0.803212  1.176466  0.403225 -0.865302 -0.057989
505 -0.613246 -0.982843 -0.803212  1.176466  0.441052 -0.669058 -1.157248

[506 rows x 14 columns]
```

0.2.4 Creating the target

[7]:
```python
y = df_scaled['MEDV']
y
```

[7]:
```
0      0.159686
1     -0.101524
2      1.324247
3      1.182758
4      1.487503
         ...
501   -0.014454
502   -0.210362
503    0.148802
504   -0.057989
505   -1.157248
Name: MEDV, Length: 506, dtype: float64
```

0.3 Task 2 : PCA

[8]:
```python
X = df_scaled.drop('MEDV', axis=1)
X
```

[8]:
```
         CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
0   -0.419782  0.284830 -1.287909 -0.272599 -0.144217  0.413672 -0.120013
1   -0.417339 -0.487722 -0.593381 -0.272599 -0.740262  0.194274  0.367166
2   -0.417342 -0.487722 -0.593381 -0.272599 -0.740262  1.282714 -0.265812
```

```
3   -0.416750 -0.487722 -1.306878 -0.272599 -0.835284  1.016303 -0.809889
4   -0.412482 -0.487722 -1.306878 -0.272599 -0.835284  1.228577 -0.511180
..        ...       ...       ...       ...       ...       ...       ...
501 -0.413229 -0.487722  0.115738 -0.272599  0.158124  0.439316  0.018673
502 -0.415249 -0.487722  0.115738 -0.272599  0.158124 -0.234548  0.288933
503 -0.413447 -0.487722  0.115738 -0.272599  0.158124  0.984960  0.797449
504 -0.407764 -0.487722  0.115738 -0.272599  0.158124  0.725672  0.736996
505 -0.415000 -0.487722  0.115738 -0.272599  0.158124 -0.362767  0.434732

          DIS       RAD       TAX   PTRATIO         B     LSTAT
0    0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562
1    0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439
2    0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727
3    1.077737 -0.752922 -1.106115  0.113032  0.416163 -1.361517
4    1.077737 -0.752922 -1.106115  0.113032  0.441052 -1.026501
..        ...       ...       ...       ...       ...       ...
501 -0.625796 -0.982843 -0.803212  1.176466  0.387217 -0.418147
502 -0.716639 -0.982843 -0.803212  1.176466  0.441052 -0.500850
503 -0.773684 -0.982843 -0.803212  1.176466  0.441052 -0.983048
504 -0.668437 -0.982843 -0.803212  1.176466  0.403225 -0.865302
505 -0.613246 -0.982843 -0.803212  1.176466  0.441052 -0.669058

[506 rows x 13 columns]
```

0.3.1 Covariance Matrix

[9]:
```python
def cov(data):
    length = data.shape[0]
    covmat = (data.T.dot(data))/length
    return covmat
covamat = cov(X)
covamat = np.round(covamat,3)
covamat
cov_mat = np.round(np.cov(X.T),3)
cov_mat = pd.DataFrame(cov_mat, columns = X.columns)
print(covamat)
print(cov_mat)
```

```
          CRIM     ZN  INDUS   CHAS    NOX     RM    AGE    DIS    RAD    TAX  \
CRIM     1.000 -0.200  0.407 -0.056  0.421 -0.219  0.353 -0.380  0.626  0.583
ZN      -0.200  1.000 -0.534 -0.043 -0.517  0.312 -0.570  0.664 -0.312 -0.315
INDUS    0.407 -0.534  1.000  0.063  0.764 -0.392  0.645 -0.708  0.595  0.721
CHAS    -0.056 -0.043  0.063  1.000  0.091  0.091  0.087 -0.099 -0.007 -0.036
NOX      0.421 -0.517  0.764  0.091  1.000 -0.302  0.731 -0.769  0.611  0.668
RM      -0.219  0.312 -0.392  0.091 -0.302  1.000 -0.240  0.205 -0.210 -0.292
AGE      0.353 -0.570  0.645  0.087  0.731 -0.240  1.000 -0.748  0.456  0.506
DIS     -0.380  0.664 -0.708 -0.099 -0.769  0.205 -0.748  1.000 -0.495 -0.534
RAD      0.626 -0.312  0.595 -0.007  0.611 -0.210  0.456 -0.495  1.000  0.910
```

```
TAX       0.583 -0.315  0.721 -0.036  0.668 -0.292  0.506 -0.534  0.910  1.000
PTRATIO   0.290 -0.392  0.383 -0.122  0.189 -0.356  0.262 -0.232  0.465  0.461
B        -0.385  0.176 -0.357  0.049 -0.380  0.128 -0.274  0.292 -0.444 -0.442
LSTAT     0.456 -0.413  0.604 -0.054  0.591 -0.614  0.602 -0.497  0.489  0.544

         PTRATIO      B  LSTAT
CRIM       0.290 -0.385  0.456
ZN        -0.392  0.176 -0.413
INDUS      0.383 -0.357  0.604
CHAS      -0.122  0.049 -0.054
NOX        0.189 -0.380  0.591
RM        -0.356  0.128 -0.614
AGE        0.262 -0.274  0.602
DIS       -0.232  0.292 -0.497
RAD        0.465 -0.444  0.489
TAX        0.461 -0.442  0.544
PTRATIO    1.000 -0.177  0.374
B         -0.177  1.000 -0.366
LSTAT      0.374 -0.366  1.000
     CRIM     ZN  INDUS   CHAS    NOX     RM    AGE    DIS    RAD    TAX  \
0   1.002 -0.201  0.407 -0.056  0.422 -0.220  0.353 -0.380  0.627  0.584
1  -0.201  1.002 -0.535 -0.043 -0.518  0.313 -0.571  0.666 -0.313 -0.315
2   0.407 -0.535  1.002  0.063  0.765 -0.392  0.646 -0.709  0.596  0.722
3  -0.056 -0.043  0.063  1.002  0.091  0.091  0.087 -0.099 -0.007 -0.036
4   0.422 -0.518  0.765  0.091  1.002 -0.303  0.733 -0.771  0.613  0.669
5  -0.220  0.313 -0.392  0.091 -0.303  1.002 -0.241  0.206 -0.210 -0.293
6   0.353 -0.571  0.646  0.087  0.733 -0.241  1.002 -0.749  0.457  0.507
7  -0.380  0.666 -0.709 -0.099 -0.771  0.206 -0.749  1.002 -0.496 -0.535
8   0.627 -0.313  0.596 -0.007  0.613 -0.210  0.457 -0.496  1.002  0.912
9   0.584 -0.315  0.722 -0.036  0.669 -0.293  0.507 -0.535  0.912  1.002
10  0.291 -0.392  0.384 -0.122  0.189 -0.356  0.262 -0.233  0.466  0.462
11 -0.386  0.176 -0.358  0.049 -0.381  0.128 -0.274  0.292 -0.445 -0.443
12  0.457 -0.414  0.605 -0.054  0.592 -0.615  0.604 -0.498  0.490  0.545

    PTRATIO      B  LSTAT
0     0.291 -0.386  0.457
1    -0.392  0.176 -0.414
2     0.384 -0.358  0.605
3    -0.122  0.049 -0.054
4     0.189 -0.381  0.592
5    -0.356  0.128 -0.615
6     0.262 -0.274  0.604
7    -0.233  0.292 -0.498
8     0.466 -0.445  0.490
9     0.462 -0.443  0.545
10    1.002 -0.178  0.375
11   -0.178  1.002 -0.367
12    0.375 -0.367  1.002
```

0.3.2 Eigenvalues and Eigenvectors

[10]:
```
eigenvalues, eigenvectors = np.linalg.eig(covamat)
pd.DataFrame(eigenvectors)
```

[10]:
```
           0         1         2         3         4         5         6  \
0  -0.251026  0.314873 -0.247820 -0.062685  0.082093  0.220388 -0.776254
1   0.256366  0.322867 -0.296386 -0.129659  0.320454  0.322801  0.273355
2  -0.346718 -0.112544  0.016300 -0.017101 -0.007122  0.076775  0.340427
3  -0.005031 -0.455616 -0.287808 -0.815927  0.085769 -0.169134 -0.074071
4  -0.342798 -0.219312 -0.120318  0.127873  0.136537  0.153525  0.200882
5   0.189231 -0.150338 -0.593871  0.280386 -0.423442 -0.058247 -0.064629
6  -0.313646 -0.312063  0.018090  0.174683  0.015716  0.071257 -0.118913
7   0.321494  0.348899  0.049669 -0.215857  0.098050 -0.024907  0.104006
8  -0.319783  0.270938 -0.287478 -0.133740 -0.204452  0.142150  0.137972
9  -0.338434  0.239347 -0.220919 -0.103768 -0.130672  0.191250  0.316333
10 -0.204959  0.306576  0.323586 -0.280776 -0.583893 -0.273920 -0.004062
11  0.202995 -0.237188  0.300714 -0.169867 -0.345984  0.803336 -0.069465
12 -0.309773  0.074872  0.266651 -0.069344  0.394771  0.053878 -0.088861

           7         8         9        10        11        12
0  -0.156680 -0.045941 -0.260458  0.086768  0.110751 -0.018015
1   0.404646  0.079039 -0.357883 -0.067947 -0.264026 -0.267152
2  -0.172194  0.252881 -0.643667 -0.119137  0.299646  0.364471
3   0.024389 -0.036642  0.013023 -0.003305 -0.014490  0.006381
4  -0.079469 -0.041032  0.018405  0.806345 -0.103907 -0.227314
5   0.325627 -0.044587 -0.046261  0.151634 -0.052020  0.433080
6   0.600996  0.038175  0.065573 -0.213702  0.457320 -0.363556
7   0.122152  0.022662  0.152278  0.383980  0.699188  0.172836
8  -0.079201  0.632915  0.470972 -0.109273 -0.038774 -0.025416
9  -0.081805 -0.720372  0.176370 -0.213820  0.106128  0.034164
10  0.317337 -0.023577 -0.255519  0.212179 -0.174059 -0.150382
11  0.004548  0.004021  0.045798  0.040854 -0.018400  0.096070
12  0.423455 -0.024877  0.198317  0.054846 -0.270402  0.600411
```

0.3.3 Kaiser's Stopping Rule

[11]:
```
indexes = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[indexes]
eigenvectors = eigenvectors[:, indexes]
sorted_eig_pairs = [(np.round(np.abs(eigenvalues[i]),3), eigenvectors[:,i]) for␣
 ↪i in range(len(eigenvalues))]

sorted_eigenvalues = []
for i in range(0, len(sorted_eig_pairs)):
    sorted_eigenvalues.append(sorted_eig_pairs[i][0])
```

```
print('Total Variance:', round(sum(sorted_eigenvalues),0))
```

```
Total Variance: 13.0
```

[12]:
```
optimal = [sorted_eig_pairs[i][0] for i in range(0, len(sorted_eig_pairs)) if␣
 ↪sorted_eig_pairs[i][0] >= 1]
optimal
```

[12]:
```
[6.127, 1.434, 1.243]
```

0.3.4 Projection Matrix: Construction to DataFrame

[13]:
```
k = 3
projection_matrix = np.array([list(np.hstack(i[1].reshape(13,1))) for i in␣
 ↪sorted_eig_pairs[:]])
projection_matrix = projection_matrix[:k]
print('Matrix Dimension:', projection_matrix.shape)
```

```
Matrix Dimension: (3, 13)
```

[14]:
```
pmatrix_df = pd.DataFrame(projection_matrix, columns = [str(i+1) for i in␣
 ↪range(len(eigenvectors))])
pmatrix_df
```

[14]:
```
          1         2         3         4         5         6         7  \
0 -0.251026  0.256366 -0.346718 -0.005031 -0.342798  0.189231 -0.313646
1  0.314873  0.322867 -0.112544 -0.455616 -0.219312 -0.150338 -0.312063
2 -0.247820 -0.296386  0.016300 -0.287808 -0.120318 -0.593871  0.018090

          8         9        10        11        12        13
0  0.321494 -0.319783 -0.338434 -0.204959  0.202995 -0.309773
1  0.348899  0.270938  0.239347  0.306576 -0.237188  0.074872
2  0.049669 -0.287478 -0.220919  0.323586  0.300714  0.266651
```

0.3.5 Transforming the DataFrame

[15]:
```
pca_df = pd.DataFrame((X).dot(projection_matrix.T))
pca_df.columns = ['PC' + str(i+1) for i in range(k)]
pca_df
```

[15]:
```
          PC1       PC2       PC3
0    2.098392 -0.773534 -0.342918
1    1.457199 -0.590802  0.695632
2    2.074525 -0.599952 -0.166628
3    2.611414  0.007009  0.100295
4    2.458096 -0.097595  0.075394
..        ...       ...       ...
501  0.314957 -0.722311  0.862257
```

```
502  0.110524 -0.756634  1.257458
503  0.312379 -1.154099  0.410569
504  0.270531 -1.039956  0.587300
505  0.125816 -0.759298  1.296462

[506 rows x 3 columns]
```

0.3.6 Explained Variance Ratio

[16]:
```python
expl = [((eigenvalues[i])/np.sum(eigenvalues)) for i in range(k)]
expl
```

[16]:
```
[0.47133242769373673, 0.11027026383021919, 0.0955950674492598]
```

[17]:
```python
expl_ratio = sum(expl)
expl_ratio
```

[17]:
```
0.6771977589732158
```

[18]:
```python
print(np.round(expl_ratio.real*100), '% of the information in the original data␣
↪was preserved')
```

```
68.0 % of the information in the original data was preserved
```

0.4 Task 3 : Linear Regression

0.4.1 Reassigning features and target with transformed data and training the model

[19]:
```python
X = pca_df
y = y.values.reshape(-1,1)
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 5)
```

[20]:
```python
LR = LinearRegression().fit(X_train,y_train)
y_pred = LR.predict(X_test)
```

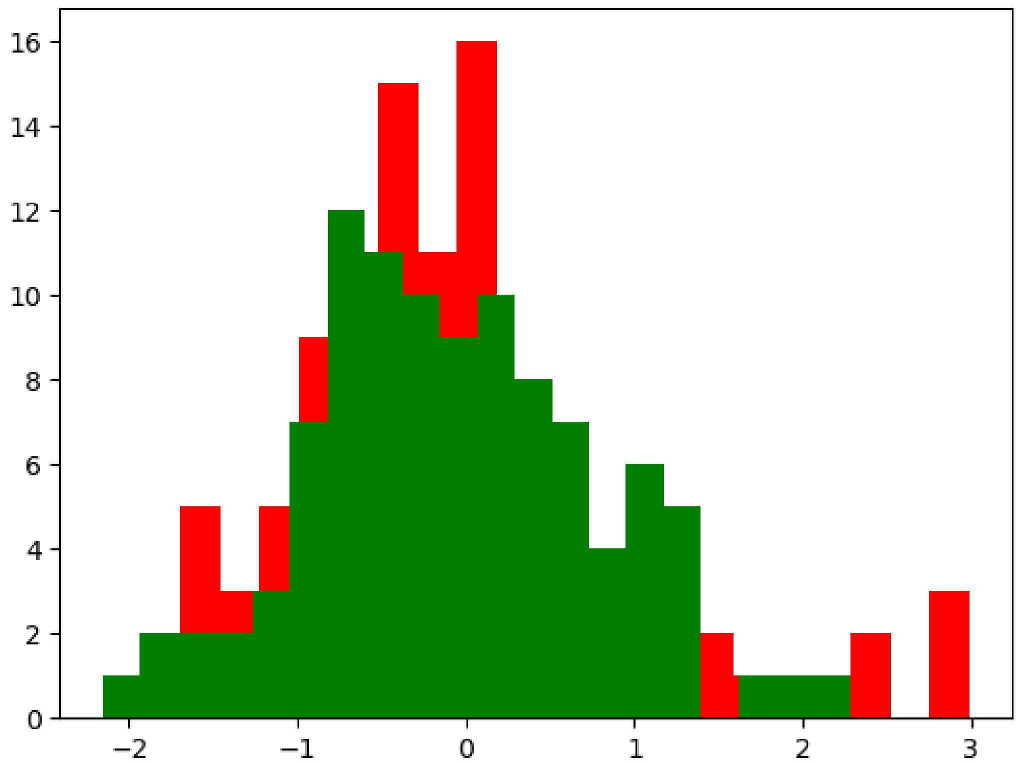
0.4.2 Finding the coefficients

[21]:
```python
coeff_df = pd.DataFrame(LR.coef_, columns = X.columns)
coeff_df
```

[21]:
```
        PC1       PC2       PC3
0  0.252321 -0.236588 -0.375265
```

0.4.3 Plotting prediction compared to test

[22]:
```
plt.hist(y_test, bins= 20, color='r')
plt.hist(y_pred, bins= 20, color = 'g')
plt.show()
```



0.4.4 Calculating the metrics

[23]:
```
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE', mean_squared_error(y_test, y_pred))
print('RMSE:', np.round(np.sqrt(mean_squared_error(y_test, y_pred)),3))
print('R2:', np.round(r2_score(y_true = y_test,  y_pred = y_pred),2)*100, '%')
print('RSS:', np.round(np.sum(np.square(y_test - y_pred)),3))
```

```
MAE: 0.39186048093515824
MSE 0.27730616080517007
RMSE: 0.527
R2: 70.0 %
RSS: 28.285
```

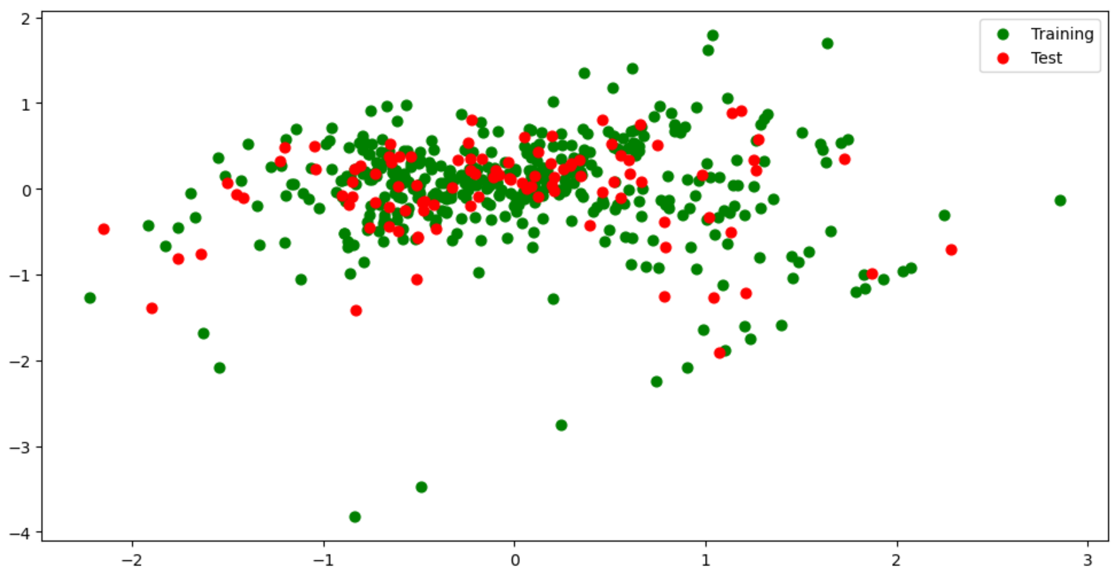
0.5 Task 4: Interpreting the metrics

By analising the error metrics, it is possible to state a relatively high margin of error in the prediction. It would be desirable to decrease the error.

Meanwhile, the R squared found is relatively good, showing that there is some fitness of the data into a linear regression, but pointing to the possibility of exploring different procedures to find better results.

[24]:
```
plt.figure(figsize=(12,6))
plt.scatter(LR.predict(X_train), LR.predict(X_train) - y_train, c='g', s=40,␣
↪label='Training')
plt.scatter(LR.predict(X_test), LR.predict(X_test) - y_test, c='r', s=40, label=␣
↪'Test')
plt.legend()
```

[24]: <matplotlib.legend.Legend at 0x1cfc634cc10>



[30]:
```
#model = CatBoostRegressor(iterations = 545, learning_rate = 0.03,␣
↪loss_function='RMSE')
#model.fit(X_train,y_train, eval_set=(X_test,y_test),)
# commenting to hide the large results
```

[26]:
```
model.score(X,y)
```

[26]: 0.8876999538834502

[27]:
```
pred = model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, pred)))
```

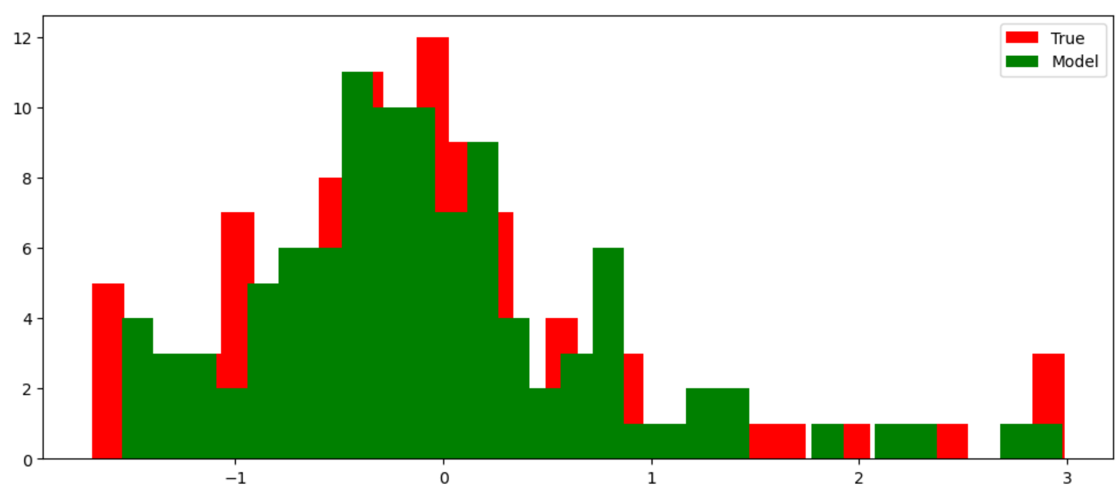
```
r2 = r2_score(y_test, pred)
print('RMSE:', np.round(rmse, 4))
print("R2:", np.round(r2*100,2), '%')
```

```
RMSE: 0.4675
R2: 76.43 %
```

It was possible to decrease the RMSE to approximately 0.47, which is still a high number. This indicates that this model is not very effective in fitting properly the predicted values into the observed values. Most likely it is best to try another approach for this data, instead of using PCA and Linear Regression. The R squared also showed some improvement with better fitness of the data into a linear regression. This means that Linear Regression might still be a way to relate to this data, but from a another approach.

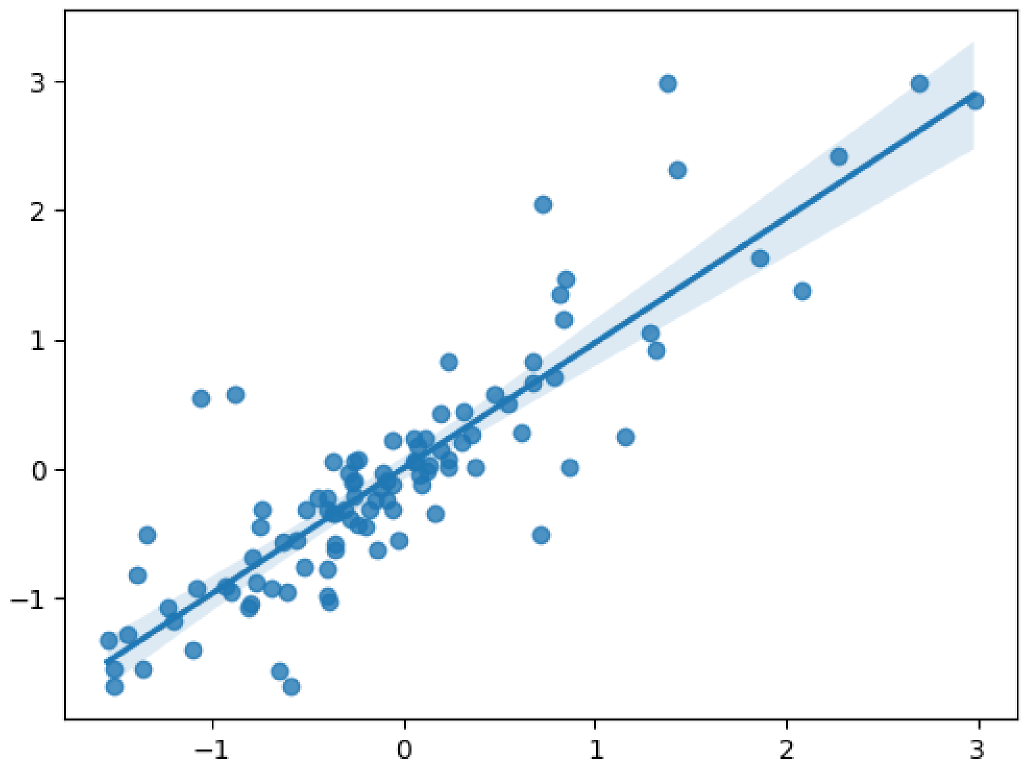
[28]:
```
plt.figure(figsize=(12,5))
plt.hist(y_test, bins=30, color='r', label='True')
plt.hist(pred, bins=30, color='g', label='Model')
plt.legend()
```

[28]:
```
<matplotlib.legend.Legend at 0x1cfc7de9e50>
```



[29]:
```
sns.regplot(x = pred, y = y_test)
```

[29]:
```
<Axes: >
```


3
2
1
0
−1
−1
0
1
2
3

```
[ ]:
```