assignment-5-bhs-atharva-pingale

April 9, 2024

1. Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset

Reference https://inside-machinelearning.com/en/how-to-do-linear-regression-with-keras/

[ ]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import pandas as pd
```

[ ]:
```
data=pd.read_csv("/content/housing (1).csv")
data
```

[ ]:
```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       …    …     …     …     …     …    …       …  …    …
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       …       …     …    …
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
```

```
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

[ ]:
```
data.dtypes
```

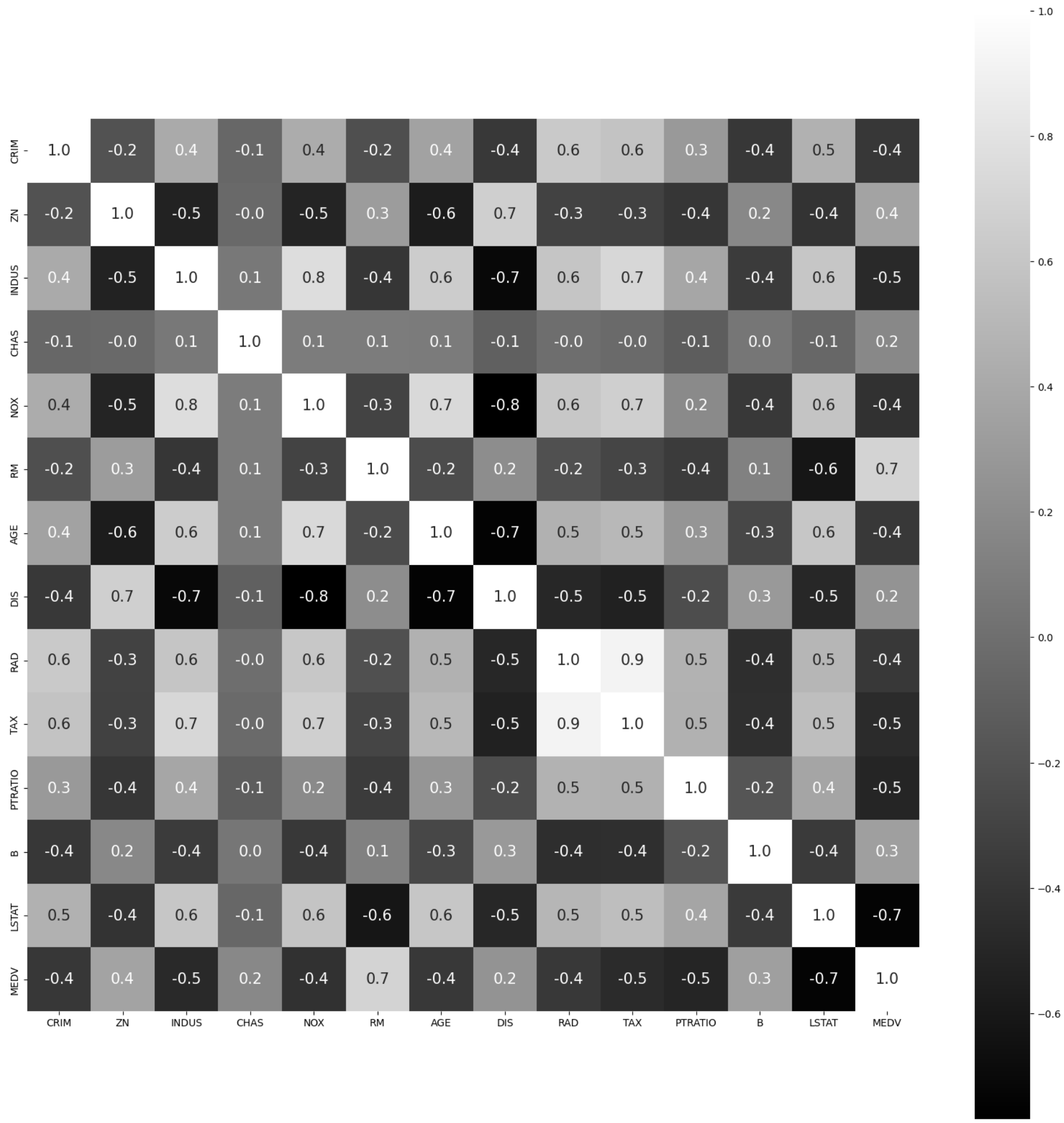
[ ]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX          int64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

[ ]:
```
# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

[ ]:
```
(14, 14)
```

[ ]:
```
# Plotting the heatmap of correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True,
  annot_kws={'size':15}, cmap='gray')
```

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




```
[ ]: X =data.drop(['MEDV'], axis = 1)# data['area']#
     y = data['MEDV']
```

```
[ ]: X
```

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

```
..        ...    ...     ...    ...     ...     ...    ...      ...  ...  ...
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     PTRATIO       B  LSTAT
0       15.3  396.90   4.98
1       17.8  396.90   9.14
2       17.8  392.83   4.03
3       18.7  394.63   2.94
4       18.7  396.90   5.33
..       ...     ...    ...
501     21.0  391.99   9.67
502     21.0  396.90   9.08
503     21.0  396.90   5.64
504     21.0  393.45   6.48
505     21.0  396.90   7.88

[506 rows x 13 columns]
```

[ ]:
```
X.columns
```

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

[ ]:
```
y
```

[ ]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: MEDV, Length: 506, dtype: float64
```

[ ]:
```
y.describe()
```

[ ]:
```
count    506.000000
mean      22.532806
```

```
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: MEDV, dtype: float64
```

[ ]:
```
# Splitting to training and testing data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3,␣
↪random_state = 4)
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
# Instantiate the scaler and fit to training dataset, X_train
scaler = MinMaxScaler()
scaler.fit(X_train)

# Replace unscaled values with scaled values
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[ ]:

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
```

[ ]:
```
model = Sequential()L_Assignment2
model.add(Dense(64, input_dim =13, activation = 'relu'))
model.add(Dropout(0.15))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(40, activation = 'relu'))
model.add(Dropout(0.15))
model.add(Dense(54, activation = 'relu'))
model.add(Dropout(0.18))
model.add(Dense(1))
```

[ ]:
```
model.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics =␣
↪['mse', 'mae'])
```

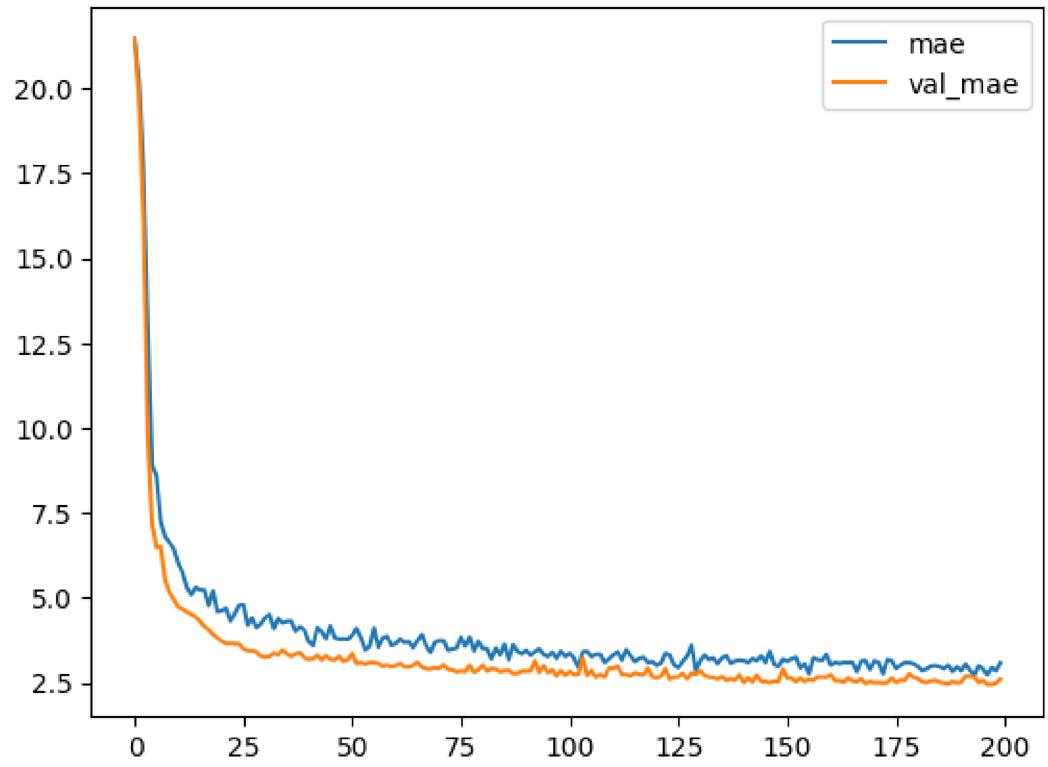
[ ]:
```
history = model.fit(X_train, y_train, validation_split=0.2, epochs=200)
```

[ ]:
```
import matplotlib.pyplot as plt
```

```
#plot the loss and validation loss of the dataset
plt.plot(history.history['mae'], label='mae')
plt.plot(history.history['val_mae'], label='val_mae')

plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7f91acbf7f70>



[ ]:

```
scores = model.evaluate(X_test, y_test, verbose = 0)

print('Mean Squared Error : ', scores[1])
print('Mean Absolute Error : ', scores[2])
```

```
Mean Squared Error :  17.62657928466797
Mean Absolute Error :  2.6688265800476074
```

[ ]:

```
Y_pred = model.predict(X_test)
Y_pred
```

[ ]:
```python
from sklearn.metrics import r2_score

print('r2 score: ', r2_score(y_test,Y_pred))
```

```
r2 score:  0.831195498482862
```

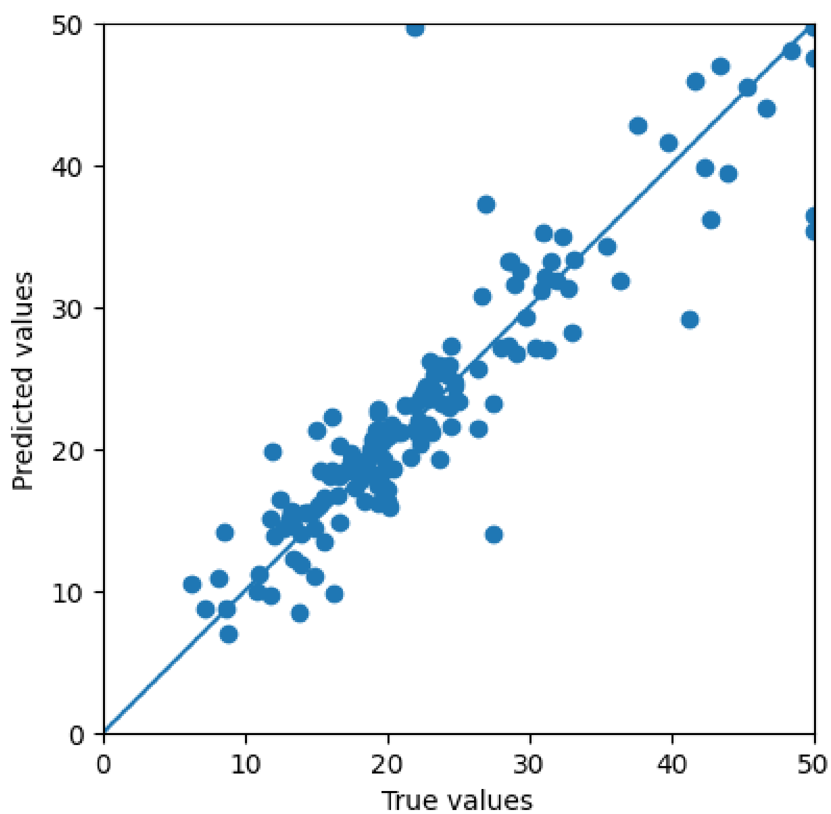
[ ]:
```python
Y_pred = model.predict(X_test)

a = plt.axes(aspect='equal')

plt.xlabel('True values')
plt.ylabel('Predicted values')
plt.xlim([0, 50])
plt.ylim([0, 50])
plt.plot([0, 50], [0,50])
plt.scatter(y_test,Y_pred)
plt.plot()
```

```
5/5 [==============================] - 0s 3ms/step
```

[ ]: []

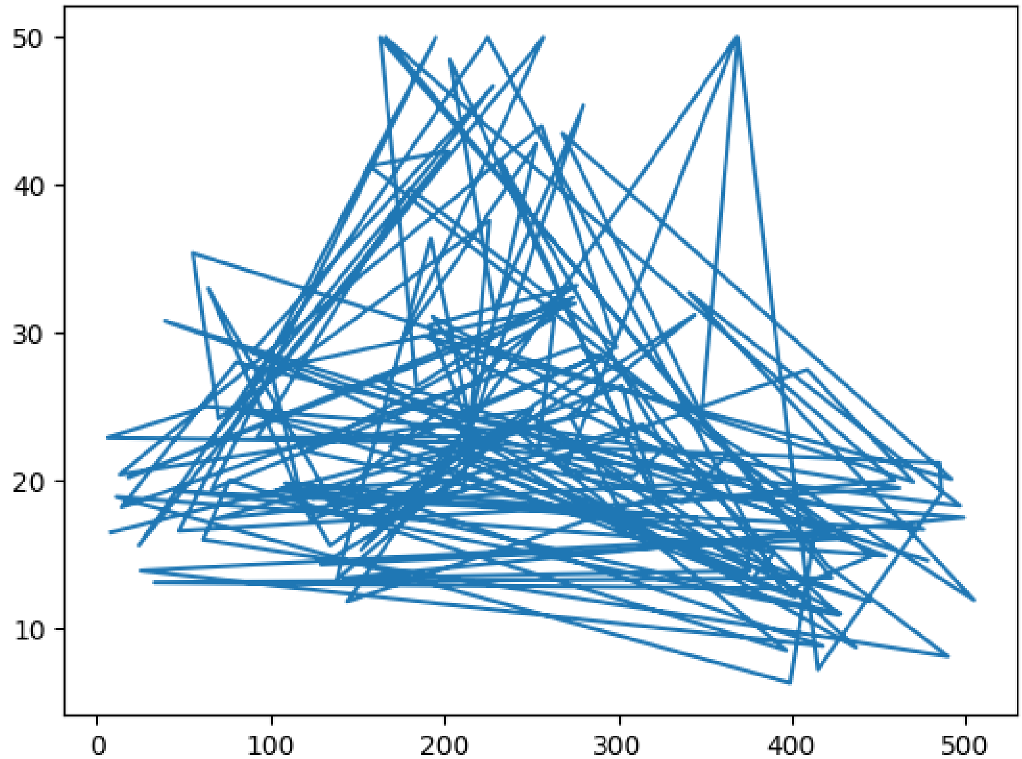


[ ]:
```

```

[ ]:
```
plt.plot(y_test)
```

[ ]: [<matplotlib.lines.Line2D at 0x7f91b0719c70>]



[ ]:
```
plt.plot(Y_pred)
Y_pred
```

[ ]:
```
print(Y_pred[:5])
print(y_test[:5])
y_test.head()
```

```
[[16.747326]
 [24.475298]
 [19.15377 ]
 [16.203077]
 [42.90115 ]]
8      16.5
289    24.8
68     17.4
211    19.3
```

```
226    37.6
Name: MEDV, dtype: float64
```

[ ]:
```
8      16.5
289    24.8
68     17.4
211    19.3
226    37.6
Name: MEDV, dtype: float64
```

Using ML MODEL lm for Linear Regression

[ ]:
```
# Import library for Linear Regression
from sklearn.linear_model import LinearRegression
```

[ ]:
```
# Create a Linear regressor
lm = LinearRegression()

# Train the model using the training sets
lm.fit(X_train, y_train)
```

[ ]:
```
LinearRegression()
```

[ ]:
```
# Value of y intercept
lm.intercept_
```

[ ]:
```
27.380936280008473
```

[ ]:
```
#Converting the coefficient values to a dataframe
coeffcients = pd.DataFrame([X.columns,lm.coef_]).T
coeffcients = coeffcients.rename(columns={0: 'Attribute', 1: 'Coefficients'})
coeffcients
```

[ ]:

	Attribute	Coefficients
0	CRIM	-10.90502
1	ZN	5.56777
2	INDUS	-0.238526
3	CHAS	4.693448
4	NOX	-7.01579
5	RM	15.865749
6	AGE	-0.33478
7	DIS	-17.068774
8	RAD	7.503741
9	TAX	-7.370872
10	PTRATIO	-7.550784
11	B	3.709485
12	LSTAT	-18.452583

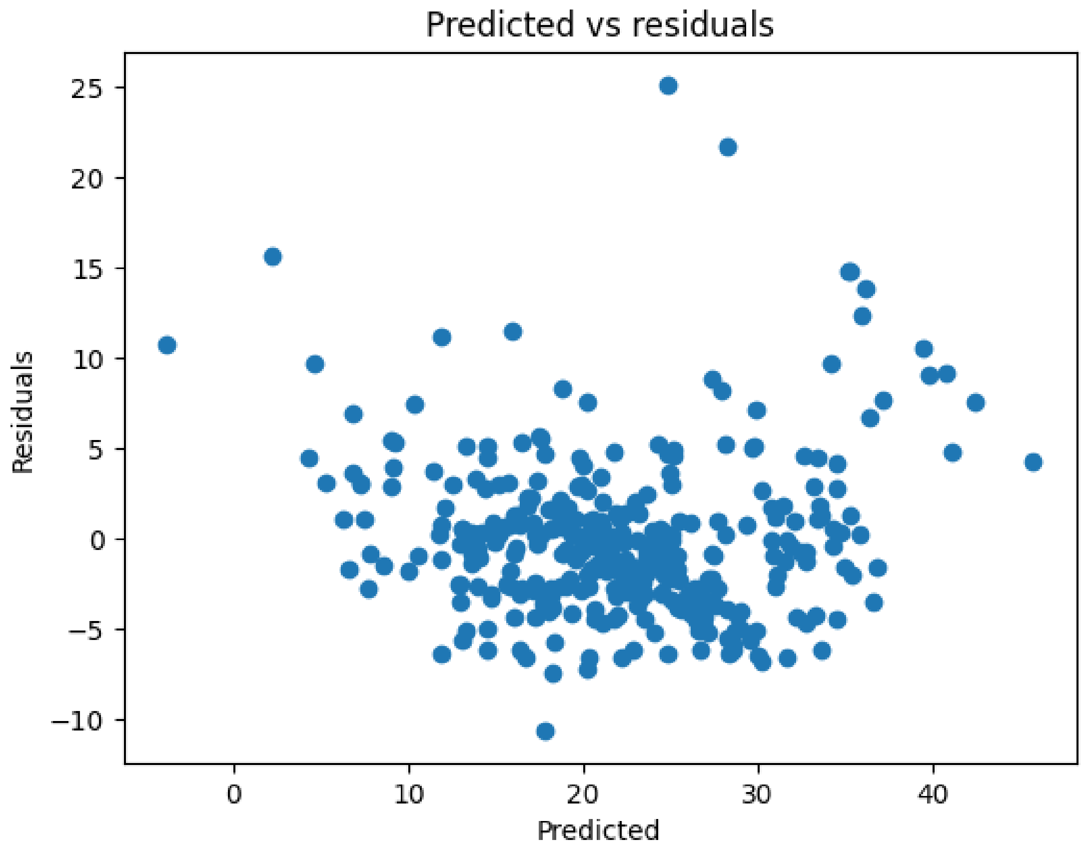
```
[ ]: # Model prediction on train data
y_pred = lm.predict(X_train)
```

```
[ ]: # Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train,␣
  ↪y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.7465991966746854
Adjusted R^2: 0.736910342429894
MAE: 3.089861094971128
MSE: 19.073688703469028
RMSE: 4.367343437774161
```

```
[ ]: # Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

[ ]:
```
# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```




```
[ ]: # Checking Normality of errors
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```

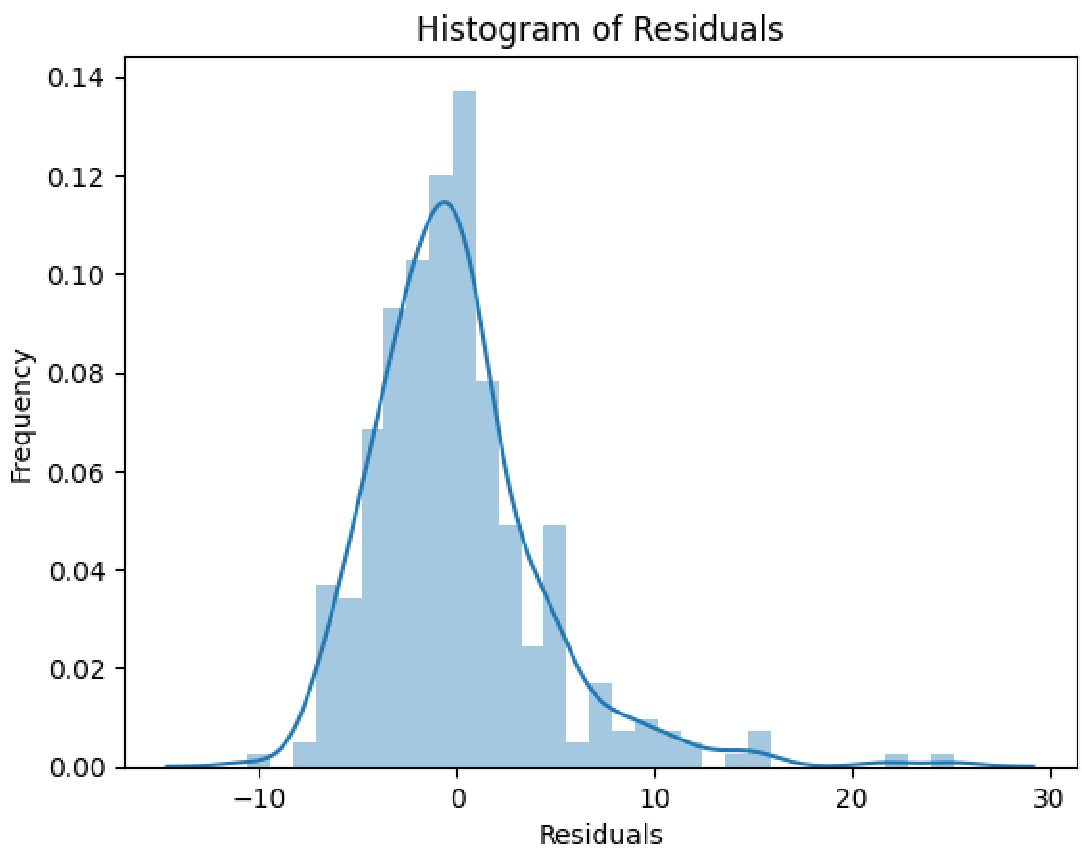
```
<ipython-input-57-c62ec83682b5>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_train-y_pred)
```




```
[ ]: # Predicting Test data with the model
     y_test_pred = lm.predict(X_test)
```

```
[ ]: # Model Evaluation
     acc_linreg = metrics.r2_score(y_test, y_test_pred)
     print('R^2:', acc_linreg)
     print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test,␣
      ↪y_test_pred))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
     print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
     print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
     print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7121818377409196
Adjusted R^2: 0.6850685326005714
MAE: 3.8590055923707407
MSE: 30.05399330712412
RMSE: 5.482152251362973
```