In [2]:
```
#Prac-01
#Linear regression by using Deep Neural network: Implement Boston housin
#price prediction problem by Linear regression using Deep Neural network
#Boston House price prediction dataset.
import pandas as pd
import numpy as np

# Load the Boston housing dataset from the original source
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

# Convert data and target into DataFrame
data_columns = ["CRIM", "ZN", "INDUS", "CHAS", "NOX",
"RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"]
data = pd.DataFrame(data, columns=data_columns)
data['PRICE'] = target

# Display the first few rows of the DataFrame
print(data.head())

# Now, you can continue with the rest of the code to build and train the
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  PRICE
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
```

In [3]:
```
#Check the shape of dataframe
data.shape
```

Out[3]:
```
(506, 14)
```

In [4]:
```
data.columns
```

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

In [5]:
```python
data.dtypes
```

Out[5]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
```

In [6]:
```python
# Identifying the unique number of values in the dataset
data.nunique()
```

Out[6]:
```
CRIM       504
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        356
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      455
PRICE      229
dtype: int64
```

In [7]:
```python
# Check for missing values
data.isnull().sum()
```

Out[7]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

In [8]:
```
# See rows with missing values
data[data.isnull().any(axis=1)]
```

Out[8]:

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE

In [9]:
```
# Viewing the data statistics
data.describe()
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AG
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.57490
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.14886
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.90000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.02500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.50000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.07500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.00000

In [10]:
```
# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

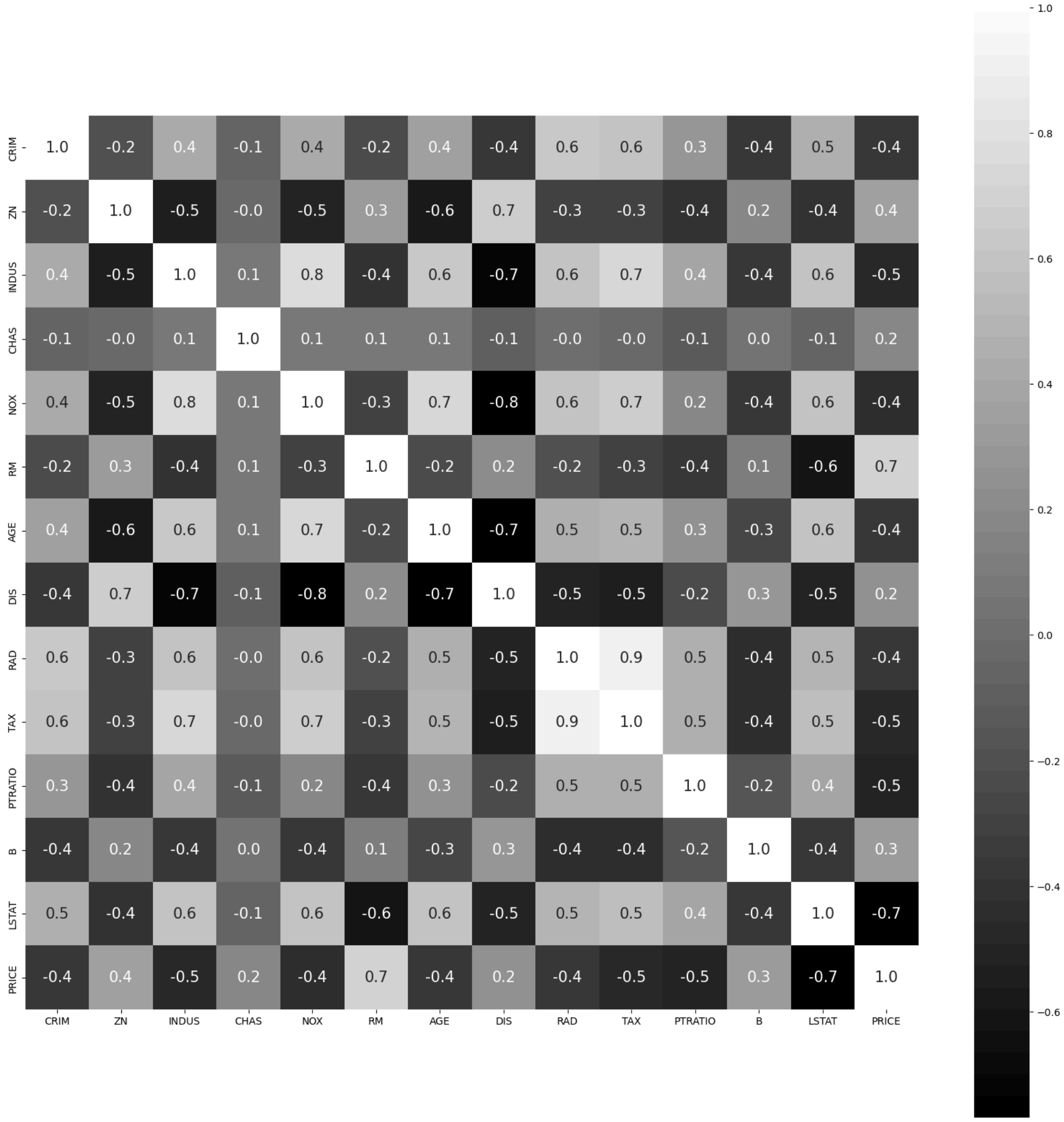
Out[10]: (14, 14)

In [13]:

```
# Plotting the heatmap of correlation between features
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True,
annot_kws={'size':15}, cmap='gray')
```

Out[13]: <Axes: >



In [14]:

```
# Spliting target variable and independent variables
X = data.drop(['PRICE'], axis = 1)
y = data['PRICE']
```

In [15]:

```
# 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)
```

In [16]:
```
# Import library for Linear Regression
from sklearn.linear_model import LinearRegression
```

In [17]:
```
# Create a Linear regressor
lm = LinearRegression()
```

In [18]:
```
# Train the model using the training sets
lm.fit(X_train, y_train)
```

Out[18]:
```
▾ LinearRegression
LinearRegression()
```

In [19]:
```
# Value of y intercept
lm.intercept_
```

Out[19]: 36.35704137659398

In [21]:
```
#Converting the coefficient values to a dataframe
coeffcients = pd.DataFrame([X_train.columns,lm.coef_]).T
coeffcients = coeffcients.rename(columns={0: 'Attribute', 1: 'Coefficien
coeffcients
```

Out[21]:

	Attribute	Coefficients
0	CRIM	-0.12257
1	ZN	0.055678
2	INDUS	-0.008834
3	CHAS	4.693448
4	NOX	-14.435783
5	RM	3.28008
6	AGE	-0.003448
7	DIS	-1.552144
8	RAD	0.32625
9	TAX	-0.014067
10	PTRATIO	-0.803275
11	B	0.009354
12	LSTAT	-0.523478

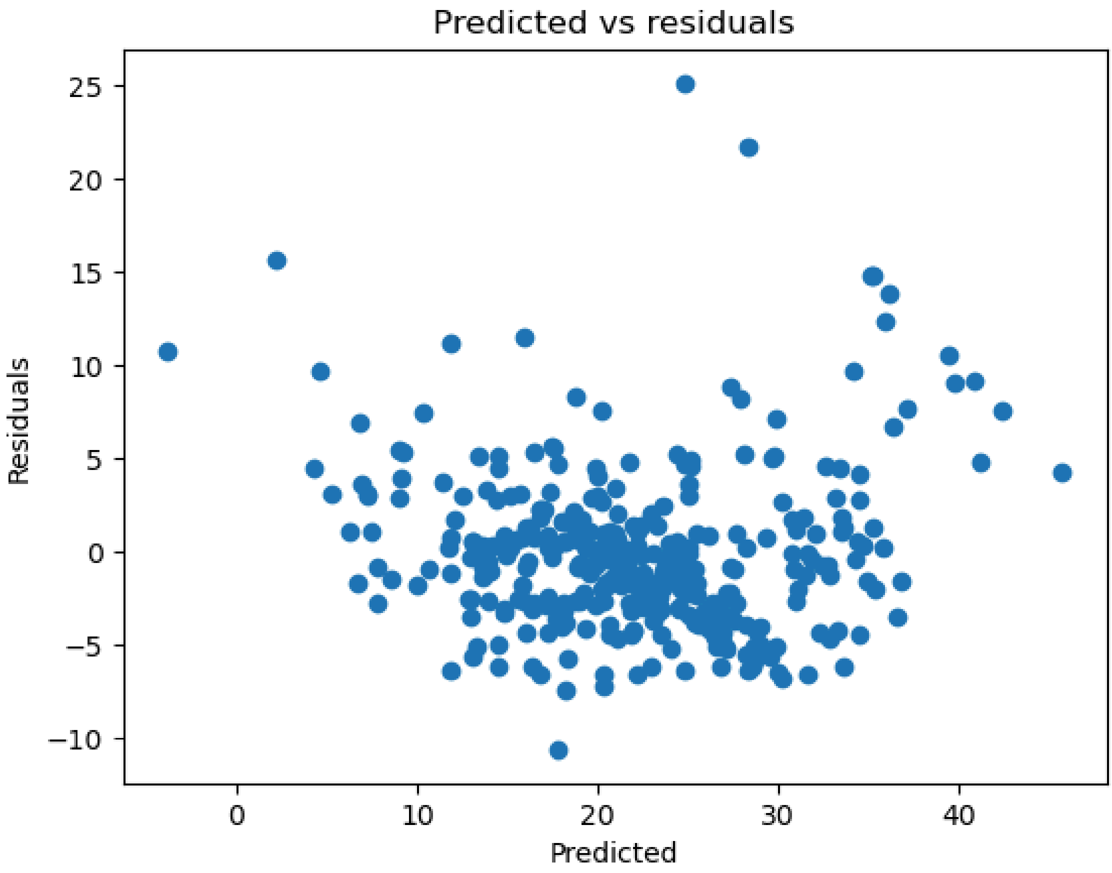
In [23]:
```
from sklearn import metrics

# 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: 3.0898610949711265
MSE: 19.07368870346903
RMSE: 4.367343437774162
```

In [24]:
```
# 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()
```

In [25]:
```
# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [26]:
```
# Checking Normality of errors
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```

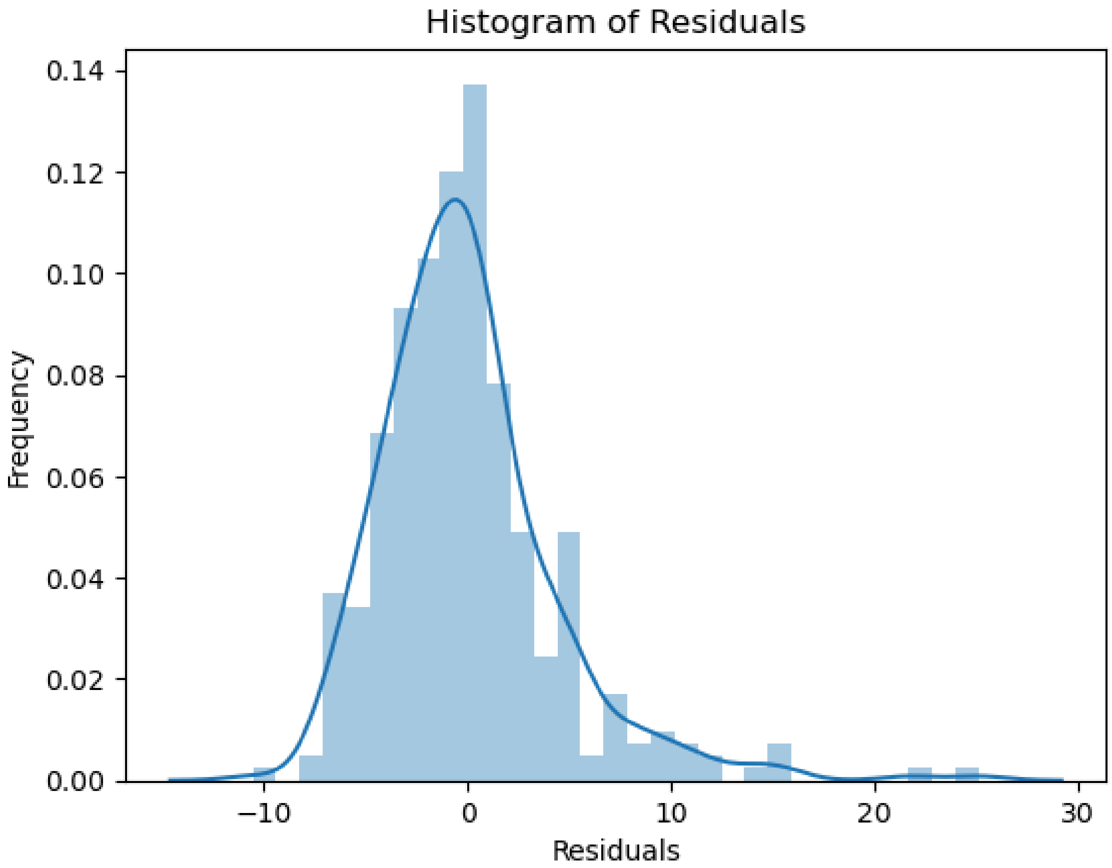
```
C:\Users\Ekata\AppData\Local\Temp\ipykernel_9316\3326403628.py:2: UserW
arning:

`distplot` is a deprecated function and will be removed in seaborn v0.1
4.0.

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

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

  sns.distplot(y_train-y_pred)
```



In [27]:
```
# 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: 0.7121818377409193
Adjusted R^2: 0.6850685326005711
MAE: 3.859005592370745
MSE: 30.053993307124163
RMSE: 5.482152251362978
```

In [ ]: