474CIS- Lab Project Report

Title:

Predict the median value of owner occupied homes

By

Name

Sara Ali [illegible]

[illegible]

[illegible]

ID@kku.edu.sa

Submitted to

Lab Techer Name

Course Coordinator : Dr. Anandhi

Mrs.Aisha Ali Al Maalawi

King Khaled University

College of Computer Science

Department of Information Systems

Semester III - Year 2022

Project Aim:

The aim of the project is to predict the median value of owner occupied homes using various regression techniques on the Boston housing dataset.

Description:

The project aims to predict the median value of owner-occupied homes using various statistical and machine learning techniques. The dataset used for this project includes various features such as crime rate, average number of rooms per dwelling, pupil-teacher ratio, and more. By analyzing these features and their impact on the median value of homes, the project aims to develop a predictive model that can accurately estimate the value of a home based on its characteristics. This information can be useful for real estate agents, homeowners, and potential buyers looking to make informed decisions about buying or selling properties.

Models Used & Its Description:

User model (Liner Regression)

A collection of statistical techniques called regression analysis is used to estimate the associations between a dependent variable and one or more independent variables. It can be used to model how strongly variables will be related in the future and to gauge the strength of that relationship. It can be used to simulate the future relationship between variables and gauge how strongly the relationships between them are currently.

decision tree :is a popular method of creating and visualizing predictive models and algorithms

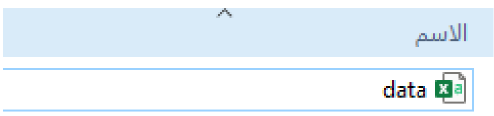
data pre-processing:

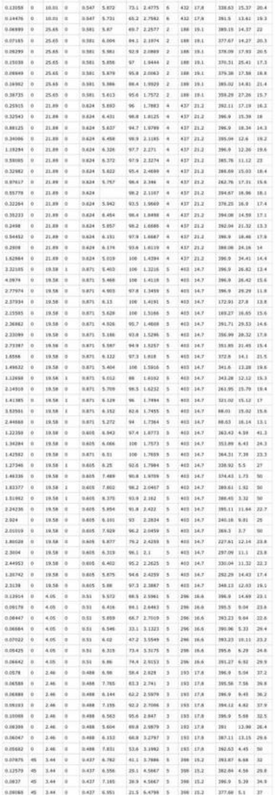
cleaning : is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

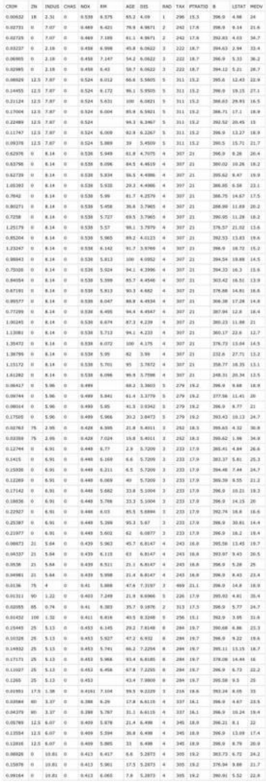
- handle with(missing values) as they refer to several different things. Perhaps the field was not applicable, the event did not occur, or the data was not available.

Dataset Used & Its Description:

This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. It was obtained from the StatLib archive (http://lib.stat.cmu.edu/datasets/boston), and has been used extensively throughout the literature to benchmark algorithms. However, these comparisons were primarily done outside of Delve and are thus somewhat suspect. The dataset is small with only 506 cases.







Results & Discussion:

1) Preparing the given Dataset

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
```

```
In [ ]: # 1) Preparing the given Dataset
```

```
In [2]: df = pd.read_csv('data.csv')
        df
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
506	0.98765	0.0	12.50	0	0.561	6.980	89.0	2.0980	3	320	23.0	396.00	12.00	12.0
507	0.23456	0.0	12.50	0	0.561	6.980	76.0	2.6540	3	320	23.0	343.00	25.00	32.0
508	0.44433	0.0	12.50	0	0.561	6.123	98.0	2.9870	3	320	23.0	343.00	21.00	54.0
509	0.77763	0.0	12.70	0	0.561	6.222	34.0	2.5430	3	329	23.0	343.00	76.00	67.0
510	0.65432	0.0	12.80	0	0.561	6.760	67.0	2.9870	3	345	23.0	321.00	45.00	24.0

511 rows × 14 columns

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

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         5
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```


```
dtype: int64
```


In [5]:
```
df.describe()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	511.000000	511.000000	511.000000	511.000000	511.000000	506.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000
mean	3.584139	11.252446	11.151096	0.068493	0.554757	6.287589	68.616243	3.783876	9.485323	407.440313	18.500000	356.600900	12.879550	22.682192
std	8.564433	23.234838	6.828175	0.252838	0.115310	0.703802	28.099130	2.098631	8.688469	167.903532	2.200348	90.882679	7.797416	9.484262
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082325	0.000000	5.190000	0.000000	0.449000	5.885500	45.050000	2.100350	4.000000	279.500000	17.400000	374.710000	7.065000	17.050000
50%	0.261690	0.000000	9.690000	0.000000	0.538000	6.209000	77.300000	3.152300	5.000000	330.000000	19.100000	391.340000	11.450000	21.200000
75%	3.621175	12.500000	18.100000	0.000000	0.624000	6.629750	94.050000	5.118000	24.000000	666.000000	20.200000	396.210000	17.105000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	23.000000	396.900000	76.000000	67.000000

In [17]:
```
df.info()
```

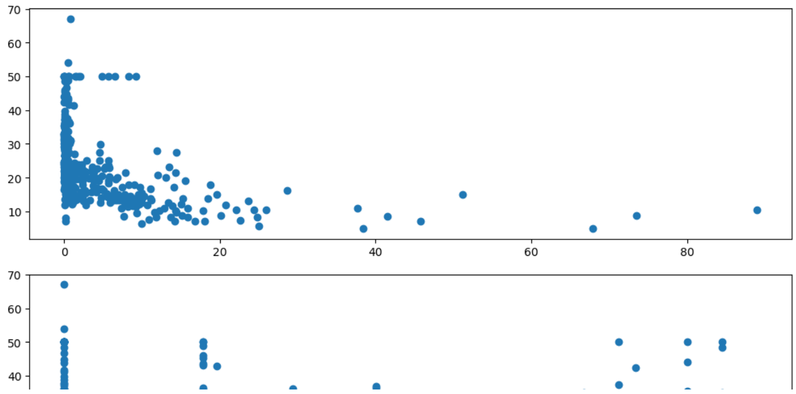
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 511 entries, 0 to 510
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     511 non-null    float64
 1   ZN       511 non-null    float64
 2   INDUS    511 non-null    float64
 3   CHAS     511 non-null    int64  
 4   NOX      511 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      511 non-null    float64
 7   DIS      511 non-null    float64
 8   RAD      511 non-null    int64  
 9   TAX      511 non-null    int64  
 10  PTRATIO  511 non-null    float64
 11  B        511 non-null    float64
 12  LSTAT    511 non-null    float64
 13  MEDV     511 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 56.0 KB
```

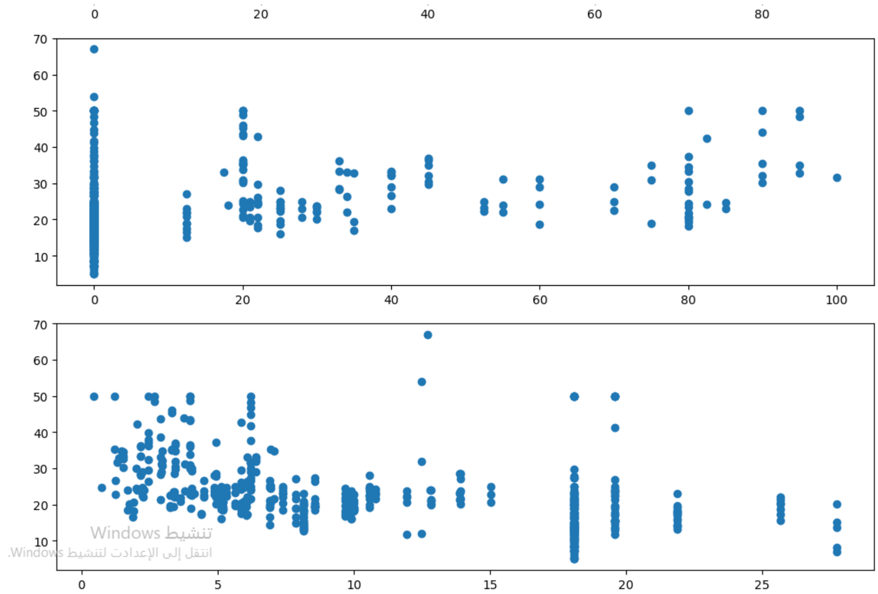
Visualization

In [ ]:
```
# Visualization
```

In [6]:
```
fig,axes = plt.subplots(figsize=(10,10),nrows=3,ncols=1)
col = 0
for ca in axes:
    ca.scatter(df.iloc[:,col],df['MEDV'])
    col+=1

plt.tight_layout()
```

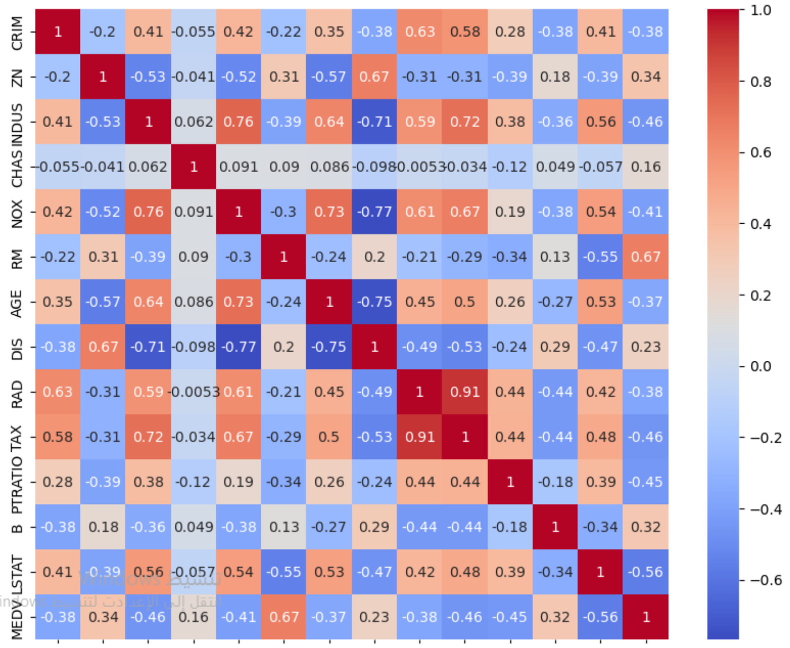




lets check the correlation between feature

In [7]:
```python
# lets check the correlation between feature
import seaborn as sns
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),cmap="coolwarm",annot=True)
```

Out[7]: <Axes: >



Data Clean

In [ ]:
```
# Data Clean
#we have null or nan values we need to clean
```

In [12]:
```
X = df.drop(['MEDV'],axis=1).values
y = df['MEDV'].values
```

In [18]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15)
```

In [19]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.metrics import r2_score
```

2) Extract necessary features

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
# we will with min max preprocessing for dataset
scaler = MinMaxScaler(feature_range=(0, 1))
```

In [22]:
```
df.head()
```

Out[22]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [23]:
```
X=df.drop(['MEDV'], axis=1)
rescaleddata = scaler.fit_transform(X)
```

In [47]:
```
newdf = pd.DataFrame(rescaleddata)
```

In [48]:
```
newdf.head()
```

Out[48]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.000000	0.18	0.067815	0.0	0.314815	0.577505	0.641607	0.269203	0.000000	0.208015	0.259615	1.000000	0.043759
1	0.000236	0.00	0.242302	0.0	0.172840	0.547998	0.782698	0.348962	0.043478	0.104962	0.500000	1.000000	0.099771
2	0.000236	0.00	0.242302	0.0	0.172840	0.694386	0.599382	0.348962	0.043478	0.104962	0.500000	0.989737	0.030968
3	0.000293	0.00	0.063050	0.0	0.150206	0.658555	0.441813	0.448545	0.086957	0.066794	0.586538	0.994276	0.016292
4	0.000705	0.00	0.063050	0.0	0.150206	0.687105	0.528321	0.448545	0.086957	0.066794	0.586538	1.000000	0.048472

n [50]:
```
newdf.describe()
```

ut[50]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
count	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000	511.000000
mean	0.040214	0.112524	0.391902	0.068493	0.349294	0.517323	0.676789	0.241366	0.368927	0.420688	0.567308	0.898383	0.150122
std	0.096262	0.232348	0.250300	0.252838	0.237263	0.143725	0.289383	0.190838	0.377760	0.320427	0.211572	0.229166	0.104987
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000854	0.000000	0.173387	0.000000	0.131687	0.444051	0.434089	0.088275	0.130435	0.176527	0.461538	0.944047	0.071833
50%	0.002870	0.000000	0.338343	0.000000	0.314815	0.506036	0.766220	0.183934	0.173913	0.272901	0.625000	0.985980	0.130874
75%	0.040630	0.125000	0.646628	0.000000	0.491770	0.587469	0.938723	0.362684	1.000000	0.914122	0.730769	0.998260	0.207015
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

check for missing values

```
In [51]: # some missing values need to be handle
         newdf.isnull().values.any()

Out[51]: False

In [52]: newdf.fillna(0, inplace = True)

In [53]: newdf.isnull().values.any()
         #### no nan values

Out[53]: False

In [54]: X=newdf.values
         y=df['MEDV'].values
```

3) Build a model

```
In [55]: # create regression model

In [56]: #we have to split the dataset into training and testing
         # to do so
         from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test = train_test_split(X, y,test_size=0.25)

In [57]: from sklearn.linear_model import LinearRegression

In [58]: model = LinearRegression()

In [59]: model=model.fit(x_train, y_train)

In [60]: r_squared = model.score(x_test, y_test)

In [61]: y_pred=model.predict(x_test)

In [62]: print(r_squared)
         0.667741871784163
```

4) Predict the asked queries (mentioned in the project) using training model.

```
In [59]: model=model.fit(x_train, y_train)

In [60]: r_squared = model.score(x_test, y_test)

In [61]: y_pred=model.predict(x_test)

In [62]: print(r_squared)
         0.667741871784163
```

5) Evaluate model performance (Like using accuracy, Precision, Recall, F1 Score etc.)

```
In [63]: # Model Evaluation

In [64]: from sklearn.model_selection import cross_val_score
         import numpy as np
         scores = cross_val_score(model, x_train, y_train, cv=5)
         print('Accuracies: %s' % scores)
         print('Mean accuracy: %s' % np.mean(scores))

         Accuracies: [0.64375278 0.13698322 0.21094195 0.73546633 0.62427374]
         Mean accuracy: 0.47028360186702844
```

The decision tree used in linear regression:

```
In [67]: #Decision Tree
         from sklearn.tree import DecisionTreeRegressor

In [68]: regr_2 = DecisionTreeRegressor(max_depth=5)

In [69]: regr_2.fit(x_train, y_train)

Out[69]: ▾ DecisionTreeRegressor
         DecisionTreeRegressor(max_depth=5)
```

Conclusion:

```
In [70]: #conclusion
         cores = cross_val_score(regr_2, x_train, y_train, cv=5)
         print('Accuracies: %s' % scores)
         print('Mean accuracy: %s' % np.mean(scores))

         Accuracies: [0.64375278 0.13698322 0.21094195 0.73546633 0.62427374]
         Mean accuracy: 0.47028360186702844

In [71]: #overall of mean accuracy for models
         print('Linear regression accuracy',np.mean(scores),'Decsion tree regression accuracy',np.mean(scores))

         Linear regression accuracy 0.47028360186702844 Decsion tree regression accuracy 0.47028360186702844
```

Conclusion:

the project aimed to predict the median value of owner-occupied homes in Boston using a regression analysis. The dataset used was fairly small with 506 rows and 14 columns, making it easy to apply various techniques without worrying about memory constraints. Through the analysis, we were able to identify key factors that influence the median value of homes in Boston, such as crime rate, pupil-teacher ratio, and accessibility to highways. Overall, this project provides valuable insights into the real estate industry in Boston and can be used as a basis for further research in this field.

References:

1- Lab Manual

2- https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

3- https://www.analyticsvidhya.com/blog/2015/11/started-machine-learning-ms-excel-xl-miner/

4- https://corporatefinanceinstitute.com/resources/data-science/regression-analysis/

Appendix: Program Code File

Machine Learning project.html

Machine Learning project.ipynb

has been added the code file in another file