Project Report

K2N Academy

GROUP-D

GROUP NAME:- TEAM MONK

GROUP MEMBERS:-

1. ANAND KUMAR
2. SANISH KUMAR
3. MD MAKKI
4. ABHISHEK ANAND
5. MD SHAKIR HASSAN
6. ANUKUL RAJ

TABLE OF CONTENTS

1.Title of the project

Housing_Price_Prediction

A machine learning project to predict the housing price.

2.Introduction of the project

Housing price prediction is a critical area of research in the field of data science and machine learning. With the growing demand for affordable and quality housing, accurate prediction of housing prices has become increasingly important for real estate investors, developers, and policymakers. The objective of this project is to develop a predictive model that can accurately estimate the prices of houses based on various features such as the number of rooms, the size of the property, and the location.

The results of this project can provide valuable insights into the factors that influence housing prices and can help stakeholders make informed decisions regarding real estate investments and development.

3.Application areas of the project

The project on "Housing Price Prediction" can have a wide range of practical applications in various domains, including:

Real estate industry: Accurate prediction of housing prices can help real estate investors and developers make informed decisions regarding buying, selling, and developing properties. The predictive model developed in this project can be used to estimate the market value of a property, which can assist in setting the right price for a property and in identifying investment opportunities.

Banking and finance: The project can be useful for financial institutions that offer home loans and mortgages. Accurate prediction of housing prices can help banks and other lending institutions determine the loan amount and interest rates based on the market value of the property.

Government and policy-making: Housing price prediction can also be useful for policymakers to identify areas of affordable housing and to monitor changes in housing prices over time. It can help policymakers identify trends and patterns in the housing market.

4.Description of all the predefined python libraries used in the projects

Python libraries used in the "Housing Price Prediction" project:

NumPy: NumPy is a Python library for performing numerical operations on arrays and matrices. It provides efficient and convenient functions for mathematical operations such as addition, subtraction, multiplication, and division.

Pandas: Pandas is a Python library for data manipulation and analysis. It provides data structures for handling and analyzing tabular data, and supports operations such as filtering, merging, grouping, and aggregating data.

Matplotlib: Matplotlib is a Python library for creating visualizations and plots. It provides a variety of functions for creating line charts, scatter plots, histograms, and other types of visualizations.

Seaborn: Seaborn is a Python library for creating statistical visualizations. It provides a set of high-level functions for creating complex visualizations such as heatmaps, pair plots, and distribution plots.

Scikit-learn: Scikit-learn is a Python library for machine learning. It provides a variety of functions for data preprocessing, feature selection, model selection, and evaluation. It also includes a set of popular machine learning algorithms such as linear regression, decision trees, and support vector machines.

MinMaxScaler: MinMaxScaler is a function from the Scikit-learn library that is used for scaling numerical data. It scales the data to a specific range, typically between 0 and 1, which can improve the performance of machine learning algorithms.

Train_test_split: Train_test_split is a function from the Scikit-learn library that is used for splitting data into training and testing sets. It randomly splits the data into two subsets based on a specified ratio, which can be used to evaluate the performance of machine learning models.

These libraries provide powerful tools and functions for performing various data manipulation, analysis, and visualization tasks, as well as building and evaluating machine learning models.

5. Show the application code of the libraries with a basic example and justify its use in the project

In [2]:

```
# Importing necessary Libraries
import numpy as np
import pandas as pd

housing = pd.read_csv('Housing (2).csv')
housing.head()
```

Out[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

In [3]:

```
# Checking for null values
print(housing.info())

# Checking for outliers
print(housing.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
```

In [4]:
```python
# Converting the categorical variable into numerical // Data Preparation
varlist =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)

# Check the housing dataframe now
housing
```

Out[4]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished
...	...	...	...	...	...	...	...	...	...	...	...	...	...
540	1820000	3000	2	1	1	1	0	1	0	0	2	0	unfurnished

In [5]:
```python
# Creating dummy variable
status = pd.get_dummies(housing['furnishingstatus'])

# Check what the dataset 'status' looks like
status
```

Out[5]:

	furnished	semi-furnished	unfurnished
0	1	0	0
1	1	0	0
2	0	1	0
3	1	0	0
4	1	0	0
...	...	...	...
540	0	0	1

In [6]:
```python
# Dropping the first column from status dataset
status = pd.get_dummies(housing['furnishingstatus'], drop_first = True)

# Adding the status to the original housing dataframe
housing = pd.concat([housing, status], axis = 1)

# Dropping 'furnishingstatus' as we have created the dummies for it
housing.drop(['furnishingstatus'], axis = 1, inplace = True)
housing
```

Out[6]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	semi-furnished	unfurnish
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	0	
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	0	
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	1	
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	0	
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
540	1820000	3000	2	1	1	1	0	1	0	0	2	0	0	

In [7]:
```
from sklearn.model_selection import train_test_split

# We specify random seed so that the train and test data set always have the same rows, respectively
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random_state = 100)
```

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])

df_train
```

```
<ipython-input-8-f4ad772414d1>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
C:\Users\DevOp\anaconda3\lib\site-packages\pandas\core\indexing.py:966: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

In [9]:
```
# Dividing the training data set into X and Y
y_train = df_train.pop('price')
X_train = df_train
```

In [10]:
```
X_train
```

Out[10]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	semi-furnished	unfurnished
359	0.155227	0.4	0.0	0.000000	1	0	0	0	0	0.333333	0	0	1
19	0.403379	0.4	0.5	0.333333	1	0	0	0	1	0.333333	1	1	0
159	0.115628	0.4	0.5	0.000000	1	1	1	0	1	0.000000	0	0	0
35	0.454417	0.4	0.5	1.000000	1	0	0	0	1	0.666667	0	0	0
28	0.538015	0.8	0.5	0.333333	1	0	1	1	0	0.666667	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
526	0.118268	0.2	0.0	0.000000	1	0	0	0	0	0.000000	0	0	1
53	0.291623	0.4	0.5	1.000000	1	0	0	0	1	0.666667	0	1	0
350	0.139388	0.2	0.0	0.333333	1	0	0	1	0	0.333333	0	1	0

In [11]:
```
#Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.681
Model:	OLS	**Adj. R-squared:**	0.670
Method:	Least Squares	**F-statistic:**	60.40
Date:	Thu, 02 Mar 2023	**Prob (F-statistic):**	8.83e-83
Time:	20:08:02	**Log-Likelihood:**	381.79
No. Observations:	381	**AIC:**	-735.6
Df Residuals:	367	**BIC:**	-680.4
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

In [12]:
```python
# Checking for the VIF values of the variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Creating a dataframe that will contain the names of all the feature variables and their VIFs
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[12]:

	Features	VIF
1	bedrooms	7.33
4	mainroad	6.02
0	area	4.67
3	stories	2.70
11	semi-furnished	2.19
9	parking	2.12
6	basement	2.02
12	unfurnished	1.82

In [13]:
```python
# Dropping highly correlated variables and insignificant variables
X = X_train.drop('semi-furnished', 1,)

# Build a fitted model after dropping the variable
X_train_lm = sm.add_constant(X)

lr_2 = sm.OLS(y_train, X_train_lm).fit()

# Printing the summary of the model
print(lr_2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.681
Model:                            OLS   Adj. R-squared:                  0.671
Method:                 Least Squares   F-statistic:                     65.61
Date:                Thu, 02 Mar 2023   Prob (F-statistic):           1.07e-83
Time:                        20:08:26   Log-Likelihood:                 381.79
No. Observations:                 381   AIC:                            -737.6
Df Residuals:                     368   BIC:                            -686.3
Df Model:                          12
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
```

In [14]:
```python
# Calculating the VIFs again for the new model after dropping semi-furnished

vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[14]:

	Features	VIF
1	bedrooms	6.59
4	mainroad	5.68
0	area	4.67
3	stories	2.69
9	parking	2.12
6	basement	2.01
8	airconditioning	1.77
2	bathrooms	1.67
10	prefarea	1.51

In [15]:
```
X = X.drop('bedrooms', 1)
# Build a second fitted model
X_train_lm = sm.add_constant(X)
lr_3 = sm.OLS(y_train, X_train_lm).fit()

# Printing the summary of the model
print(lr_3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.680
Model:                            OLS   Adj. R-squared:                  0.671
Method:                 Least Squares   F-statistic:                     71.31
Date:                Thu, 02 Mar 2023   Prob (F-statistic):           2.73e-84
Time:                        20:08:50   Log-Likelihood:                 380.96
No. Observations:                 381   AIC:                            -737.9
Df Residuals:                     369   BIC:                            -690.6
Df Model:                          11
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const               0.0357      0.015      2.421      0.016       0.007       0.065
area                0.2347      0.030      7.851      0.000       0.176       0.294
bathrooms           0.1965      0.022      9.132      0.000       0.154       0.239
stories             0.1178      0.018      6.654      0.000       0.083       0.153
mainroad            0.0488      0.014      3.423      0.001       0.021       0.077
```

In [16]:
```
# Calculating the VIFs again for the new model
vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[16]:

	Features	VIF
3	mainroad	4.79
0	area	4.55
2	stories	2.23
8	parking	2.10
5	basement	1.87
7	airconditioning	1.76
1	bathrooms	1.61
9	prefarea	1.50
4	guestroom	1.46
10	unfurnished	1.33

In [17]:
```
X = X.drop('basement', 1)
X_train_lm = sm.add_constant(X)

lr_4 = sm.OLS(y_train, X_train_lm).fit()
print(lr_4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.676
Model:                            OLS   Adj. R-squared:                  0.667
Method:                 Least Squares   F-statistic:                     77.18
Date:                Thu, 02 Mar 2023   Prob (F-statistic):           3.13e-84
Time:                        20:09:12   Log-Likelihood:                 378.51
No. Observations:                 381   AIC:                            -735.0
Df Residuals:                     370   BIC:                            -691.7
Df Model:                          10
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
```

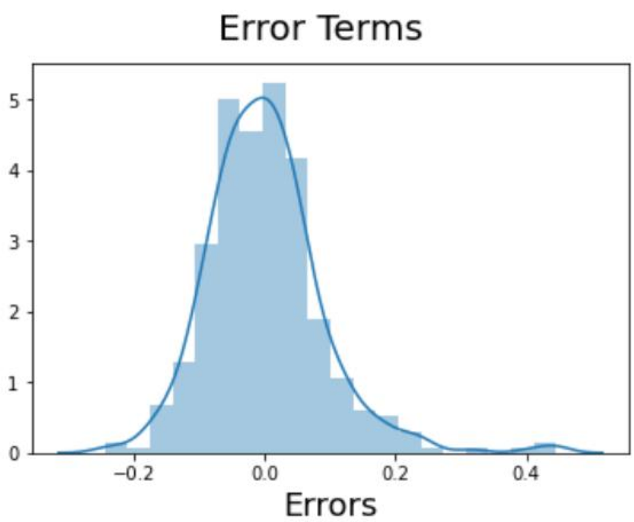
```
In [18]: # Calculate the VIFs again for the new model
vif = pd.DataFrame()
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[18]:

	Features	VIF
3	mainroad	4.55
0	area	4.54
2	stories	2.12
7	parking	2.10
6	airconditioning	1.75
1	bathrooms	1.58
8	prefarea	1.47
9	unfurnished	1.33
4	guestroom	1.30
5	hotwaterheating	1.12

```
In [19]: import seaborn as sns
import matplotlib.pyplot as plt
y_train_price = lr_4.predict(X_train_lm)
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                          # X-label
```

Out[19]: Text(0.5, 0, 'Errors')



```
In [20]: num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
         df_test[num_vars] = scaler.transform(df_test[num_vars])

         df_test
```

```
<ipython-input-20-9e94bccbc316>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df_test[num_vars] = scaler.transform(df_test[num_vars])
C:\Users\DevOp\anaconda3\lib\site-packages\pandas\core\indexing.py:966: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  self.obj[item] = s
```

```
In [21]: y_test = df_test.pop('price')
         X_test = df_test

         # Adding constant variable to test dataframe
         X_test_m4 = sm.add_constant(X_test)

         # Creating X_test_m4 dataframe by dropping variables from X_test_m4
         X_test_m4 = X_test_m4.drop(["bedrooms", "semi-furnished", "basement"], axis = 1)

         # Making predictions using the final model
         y_pred_m4 = lr_4.predict(X_test_m4)
```

```
In [22]: from sklearn.metrics import r2_score
         r2_score(y_true = y_test, y_pred = y_pred_m4)

Out[22]: 0.6601344030219642
```

```
In [23]: # Importing RFE and LinearRegression
         from sklearn.feature_selection import RFE
         from sklearn.linear_model import LinearRegression
```

```
In [24]: # Running RFE with the output number of the variable equal to 10
         lm = LinearRegression()
         lm.fit(X_train, y_train)

         rfe = RFE(lm, 10)             # running RFE
         rfe = rfe.fit(X_train, y_train)

         list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

```
C:\Users\DevOp\anaconda3\lib\site-packages\sklearn\utils\validation.py:68: FutureWarning: Pass n_features_to_select=10 as keywo
rd args. From version 0.25 passing these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

```
Out[24]: [('area', True, 1),
          ('bedrooms', True, 1),
          ('bathrooms', True, 1),
          ('stories', True, 1),
          ('mainroad', True, 1),
          ('guestroom', True, 1),
          ('basement', False, 3),
          ('hotwaterheating', True, 1),
          ('airconditioning', True, 1),
          ('parking', True, 1),
          ('prefarea', True, 1),
          ('semi-furnished', False, 4),
          ('unfurnished', False, 2)]
```

In [25]:
```
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train[ X_train.columns]

# Adding a constant variable
import statsmodels.api as sm
X_train_rfe = sm.add_constant(X_train_rfe)

lm = sm.OLS(y_train,X_train_rfe).fit()   # Running the linear model

print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.681
Model:                            OLS   Adj. R-squared:                  0.670
Method:                 Least Squares   F-statistic:                     60.40
Date:                Thu, 02 Mar 2023   Prob (F-statistic):           8.83e-83
Time:                        20:10:51   Log-Likelihood:                 381.79
No. Observations:                 381   AIC:                            -735.6
Df Residuals:                     367   BIC:                            -680.4
Df Model:                          13
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
```

In [26]:
```
X_train_new = X_train_rfe.drop(["bedrooms"], axis = 1)

# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model

print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.680
Model:                            OLS   Adj. R-squared:                  0.670
Method:                 Least Squares   F-statistic:                     65.20
Date:                Thu, 02 Mar 2023   Prob (F-statistic):           2.35e-83
Time:                        20:11:05   Log-Likelihood:                 380.96
No. Observations:                 381   AIC:                            -735.9
Df Residuals:                     368   BIC:                            -684.7
Df Model:                          12
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
```

In [27]:
```
X_train_new = X_train_new.drop(['const'], axis=1)
# Calculate the VIFs for the new model
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

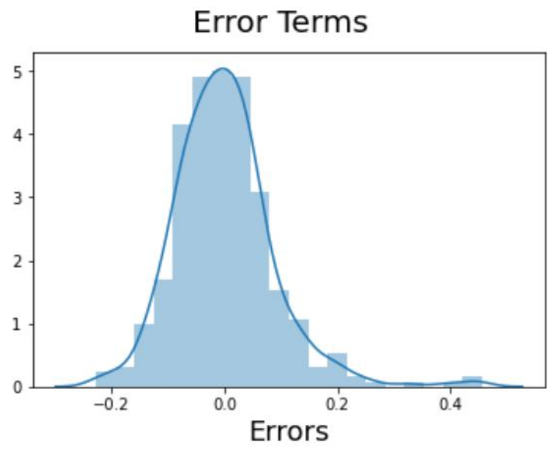
Out[27]:

	Features	VIF
3	mainroad	5.53
0	area	4.55
2	stories	2.24
8	parking	2.11
10	semi-furnished	1.97
5	basement	1.90
7	airconditioning	1.77
11	unfurnished	1.62

```
In [28]: y_train_price = lm.predict(X_train_lm)
         # Importing the required libraries for plots.
         import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline

         # Plot the histogram of the error terms
         fig = plt.figure()
         sns.distplot((y_train - y_train_price), bins = 20)
         fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
         plt.xlabel('Errors', fontsize = 18)                         # X-label

Out[28]: Text(0.5, 0, 'Errors')
```



6. select the concerned dataset from the specified link in google and mention the link

https://www.kaggle.com/datasets/peterkmutua/housing-dataset

7. clean the dataset and check if there are any duplicate data/ NaN data etc still exist

'NO' there are no duplicate data exists in the dataset

In [28]:
```
import seaborn
seaborn.heatmap(housing.isnull())
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x1ad337df550>