PROBLEM STATEMENT:

PREDICTION OF HOUSE PRICING USING MACHINE LEARNING WITH PYTHON

Importing the necessary packages and modules

- **numpy** package can be used to perform mathematical operations like 'mean'.
- **pandas** package can be used to process dataframes.
- **seaborn** package can be used to visualise data in the form of various effective graphs and plots.
- **sklearn** is the main package which is used for machine learning.
- **LabelEncoder** is used to encode the non-numeric data into numericals so that machine learning model can be built.
- **train_test_split module** is used to split the data into training and testing sets.
- **LinearRegression** module is used to fit a LinearRegression model.
- **sklearn.metrics** can be used to calculate statistical results like mean squared error, root mean squared error, etc.

In [127]:
```
#importing the libraries
import numpy as np
import pandas as pd
import seaborn as seb
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score
%matplotlib inline
```

Reading the dataset

- The **dataset** needs to be imported and read - we use **pandas** to acheive this.

In [128]:
```
#reading the dataset
train_data = pd.read_csv('house_sales_data.csv')
train_data.head(5)
```

Out[128]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...
0	7129300520	20141013	221900.0	3	1.00	1180	5650	1.0	0	0	...
1	6414100192	20141209	538000.0	3	2.25	2570	7242	2.0	0	0	...
2	5631500400	20150225	180000.0	2	1.00	770	10000	1.0	0	0	...
3	2487200875	20141209	604000.0	4	3.00	1960	5000	1.0	0	0	...
4	1954400510	20150218	510000.0	3	2.00	1680	8080	1.0	0	0	...

5 rows × 21 columns

In [129]:
```
train_data.shape
```

Out[129]:
```
(21613, 21)
```

In [130]:
```python
train_data.drop(columns=['id','view','condition','grade','sqft_above','yr_built','sqft_l
```

Out[130]:

	date	price	bedrooms	bathrooms	sqft_living	floors	waterfront	sqft_basement	zipcode
0	20141013	221900.0	3	1.00	1180	1.0	0	0	98178
1	20141209	538000.0	3	2.25	2570	2.0	0	400	98125
2	20150225	180000.0	2	1.00	770	1.0	0	0	98028
3	20141209	604000.0	4	3.00	1960	1.0	0	910	98136
4	20150218	510000.0	3	2.00	1680	1.0	0	0	98074
...	...	...	...	...	...	...	...	...	...
21608	20140521	360000.0	3	2.50	1530	3.0	0	0	98103
21609	20150223	400000.0	4	2.50	2310	2.0	0	0	98146
21610	20140623	402101.0	2	0.75	1020	2.0	0	0	98144
21611	20150116	400000.0	3	2.50	1600	2.0	0	0	98027
21612	20141015	325000.0	2	0.75	1020	2.0	0	0	98144

21613 rows × 11 columns

Processing the dataset

- After the data has been imported, we have to **clean/preprocess** the data to actually fit into a **regression** model

1. Checking for missing entries

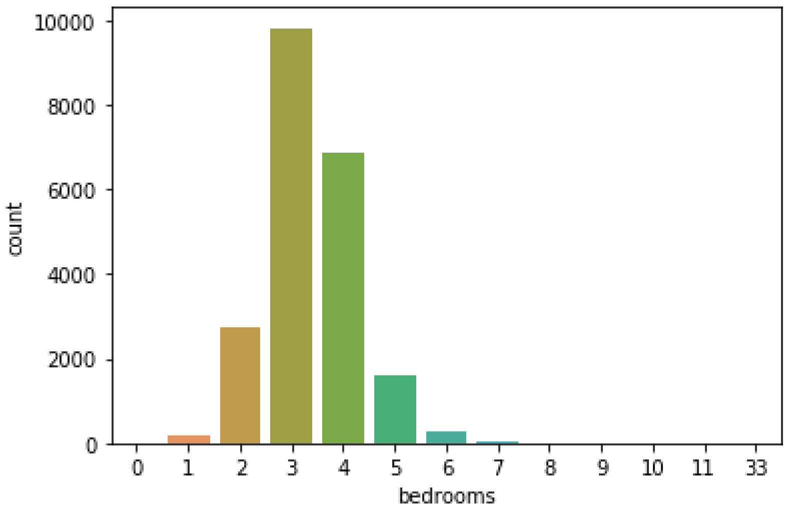
In [131]:
```python
#Checking for missing entries
train_data.isnull().sum()
```

Out[131]:
```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
view              0
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated      0
zipcode           0
lat               0
long              0
```

VISUALIZATIONS OF THE DATA

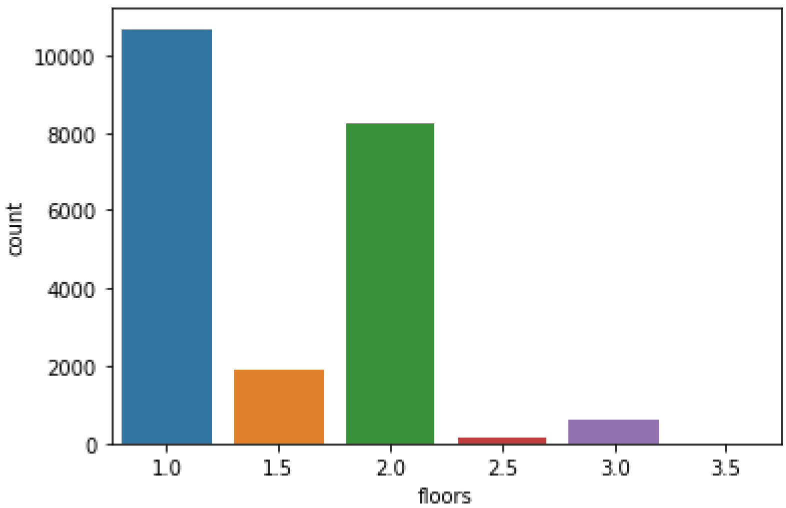
In [132]:
```
#barplot on gender
seb.countplot(train_data['bedrooms'])
```

Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x152527a1e48>



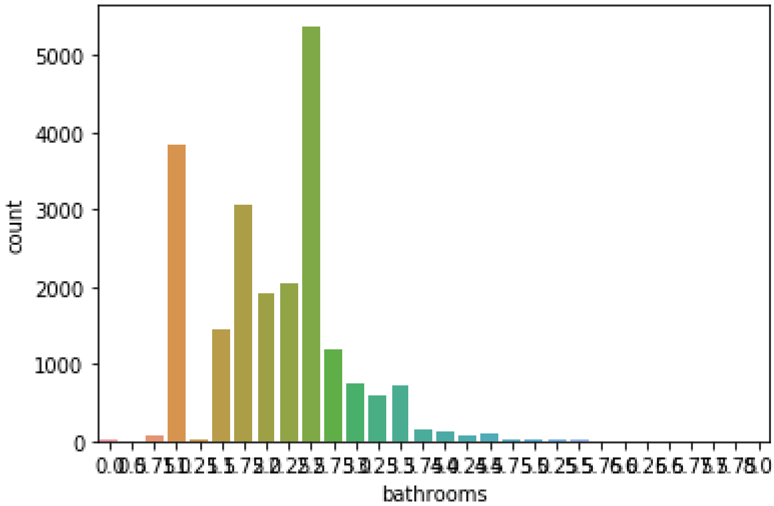
In [133]:
```
#barplot on martial status
seb.countplot(train_data['floors'])
```

Out[133]: <matplotlib.axes._subplots.AxesSubplot at 0x152527ece08>



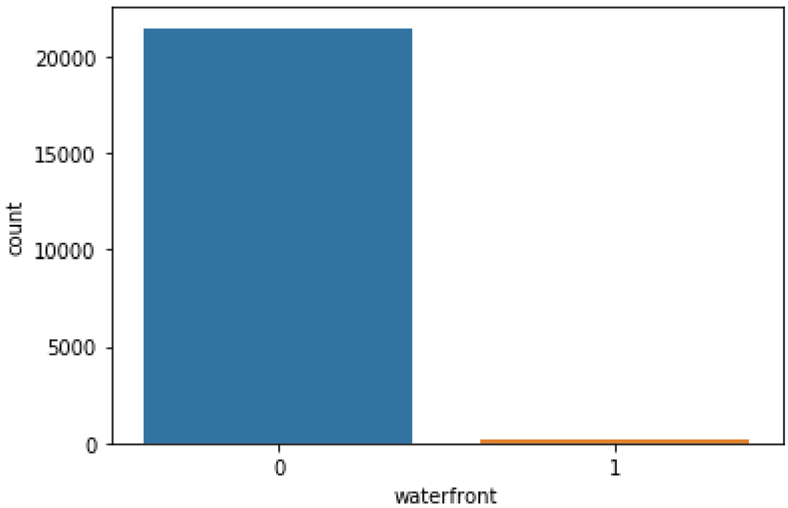
In [134]:
```python
#barplot on martial status
seb.countplot(train_data['bathrooms'])
```

Out[134]: <matplotlib.axes._subplots.AxesSubplot at 0x15249860dc8>



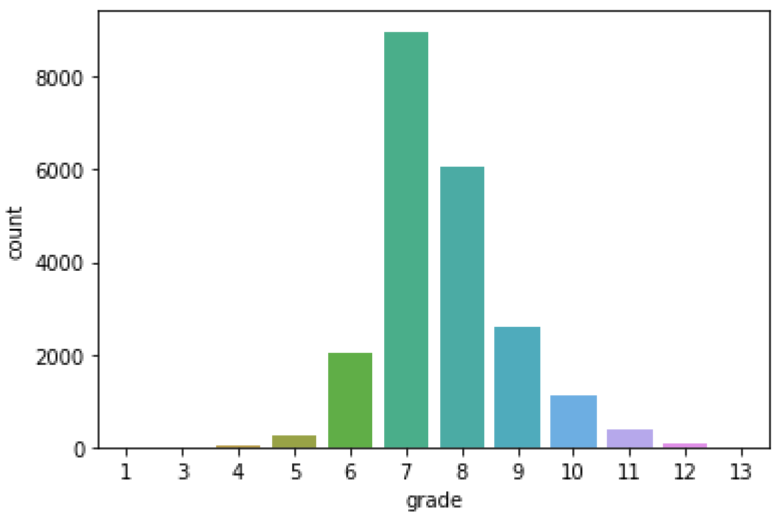
In [135]:
```python
#barplot on stay in current city
seb.countplot(train_data['waterfront'])
```

Out[135]: <matplotlib.axes._subplots.AxesSubplot at 0x15249729048>



In [136]:
```python
seb.countplot(train_data['grade'])
```

Out[136]: <matplotlib.axes._subplots.AxesSubplot at 0x15249840788>



3. Encoding the data set so as to make it easy for building machine learning

model

- The original data has **non-numerical** entries for few columns
- We encode these non-numerical entries using **LabelEncoder**

In [137]:
```
label_enc = LabelEncoder()
data_enc = train_data

# encoding few string-contained columns
#data_enc.Product_ID = label_enc.fit_transform(train_data.Product_ID)
data_enc.waterfront = label_enc.fit_transform(train_data.waterfront)
data_enc.head(10)
```

Out[137]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...
0	7129300520	20141013	221900.0	3	1.00	1180	5650	1.0	0	0	...
1	6414100192	20141209	538000.0	3	2.25	2570	7242	2.0	0	0	...
2	5631500400	20150225	180000.0	2	1.00	770	10000	1.0	0	0	...
3	2487200875	20141209	604000.0	4	3.00	1960	5000	1.0	0	0	...
4	1954400510	20150218	510000.0	3	2.00	1680	8080	1.0	0	0	...
5	7237550310	20140512	1230000.0	4	4.50	5420	101930	1.0	0	0	...
6	1321400060	20140627	257500.0	3	2.25	1715	6819	2.0	0	0	...
7	2008000270	20150115	291850.0	3	1.50	1060	9711	1.0	0	0	...
8	2414600126	20150415	229500.0	3	1.00	1780	7470	1.0	0	0	...
9	3793500160	20150312	323000.0	3	2.50	1890	6560	2.0	0	0	...

10 rows × 21 columns

Finding the correlation using corr()

In [138]:
```
#Finding the correlation using corr()
train_data.corr()
```

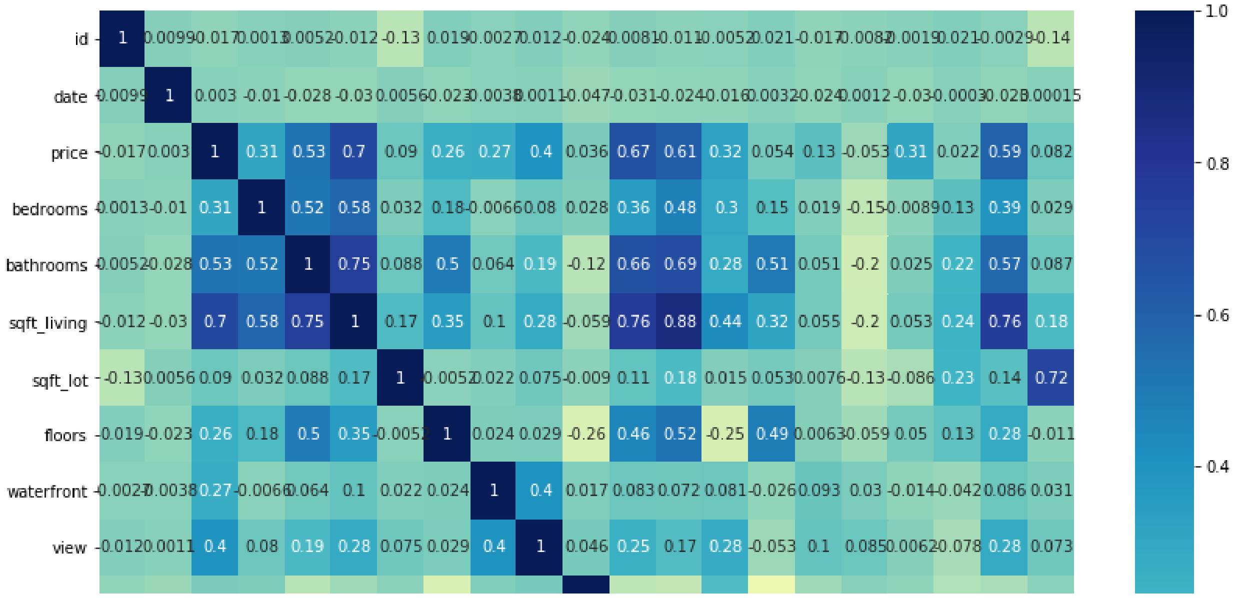
Out[138]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
id	1.000000	0.009857	-0.016797	0.001286	0.005160	-0.012258	-0.132109	0.018525
date	0.009857	1.000000	0.003011	-0.010493	-0.027523	-0.029791	0.005599	-0.022550
price	-0.016797	0.003011	1.000000	0.308338	0.525134	0.702044	0.089655	0.256786
bedrooms	0.001286	-0.010493	0.308338	1.000000	0.515884	0.576671	0.031703	0.175429
bathrooms	0.005160	-0.027523	0.525134	0.515884	1.000000	0.754665	0.087740	0.500653
sqft_living	-0.012258	-0.029791	0.702044	0.576671	0.754665	1.000000	0.172826	0.353949
sqft_lot	-0.132109	0.005599	0.089655	0.031703	0.087740	0.172826	1.000000	-0.005201
floors	0.018525	-0.022550	0.256786	0.175429	0.500653	0.353949	-0.005201	1.000000
waterfront	-0.002721	-0.003798	0.266331	-0.006582	0.063744	0.103818	0.021604	0.023698
view	0.011592	0.001063	0.397346	0.079532	0.187737	0.284611	0.074710	0.029444

visualizing the correlation

In [139]:
```
#visualizing the correlation
plt.figure(figsize=(14,14))
seb.heatmap(train_data.corr(),annot=True,cmap='YlGnBu')
```

Out[139]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x152497fe888>
```



Declaring input and output variables

- Input variables are considered as all the columns except the Purchase column
- Output variables are considered as the last column, i.e, the Purchase column

In [140]:
```
#Declaring input and output variables
X=train_data.drop(['price'],axis=1)
y=train_data.price
```

Splitting the data into train and test sets

- Input train and test sets are 2 dimensional
- Output train and test sets are 1 dimensional

In [141]:
```
#divide the X and y into train and test
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,random_state=0)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(16209, 20)
(5404, 20)
(16209,)
(5404,)
```

Fitting the data into Linear Regression model

In [142]:
```
#Fitting the data into Linear Regression model
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(X_train,y_train)
```

Out[142]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

intercept and coefficient values

In [143]:
```
#intercept and coefficient values
print(lm.intercept_)
lm.coef_
```

```
-64311704.26838564
```

Out[143]:
```
array([-1.43118221e-06,  3.39620948e+00, -3.31970856e+04,  3.43628407e+04,
        1.11342875e+02,  1.75115838e-01,  1.23078695e+04,  6.12741964e+05,
        5.04666180e+04,  2.86207206e+04,  9.47344313e+04,  6.69725948e+01,
        4.43702801e+01, -2.56332132e+03,  2.27184570e+01, -5.50529117e+02,
        6.06775307e+05, -2.07808176e+05,  2.81413801e+01, -4.45231202e-01])
```

creating a dataframe for cofficients

In [144]:
```
#creating a dataframe for cofficients
coefficients=pd.DataFrame([X_train.columns,lm.coef_]).T
coefficients
```

Out[144]:

	0	1
0	id	-1.43118e-06
1	date	3.39621
2	bedrooms	-33197.1
3	bathrooms	34362.8
4	sqft_living	111.343
5	sqft_lot	0.175116
6	floors	12307.9
7	waterfront	612742
8	view	50466.6
9	condition	28620.7
10	grade	94734.4

checking the model prediction on training data

In [145]:
```
#checking the model prediction on training data
y_train_pred = lm.predict(X_train)
y_train_pred
```

Out[145]:
```
array([239946.65604577, 626093.45027984, 660257.21464739, ...,
       626236.47679655, 482824.07957322, 542928.42368714])
```

comparing the actual values(y_train) and the predicted values(y_train_pred)

In [146]:
```
#comparing the actual values(y_train) and the predicted values(y_train_pred)
y_train==y_train_pred
```

Out[146]:
```
1956     False
15678    False
8729     False
19064    False
11291    False
         ...
13123    False
19648    False
9845     False
10799    False
2732     False
Name: price, Length: 16209, dtype: bool
```

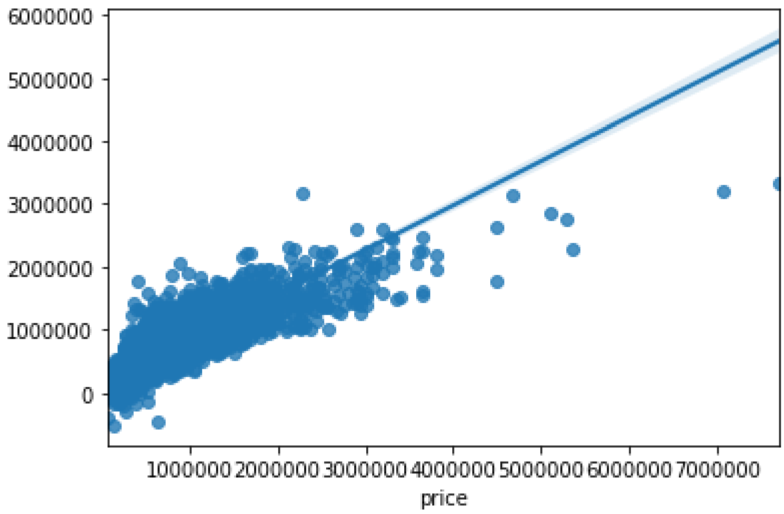
In [147]:
```
#r2_score
from sklearn.metrics import r2_score
print("R^2:",r2_score(y_train,y_train_pred))
print("Adjusted R^2:",1-(1-r2_score(y_train,y_train_pred))*(len(X_train)-1)/
     (len(X_train)-X_train.shape[1]-1))
```

```
R^2: 0.7043486540498882
Adjusted R^2: 0.7039833818161965
```

regplot

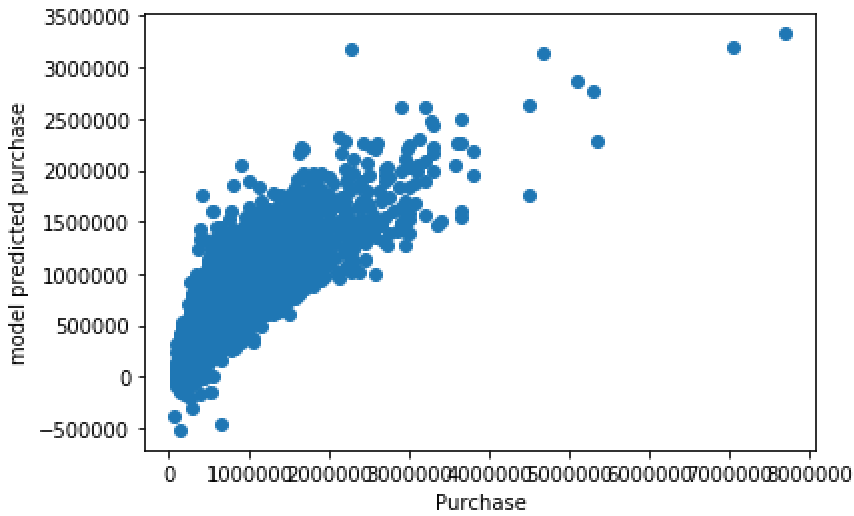
In [148]:
```
#regplot
seb.regplot(y_train,y_train_pred)
```

Out[148]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15242b63d48>
```



Visualizing the differences between acutal values and predicted values

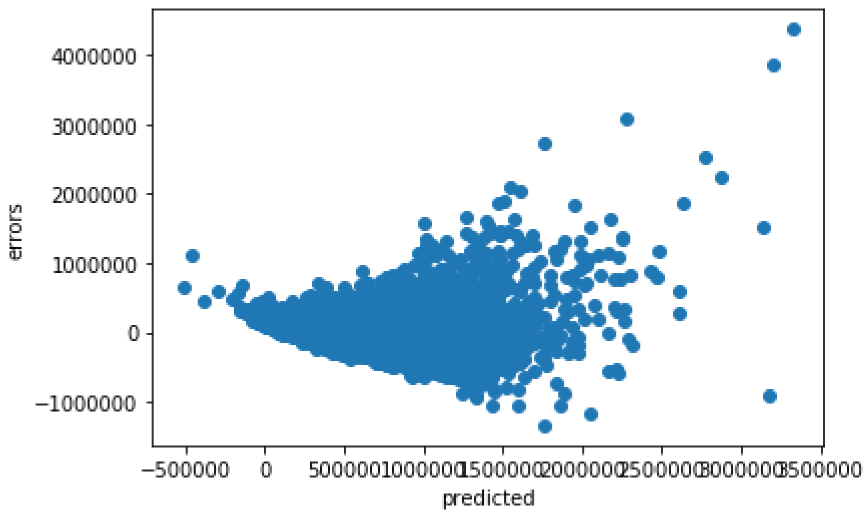
In [149]:
```
#Visualizing the differences between acutal values and predicted values
plt.scatter(y_train,y_train_pred)
plt.xlabel('Purchase')
plt.ylabel('model predicted purchase')
```
Out[149]: Text(0, 0.5, 'model predicted purchase')



plot for residuals(errors)

- y_train-y_train_pred=Errors

In [150]:
```
#plot for residuals(errors)
plt.scatter(y_train_pred,y_train-y_train_pred)
plt.xlabel("predicted")
plt.ylabel("errors")
```
Out[150]: Text(0, 0.5, 'errors')



Predicting the output test values for input test values

In [151]:
```
#Predicting the output test values for input test values
y_test_pred=lm.predict(X_test)
y_test_pred
```

Out[151]:
```
array([ 379999.00895102, 1524808.66022637,  534097.77845981, ...,
        740534.70096395,  220123.8309894 ,  580489.20148619])
```

Comparing the actual output test values with the predicted output test values

- returning boolean values while comparing the output test values with predicted output test values

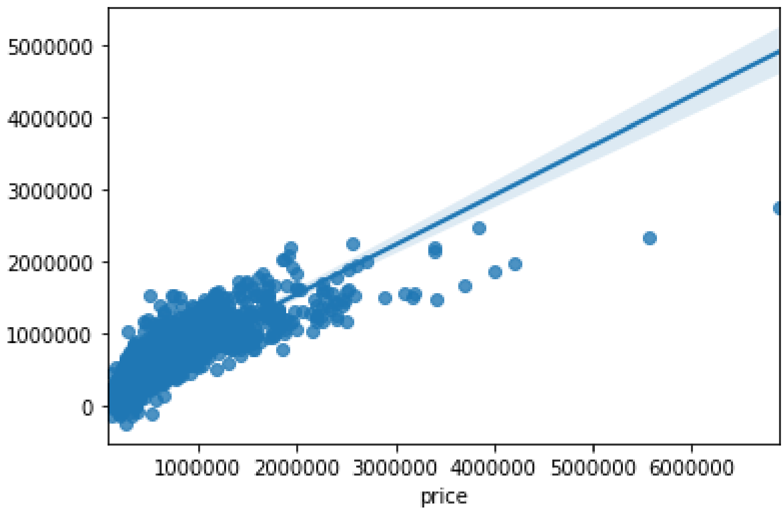
In [152]:
```
#Comparing the actual output test values with the predicted output test values
y_test==y_test_pred
```

Out[152]:
```
17384    False
722      False
2680     False
18754    False
14554    False
         ...
8709     False
12346    False
10458    False
10894    False
15647    False
Name: price, Length: 5404, dtype: bool
```

Visualising the data for the actual values vs. the predicted values

In [153]:
```
#Visualising the data for the actual values vs. the predicted values
seb.regplot(y_test, y_test_pred)
```

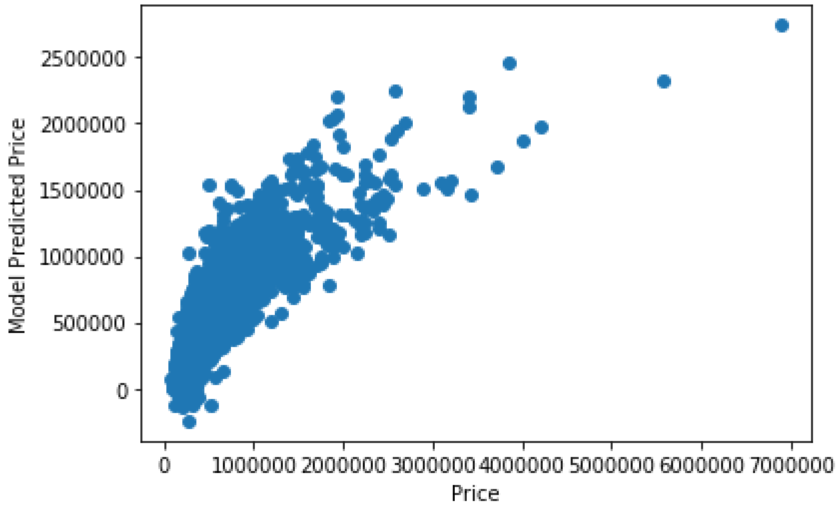
Out[153]: `<matplotlib.axes._subplots.AxesSubplot at 0x15252dcd8c8>`



Visualizing the differences between acutal values and predicted values

In [154]:
```
#Visualizing the differences between acutal values and predicted values
plt.scatter(y_test, y_test_pred)
plt.xlabel('Price')
plt.ylabel('Model Predicted Price')
```

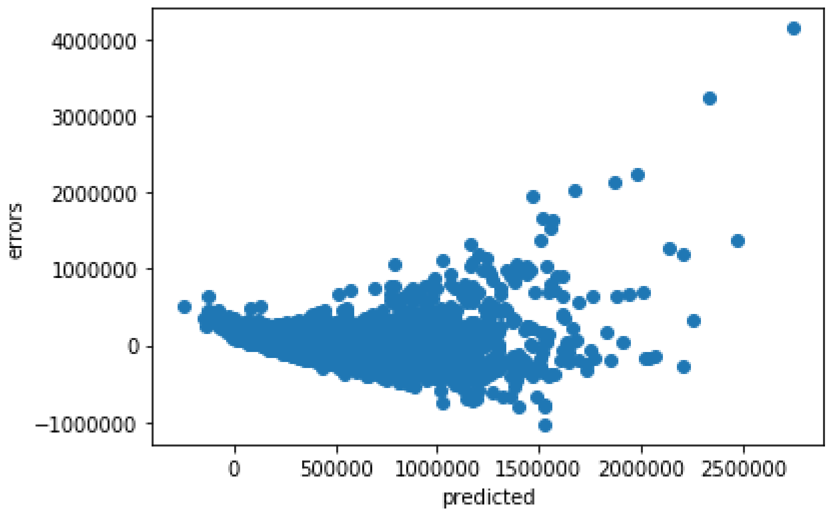
Out[154]: Text(0, 0.5, 'Model Predicted Price')



plot for residuals(errors)

In [155]:
```
#plot for residuals(errors)
plt.scatter(y_test_pred,y_test-y_test_pred)
plt.xlabel("predicted")
plt.ylabel("errors")
```

Out[155]: Text(0, 0.5, 'errors')



Statistical Results

In [156]:
```
#Statistical Results
difference = np.mean(y_test) - np.mean(y_test_pred)
error = (np.mean(y_test) - np.mean(y_test_pred))/np.mean(y_test)

print('Predicted Mean : %.2f' % np.mean(y_test_pred), end = '\n\n')
print('Actual Mean : %.2f' % np.mean(y_test), end = '\n\n')
print('Difference : %.2f' % difference, end = '\n\n')
print('Coefficients :')
print(lm.coef_, end = '\n\n')
print('Variance score: %.4f' % lm.score(X_test, y_test), end = '\n\n')
print('Percentage Error : %.4f' % (error*100), end = '\n\n')
```

```
Predicted Mean : 536509.96

Actual Mean : 535194.21

Difference : -1315.75

Coefficients :
[-1.43118221e-06  3.39620948e+00 -3.31970856e+04  3.43628407e+04
  1.11342875e+02  1.75115838e-01  1.23078695e+04  6.12741964e+05
  5.04666180e+04  2.86207206e+04  9.47344313e+04  6.69725948e+01
  4.43702801e+01 -2.56332132e+03  2.27184570e+01 -5.50529117e+02
  6.06775307e+05 -2.07808176e+05  2.81413801e+01 -4.45231202e-01]

Variance score: 0.6912

Percentage Error : -0.2458
```

Error Metrics

1. Mean Absolute Error
2. Mean Squared Error
3. Root Mean Squared Error

Performance Metrics

4. R^2 value
5. Adjusted R^2 value

In [157]:
```
#Error Metrics
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
print("R^2:",r2_score(y_test,y_test_pred))
print("MAE:",mean_absolute_error(y_test,y_test_pred))
print("MSE:",mean_squared_error(y_test,y_test_pred))
print("RMSE:",np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

```
R^2: 0.6912143237690274
MAE: 123084.86547643815
MSE: 41073289683.144936
RMSE: 202665.4624822516
```

scaling

In [158]:
```python
# Scaling Data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Scaling for training data
scaled_X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X_train.columns)
scaled_X_train

#Scaling for test data
#Testing the data based on training data
scaled_X_test = pd.DataFrame(scaler.transform(X_test), columns = X_test.columns)
scaled_X_test
```

Out[158]:

	id	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vie
0	-1.086728	-0.651031	-1.448419	-0.801701	-0.704248	-0.321932	2.769831	-0.089917	-0.30789
1	-0.817465	-0.700757	0.667617	1.467506	2.799781	0.864530	0.925552	-0.089917	-0.30789
2	-0.627961	-0.741933	-1.448419	-1.774219	-0.693433	-0.273468	-0.918726	-0.089917	-0.30789
3	0.786002	-0.742158	-1.448419	-1.450046	-1.028695	-0.298528	-0.918726	-0.089917	-0.30789
4	-0.188286	1.463323	0.667617	0.494988	1.188360	-0.133913	0.925552	-0.089917	2.30298
...	...	...	...	...	...	...	...	...	.
5399	1.025058	-0.674657	0.667617	0.170816	0.052795	-0.079609	0.925552	-0.089917	-0.30789
5400	-0.345106	1.413822	-0.390401	-0.153356	-1.158474	-0.025636	-0.918726	-0.089917	-0.30789
5401	0.907430	-0.629656	0.667617	0.494988	1.437103	3.880759	0.925552	-0.089917	-0.30789
5402	0.785548	-0.746433	-0.390401	-0.801701	-0.974621	-0.069585	-0.918726	-0.089917	-0.30789

In [159]:
```python
# Model Building:
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=40, metric='euclidean')

# Apply the knn object on the dataset(Training Phase)
# Syntax: objectName.fit(Input, Output)
knn.fit(scaled_X_train, y_train)
```

Out[159]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='euclidean',
                    metric_params=None, n_jobs=None, n_neighbors=40, p=2,
                    weights='uniform')
```

In [160]:
```python
# Predictions on the data
#predict function--> gives the predicted values
# Syntax:objectname.predict(Input)
y_train_pred = knn.predict(scaled_X_train)
y_train_pred
```

Out[160]:
```
array([332339.875, 501747.5  , 522470.   , ..., 524873.525, 336740.65 ,
       560839.025])
```

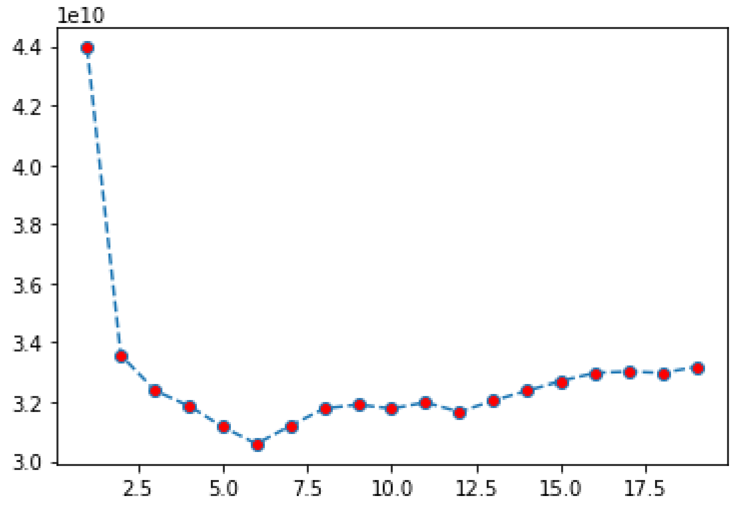
Implementing knn regressor

In [161]:
```
# Checking for optimum k-value
# Build the models with multiple k values
scores=[]
for k in range(1, 20):
    knn_model = KNeighborsRegressor(n_neighbors = k)
    knn_model.fit(scaled_X_train, y_train)
    pred_test = knn_model.predict(scaled_X_test)
    scores.append(mean_squared_error(y_test, pred_test))
scores
```

Out[161]:
```
[43953341282.64415,
 33563658924.766006,
 32392041124.890846,
 31872224783.449516,
 31156133341.33798,
 30584384339.37212,
 31194779038.290142,
 31777420980.003494,
 31910522500.994877,
 31775748547.535015,
 31986163036.668613,
 31665755501.021454,
 32043421759.26801,
 32379021003.79867,
 32695944209.45484,
 32965916001.68639,
 33013250627.648537,
 32966067904.589268,
 33171096388.3296]
```

In [162]:
```
# Plottting of K values and Scores
plt.plot(range(1,20), scores, marker='o', markerfacecolor='r', linestyle='--')
```

Out[162]:
```
[<matplotlib.lines.Line2D at 0x1525381da08>]
```



In [163]:
```
# Optimum k value is 19
final_model = KNeighborsRegressor(n_neighbors=19, metric='euclidean')
final_model.fit(scaled_X_train, y_train)
```

Out[163]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='euclidean',
                    metric_params=None, n_jobs=None, n_neighbors=19, p=2,
                    weights='uniform')
```

In [164]:
```python
# Prediction on  training data
final_train_pred = final_model.predict(scaled_X_train)
final_train_pred
```

Out[164]:
```
array([317091.84210526, 461968.42105263, 541684.21052632, ...,
       521613.31578947, 323257.15789474, 553554.        ])
```

In [165]:
```python
print(r2_score(y_train, final_train_pred))
```

```
0.7917616156763304
```

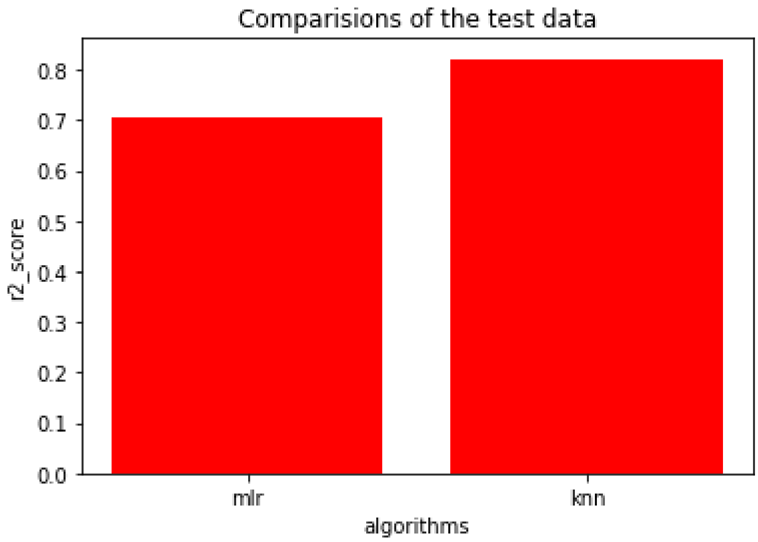
In [166]:
```python
# Predictions on Test Data
final_test_pred = final_model.predict(scaled_X_test)  # y_test
final_test_pred
```

Out[166]:
```
array([ 383250.42105263, 1681789.47368421,  501072.84210526, ...,
        697781.57894737,  431689.47368421,  507549.47368421])
```

In [167]:
```python
print(r2_score(y_test, final_test_pred))
```

```
0.7506223750615134
```

In [168]:
```python
#Comparision of the test data
algorithms=['mlr','knn']
r2_score=[0.7052419493142321,0.8212343397542601]
plt.bar(algorithms,r2_score,color="red")
plt.xlabel("algorithms")
plt.ylabel("r2_score")
plt.title('Comparisions of the test data')
plt.show()
```



In [ ]: