Problem Statement -

Find publicly available data for key factors that influence US home prices nationally. Then,build a data science model that explains how these factors impacted home prices over the last 20years. Use the S&P Case-Schiller Home Price Index as a proxy for home prices

```
import warnings
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score,GridSearchCV


from sklearn.model_selection import train_test_split
from sklearn.metrics import
r2_score,mean_squared_error,mean_absolute_error
```

Data Collection -

Afetr reading so many articles about Home price index in US I came to know that there are two major factor that are affecting home price in Us nationally. Data on factors affecting the housing market's are supply and demand were gathered from publicly available sources (https://fred.stlouisfed.org/). Limited housing supply and high demand can drive up prices, while an oversupply may lead to lower prices

in Supply we got so many factors(Permit.MSACSR,TLRESCONS) that decides the price index for home in US In demand we got factors like (EVACANTUSQ176N,INTDSRUSM193N,UMCSENT,GDP,MORTGAGE15US,MSPUS)

__This datasets contain quarterly data on key supply-demand factors that influence US home prices nationally.

Sources-

- personal income- https://apps.bea.gov/iTable/?reqid=19&step=2&isuri=1&categories=survey&_gl=1*rvamh2*_ga*NTU0NjA4Mjg3LjE3MDIzMTc4ODM.*_ga_J4698JNNFT*MTcwMjMxNzg4My4xLjEuMTcwMjMyMDI4NS4wLjAuMA..#eyJhcHBpZCI6MTksInN0ZXBzIjpbMSwyLDMsM10sImRhdGEiOltbImNhdGVnb3JpZXMiLCJTdXJ2ZXkiXSxbIk5JUEFfVGFibGVfTGlzdCIsIjYwIl0sWyJGaXJzd

F9ZZWFyIiwiMjAwMyJdLFsiTGFzdF9ZZWFyIiwiMjAyMyJdLFsiU2NhbGUiLCItOSJdLFsiU2VyaWVzIiwiUSJdLFsiU2VsZWN0X2FsbF95ZWFycyIsIjEiXV19

- Permit- https://fred.stlouisfed.org/series/PERMIT1
- MSACSR - https://fred.stlouisfed.org/series/MSACSR
- TLESCONS-https://fred.stlouisfed.org/series/TLRESCONS
- EVACANTUSQ176N-https://fred.stlouisfed.org/series/EVACANTUSQ176N
- INTDSRUSM193N- https://fred.stlouisfed.org/series/INTDSRUSM193N
- UMCSENT - https://fred.stlouisfed.org/series/UMCSENT
- GDP - https://fred.stlouisfed.org/series/GDP
- MORTGAGE15US- https://fred.stlouisfed.org/series/MORTGAGE15US
- MSPUS- https://fred.stlouisfed.org/series/MSPUS

All this data are from 2003 to 2023 (20 years)

Data Preparation-

- After Collecting all the data from the source I consolidated the data into a single a file (homellcdata.csv)

Accesing the data and converting it ino Dataframe

```
df=pd.read_csv('homellcdata.csv')
df.head()

      year  Personal Income(in $million)       Permit    MSACSR
TLRESCONS  \
0  2003-Q1                          5.03  1377.333333  4.200000
421328.6667
1  2003-Q2                          5.10  1413.666667  3.833333
429308.6667
2  2003-Q3                          5.17  1510.666667  3.633333
458890.0000
3  2003-Q4                          5.26  1542.666667  3.966667
491437.3333
4  2004-Q1                          5.27  1583.666667  3.700000
506856.3333

   EVACANTUSQ176N  INTDSRUSM193N    UMCSENT        GDP  MORTGAGE15US
MSPUS  \
0           14908       2.250000  79.966667  11174.129      5.204615
186000
1           15244       2.166667  89.266667  11312.766      4.867692
191800
```

```
2          15614       2.000000  89.300000  11566.669      5.356923
191900
3          15654       2.000000  91.966667  11772.234      5.256154
198800
4          15895       2.000000  98.000000  11923.447      4.872727
212700

   price_index
0   129.321333
1   131.756000
2   135.013333
3   138.834667
4   143.299000
```

```
df.columns
```

```
Index(['year', 'Personal Income(in $million)', 'Permit', 'MSACSR',
'TLRESCONS',
       'EVACANTUSQ176N', 'INTDSRUSM193N', 'UMCSENT', 'GDP',
'MORTGAGE15US',
       'MSPUS', 'price_index'],
      dtype='object')
```

Column Description:-

- year :- this column has quarter for each year from 2003-2023(20 years data)
- Personal Income(in million) :- Income levels and employment rates are key indicators of a population's ability to afford housing. Higher incomes generally support higher home prices
- Permit : This variable represents the number of new housing units authorized for construction in permit-issuing places.
- MSACSR(Monthly Supply of New Houses in the United States)- It indicates the monthly supply of new houses available in the US
- TLRESCONS(Total Construction Spending):- This variable represents the total construction spending on residential projects.
- EVACANTUSQ176N - It provides an estimate of the number of vacant housing units in the United States
- INTDSRUSM193N - This column represents the interest rates or discount rates for the United States.
- UMCSENT(University of Michigan: Consumer Sentiment)- It measures the consumer sentiment index based on surveys conducted by the University of Michigan.
- GDP- Gross Domestic Product
- MORTGAGE15US - It indicates the average interest rate for a 30-year fixed-rate mortgage.
- MSPUS - Median Sales Price of Houses Sold for the United States
- price_index - This variable serves as a proxy for home prices and represents the home price index for the United States.This is our label for this dataset

```python
# Checking the shape
print("We have {} Rows and {} Columns in our
dataframe".format(df.shape[0], df.shape[1]))
df.head()
```

```
We have 83 Rows and 12 Columns in our dataframe

      year  Personal Income(in $million)       Permit    MSACSR
TLRESCONS  \
0  2003-Q1                          5.03  1377.333333  4.200000
421328.6667
1  2003-Q2                          5.10  1413.666667  3.833333
429308.6667
2  2003-Q3                          5.17  1510.666667  3.633333
458890.0000
3  2003-Q4                          5.26  1542.666667  3.966667
491437.3333
4  2004-Q1                          5.27  1583.666667  3.700000
506856.3333

   EVACANTUSQ176N  INTDSRUSM193N    UMCSENT        GDP  MORTGAGE15US
MSPUS  \
0           14908       2.250000  79.966667  11174.129      5.204615
186000
1           15244       2.166667  89.266667  11312.766      4.867692
191800
2           15614       2.000000  89.300000  11566.669      5.356923
191900
3           15654       2.000000  91.966667  11772.234      5.256154
198800
4           15895       2.000000  98.000000  11923.447      4.872727
212700

   price_index
0   129.321333
1   131.756000
2   135.013333
3   138.834667
4   143.299000
```

```python
#checking for null values
df.isnull().sum()
```

```
year                            0
Personal Income(in $million)    0
Permit                          0
MSACSR                          0
TLRESCONS                       0
EVACANTUSQ176N                  0
INTDSRUSM193N                   8
```

```
UMCSENT                             0
GDP                                 0
MORTGAGE15US                        0
MSPUS                               0
price_index                         0
dtype: int64
```

```python
 ## Looks like we got only 8 null values in the INTDSRUSM193N column lets check its statstic for filling the values
df['INTDSRUSM193N'].describe()
```

```
count    75.000000
mean      1.938889
std       1.732448
min       0.250000
25%       0.750000
50%       1.000000
75%       2.541667
max       6.250000
Name: INTDSRUSM193N, dtype: float64
```

```python
# Since the data is Continous so Filled the null value susing mean method
df['INTDSRUSM193N']=df['INTDSRUSM193N'].fillna(df['INTDSRUSM193N'].mean())
```

```python
df['INTDSRUSM193N'].isnull().sum()
```

```
0
```

```python
df.isnull().sum()
```

```
year                                0
Personal Income(in $million)        0
Permit                              0
MSACSR                              0
TLRESCONS                           0
EVACANTUSQ176N                      0
INTDSRUSM193N                       0
UMCSENT                             0
GDP                                 0
MORTGAGE15US                        0
MSPUS                               0
price_index                         0
dtype: int64
```

```python
#Cheking info()- This will give Index, Datatype and Memory information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83 entries, 0 to 82
```

```
Data columns (total 12 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   year                          83 non-null     object 
 1   Personal Income(in $million)  83 non-null     float64
 2   Permit                        83 non-null     float64
 3   MSACSR                        83 non-null     float64
 4   TLRESCONS                     83 non-null     float64
 5   EVACANTUSQ176N                83 non-null     int64  
 6   INTDSRUSM193N                 83 non-null     float64
 7   UMCSENT                       83 non-null     float64
 8   GDP                           83 non-null     float64
 9   MORTGAGE15US                  83 non-null     float64
 10  MSPUS                         83 non-null     int64  
 11  price_index                   83 non-null     float64
dtypes: float64(9), int64(2), object(1)
memory usage: 7.9+ KB
```

So we from above information we can see that we got 83 non null entries in our datset that means we have no null or missing values in our dataset also with 11 float and 1 object datatype.

```
df.describe()
```

```
       Personal Income(in $million)       Permit     MSACSR  
TLRESCONS  \
count                     83.000000    83.000000  83.000000  
83.000000   
mean                       7.635663   889.494980   6.216064  
505259.248992   
std                        1.802239   381.738388   1.898726  
190344.432418   
min                        5.030000   358.333333   3.366667  
246953.333300   
25%                        6.310000   619.000000   4.950000  
359693.166700   
50%                        7.100000   802.666667   5.633333  
506856.333300   
75%                        8.905000  1119.166667   7.483333  
591055.500000   
max                       11.900000  1745.333333  11.400000  
973236.666700   

       EVACANTUSQ176N  INTDSRUSM193N    UMCSENT           GDP  
MORTGAGE15US  \
count       83.000000      83.000000  83.000000     83.000000  
83.000000   
mean     17051.216867       1.938889  81.759036  17617.994120  
4.134555   
std       1397.893677       1.645770  12.521620   4191.128485  
```

```
1.233405
min      13876.000000       0.250000  56.100000  11174.129000
2.171429
25%      15785.500000       0.750000  72.983333  14506.499500
3.126401
50%      17258.000000       1.583333  82.833333  16728.687000
3.844615
75%      18198.000000       2.291667  92.766667  20454.732500
5.284560
max      19137.000000       6.250000  98.933333  27644.463000
6.396154

               MSPUS  price_index
count      83.000000    83.000000
mean   284472.289157   185.081968
std     70648.570858    46.702384
min    186000.000000   129.321333
25%    228450.000000   148.223667
50%    264800.000000   174.580000
75%    321500.000000   202.544500
max    479500.000000   309.032333
```

Observatons-

Using the describe method I can see the count, mean, standard deviation, minimum, maximum and inter quantile values of our dataset.

As per my observation:

there is no sign of skewness in few columns that we check later on

Exploratory Data Analysis

```
df.columns

Index(['year', 'Personal Income(in $million)', 'Permit', 'MSACSR',
'TLRESCONS',
       'EVACANTUSQ176N', 'INTDSRUSM193N', 'UMCSENT', 'GDP',
'MORTGAGE15US',
       'MSPUS', 'price_index'],
      dtype='object')
```

Univariate Analysis

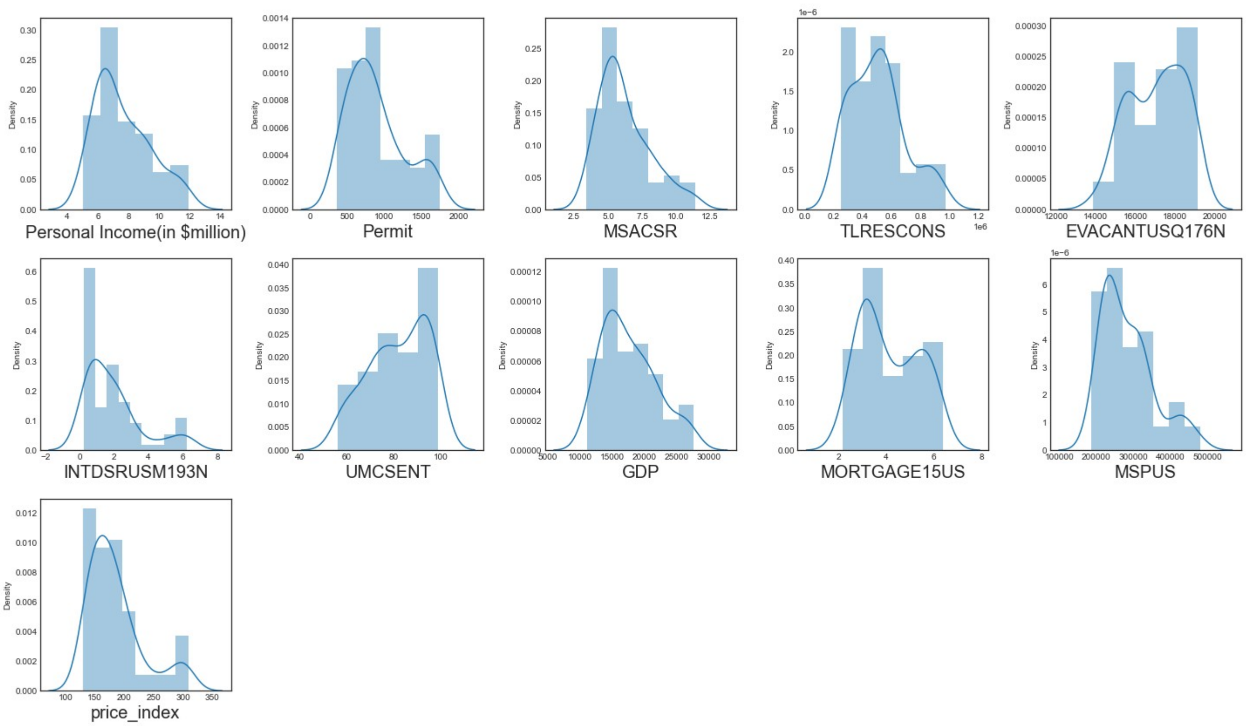
Exploring continus columns

```
## COntinous column
numerical_cols= []
for x in df.dtypes.index:
    if df.dtypes[x] == 'float64' or df.dtypes[x] == 'int64':
```

```
        numerical_cols.append(x)
print(f"\nNumber Data Type Columns are:\n", numerical_cols)
```

```
Number Data Type Columns are:
 ['Personal Income(in $million)', 'Permit', 'MSACSR', 'TLRESCONS',
'EVACANTUSQ176N', 'INTDSRUSM193N', 'UMCSENT', 'GDP', 'MORTGAGE15US',
'MSPUS', 'price_index']
```

```
plt.figure(figsize=(20,15),facecolor='white')
plot_number=1

for column in df[numerical_cols]:
    if plot_number<=20:
        ax=plt.subplot(4,5,plot_number)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=20)

    plot_number +=1


plt.tight_layout()
```



- Except price_index(label column) we see some skewness in permit,MSACSR,TLRESCONS ,INTDSRUSM193N,GDP,MSPUS columnslets check wit .ske method to confirm the skewness

```
df[numerical_cols].skew()
```

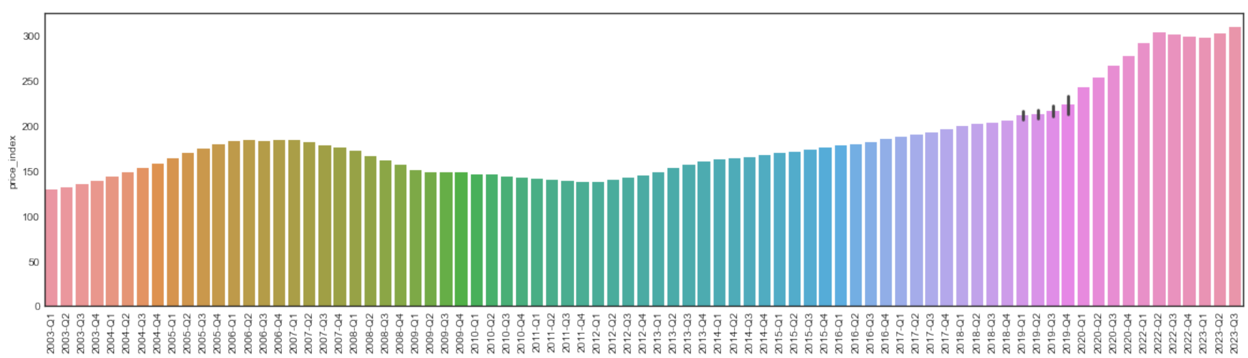
```
Personal Income(in $million)    0.693947
Permit                          0.745394
MSACSR                          0.923787
TLRESCONS                       0.590446
EVACANTUSQ176N                 -0.246561
INTDSRUSM193N                   1.394812
UMCSENT                        -0.397624
GDP                             0.625702
MORTGAGE15US                    0.269950
MSPUS                           1.000970
price_index                     1.326674
dtype: float64
```

With the skew method we see that there are columns present in our dataset that are above the acceptable range of +/-0.5 value. Having said that we will treat the skewness that is present in our continous data columns later in the project.

Bivariate Analyis

1- year VS price_index

```
plt.figure(figsize=(20, 5))
sns.barplot(x='year', y='price_index',data=df)
plt.xticks(rotation='90')
#plt.savefig("1.jpg")
plt.show()
```

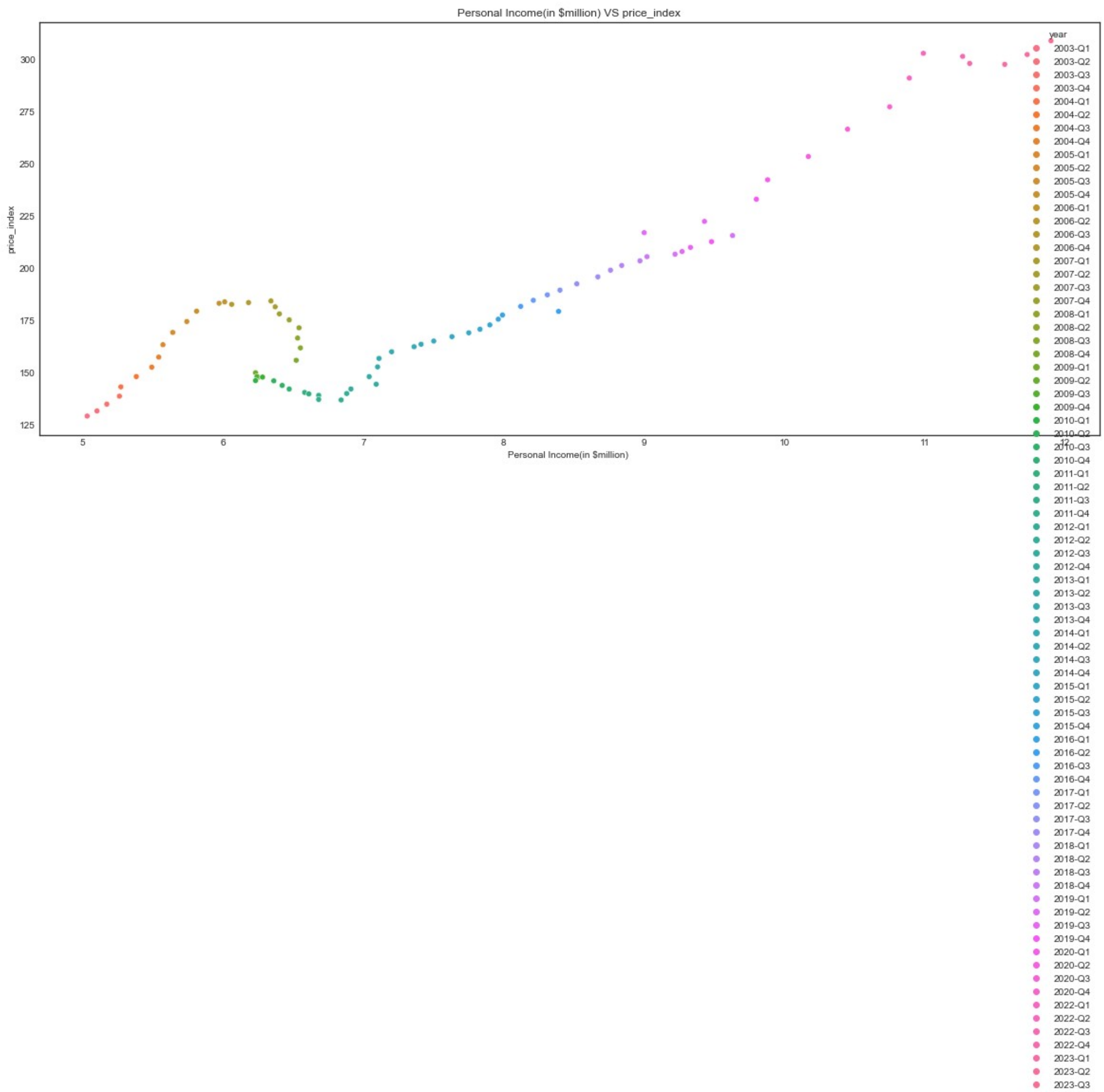


This bar plot clerly show that price index is increasing from 2003 although it shows some downfalll in the year 2012 but after 2012 it keeps on increasing

2- Personal Income(in $million) VS price_index

```
plt.figure(figsize=(20,8),facecolor='white')
sns.scatterplot(x='Personal Income(in $million)',
y='price_index',data=df,hue='year')
# Add labels and title
plt.xlabel('Personal Income(in $million) ')
plt.ylabel('price_index')
```

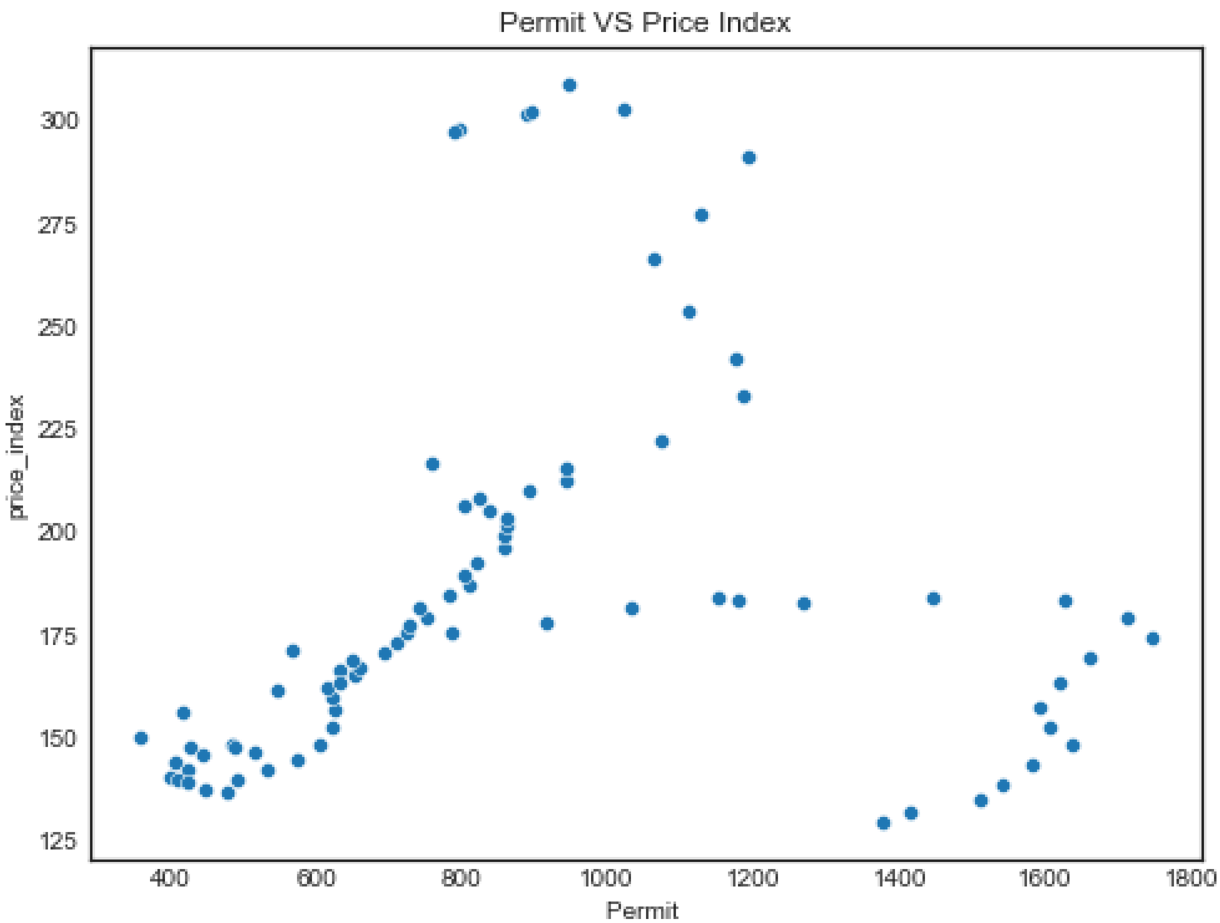
```python
plt.title('Personal Income(in $million) VS price_index')


# Show the plot
plt.show()
```



- **in the year 2009-2010 we see some downfall in the price index of the house but after that price and income are kept on increasing**

3- Permit VS price_index

```
plt.figure(figsize=(8,6),facecolor='white')
sns.scatterplot(x='Permit', y='price_index',data=df)
# Add labels and title
plt.xlabel('Permit ')
plt.ylabel('price_index')
plt.title('Permit VS Price Index')



# Show the plot
plt.show()
```

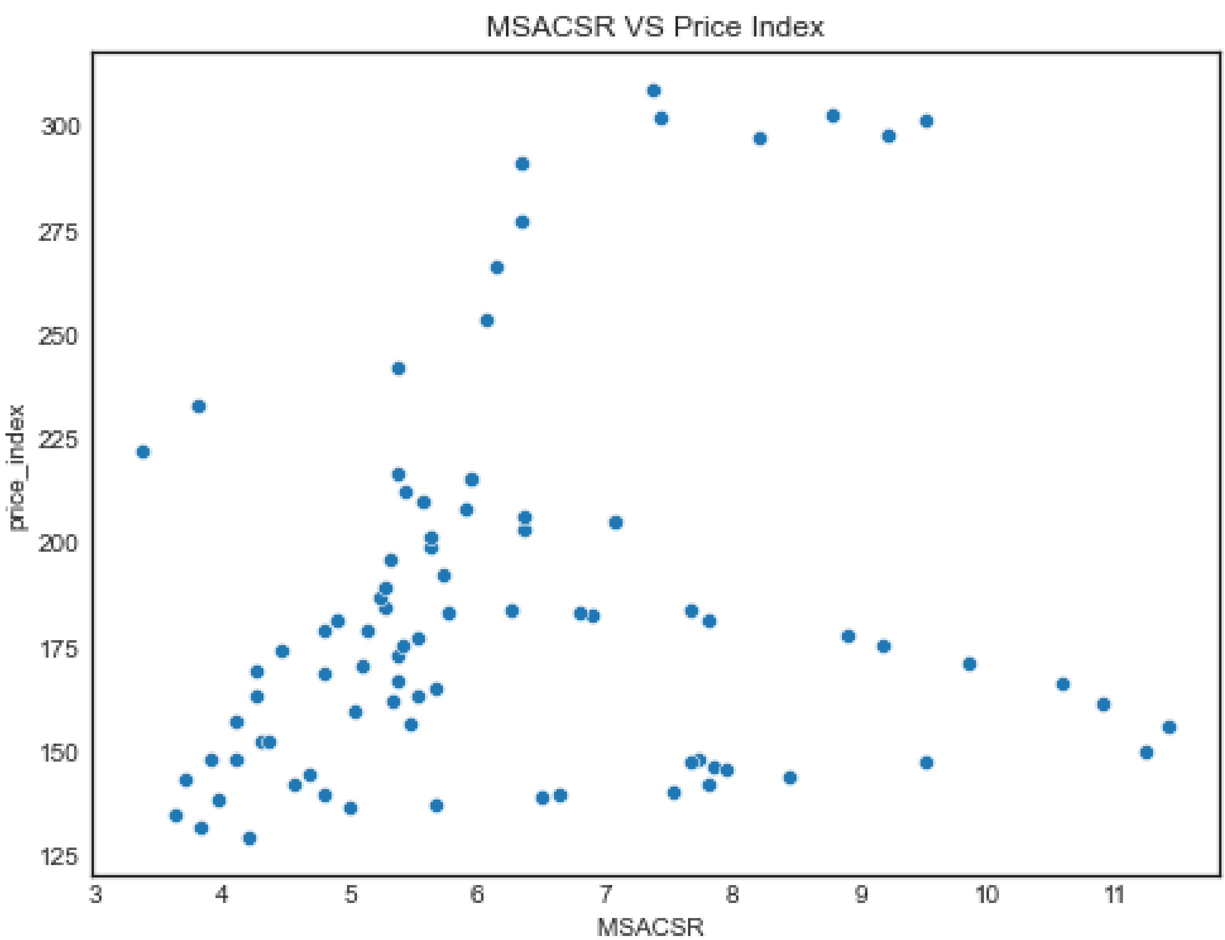


- PriceIndex are higher when Permit is in range between 800-1000 units that means when there is limited supply price definietly will go higher but when there is higher supply its definately has negative impact on price index

3- MSACSR VS price Index

```
plt.figure(figsize=(8,6),facecolor='white')
sns.scatterplot(x='MSACSR', y='price_index',data=df)
```

```python
# Add labels and title
plt.xlabel('MSACSR ')
plt.ylabel('price_index')
plt.title('MSACSR VS Price Index')



# Show the plot
plt.show()
```

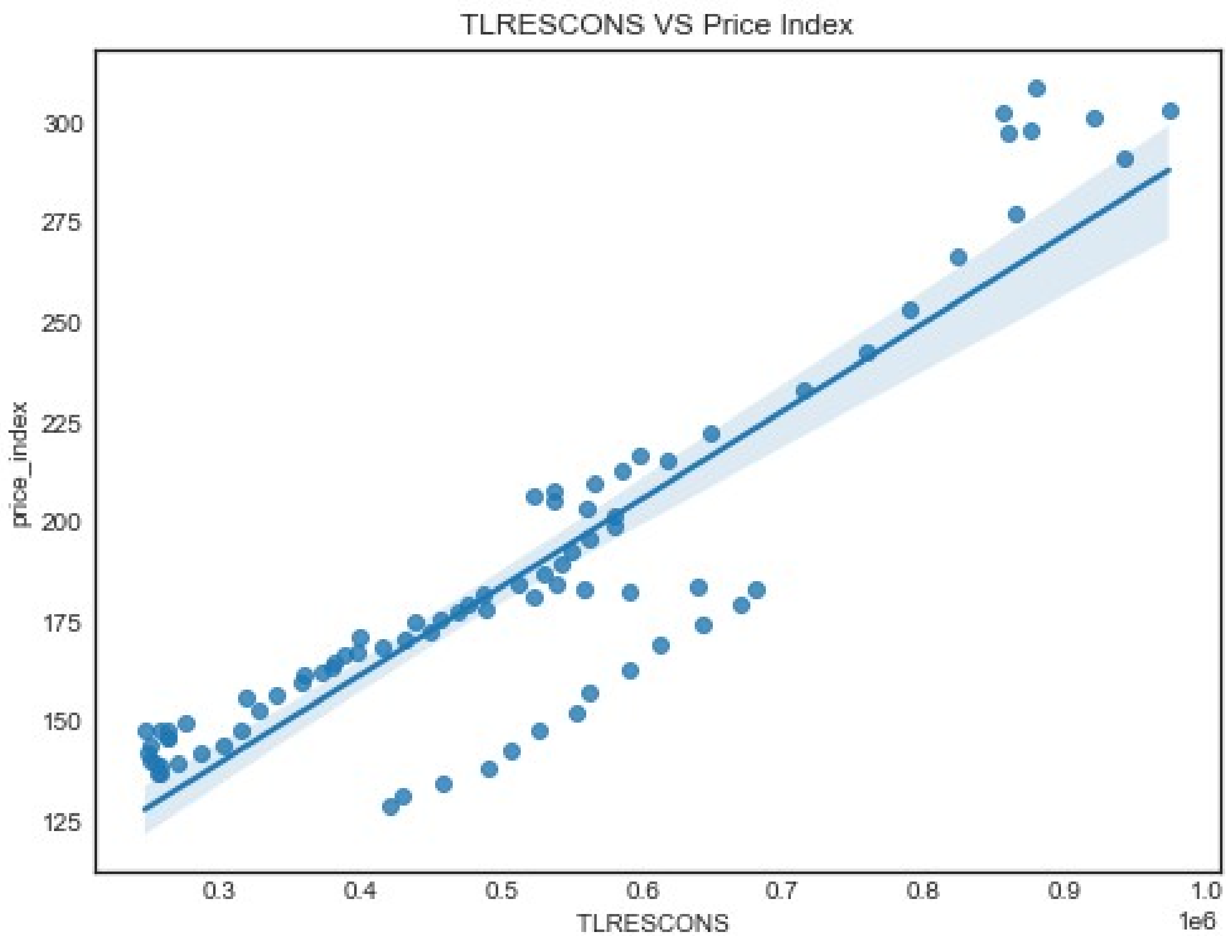


- There is a weak positive relationship between the monthly supply of new houses and price index. This suggests that as the supply of new houses increases, it may have a slight positive impact on home prices.

4- TLRESCONS VS price Index

```python
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='TLRESCONS', y='price_index',data=df)
# Add labels and title
plt.xlabel('TLRESCONS ')
plt.ylabel('price_index')
```

```
plt.title('TLRESCONS VS Price Index')


# Show the plot
plt.show()
```

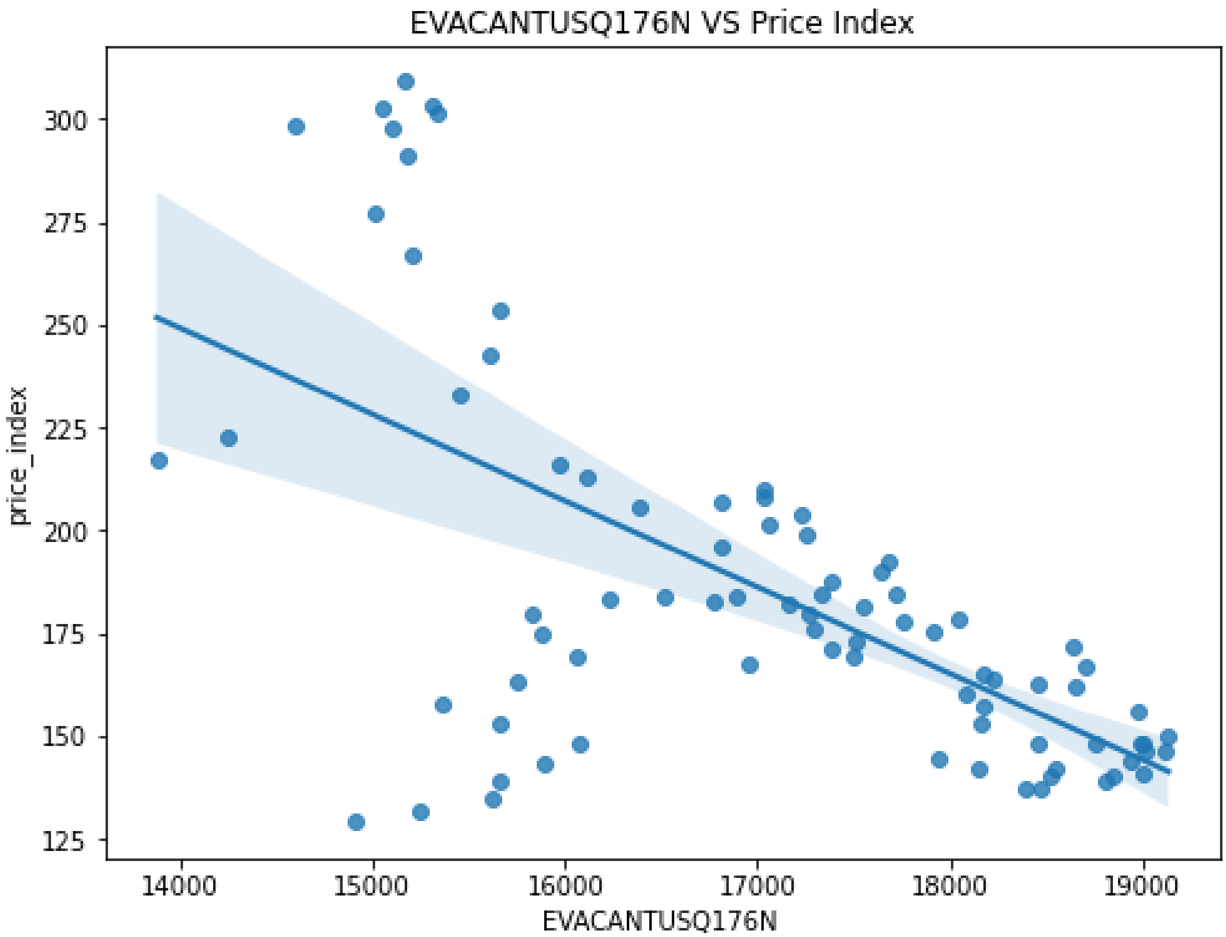


- TLRESCONS(Total Construction Spending: Residential) is directly imapcting on the Price index in above graph means its highly positively related with price index column.This suggests that higher construction spending is strongly associated with higher home prices.

5- EVACANTUSQ176N VS price_index -

```
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='EVACANTUSQ176N', y='price_index',data=df)
# Add labels and title
plt.xlabel('EVACANTUSQ176N ')
plt.ylabel('price_index')
plt.title('EVACANTUSQ176N VS Price Index')
```

```
# Show the plot
plt.show()
```

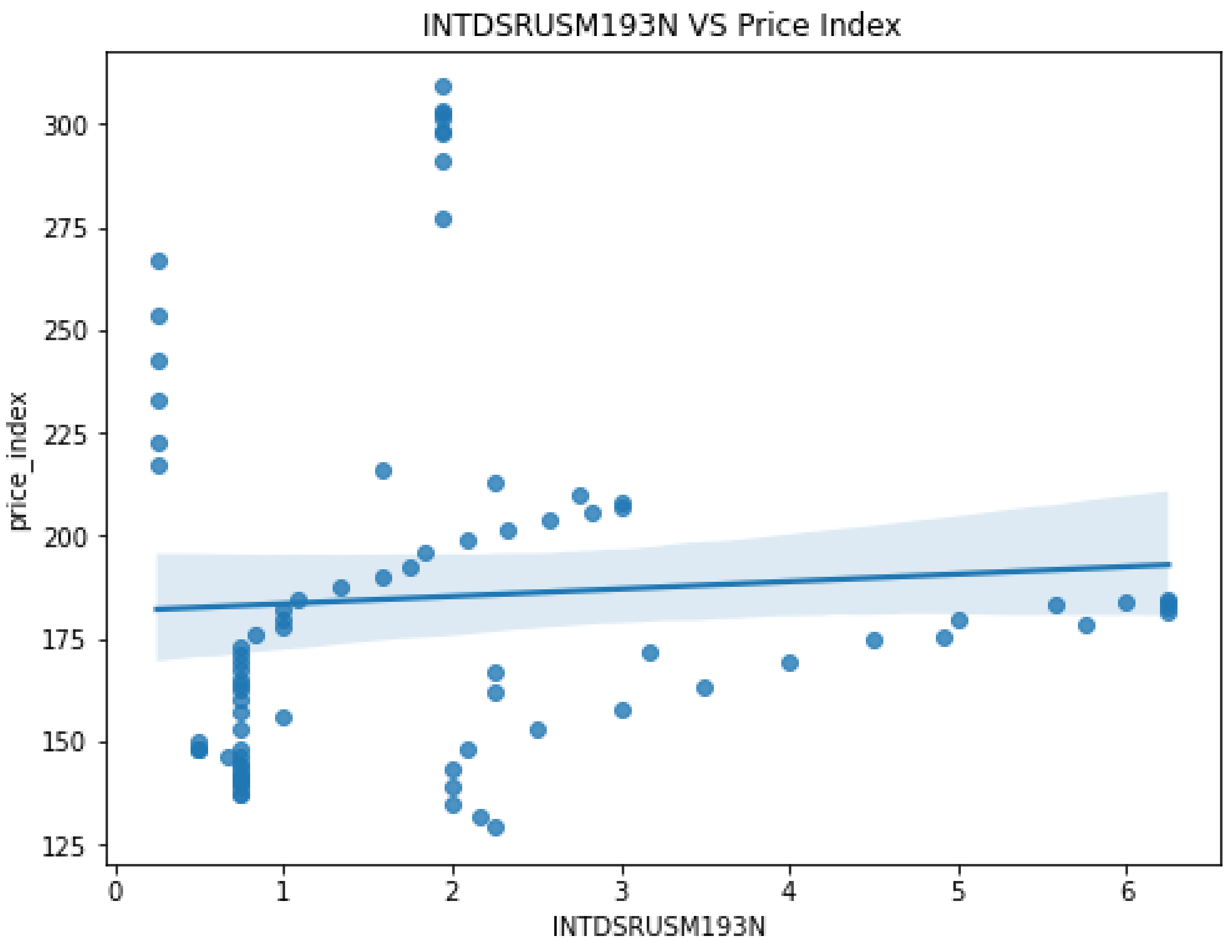


- EVACANTUSQ176N (Housing Inventory Estimate: Vacant Housing Units):This indicates that a higher number of vacant housing units may exert downward pressure on home prices.its negatively correlated with Price index

6- INTDSRUSM193N VS price_index

```
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='INTDSRUSM193N', y='price_index',data=df)
# Add labels and title
plt.xlabel('INTDSRUSM193N ')
plt.ylabel('price_index')
plt.title('INTDSRUSM193N VS Price Index')
```

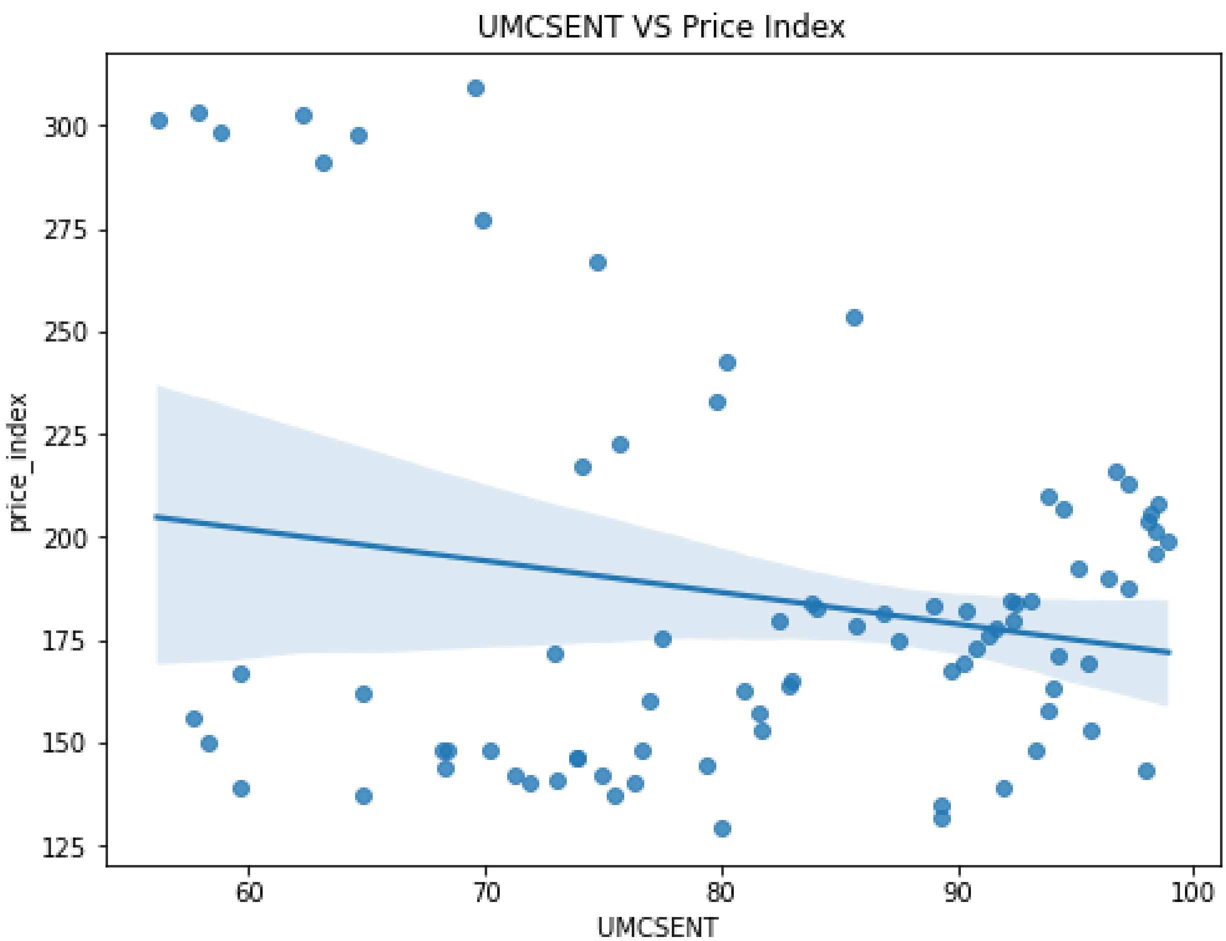
```
# Show the plot
plt.show()
```



- This is simple when interst rate are high definietly home price will be less . it less positively related to home price index

7-UMCSENT VS price_index

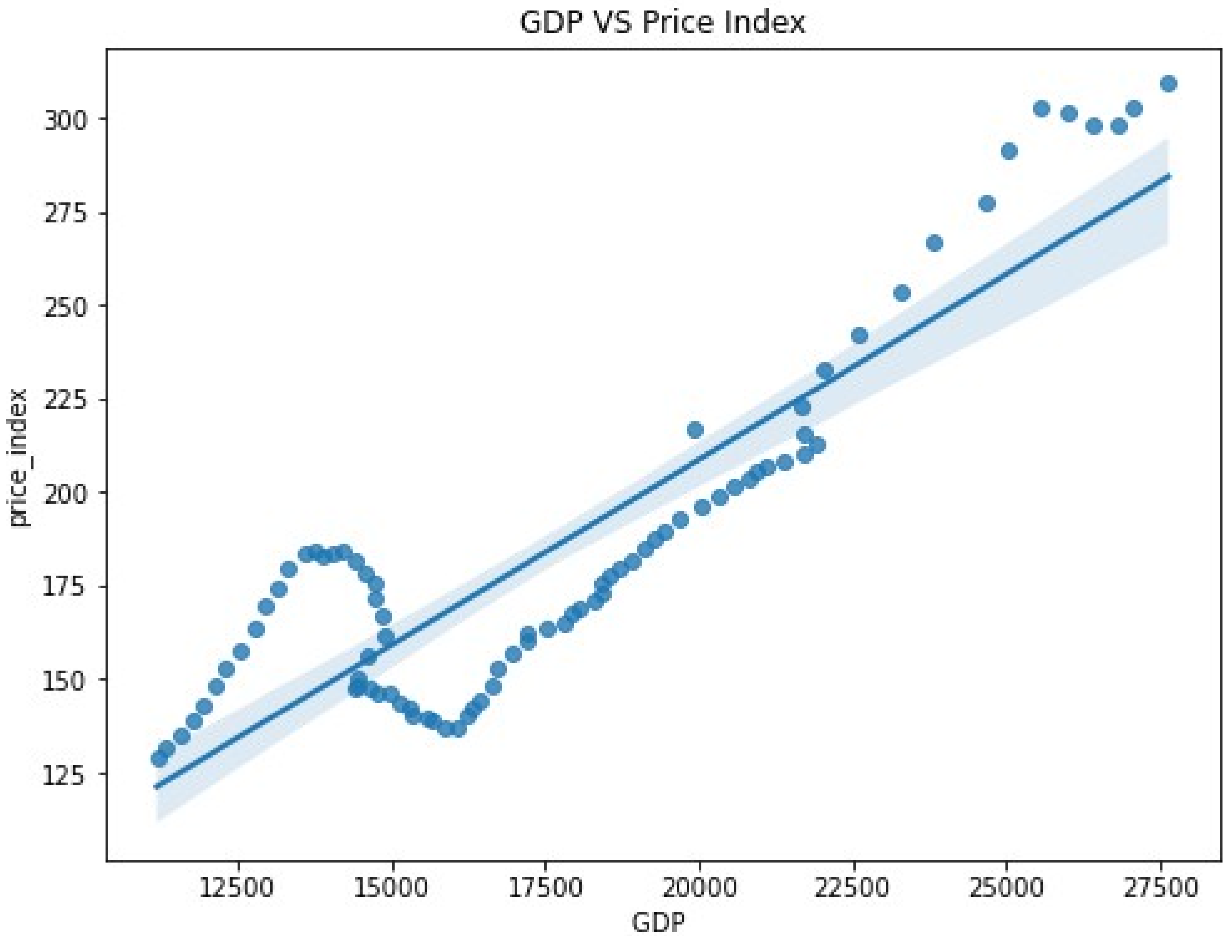
```
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='UMCSENT', y='price_index',data=df)
# Add labels and title
plt.xlabel('UMCSENT ')
plt.ylabel('price_index')
plt.title('UMCSENT VS Price Index')



# Show the plot
plt.show()
```



- Lower consumer sentiment is associated with slightly lower home prices. Means when Consumer are more confident with the ecomnomy they will not hesitate to pay high price for the house

7- GDP VS price_index

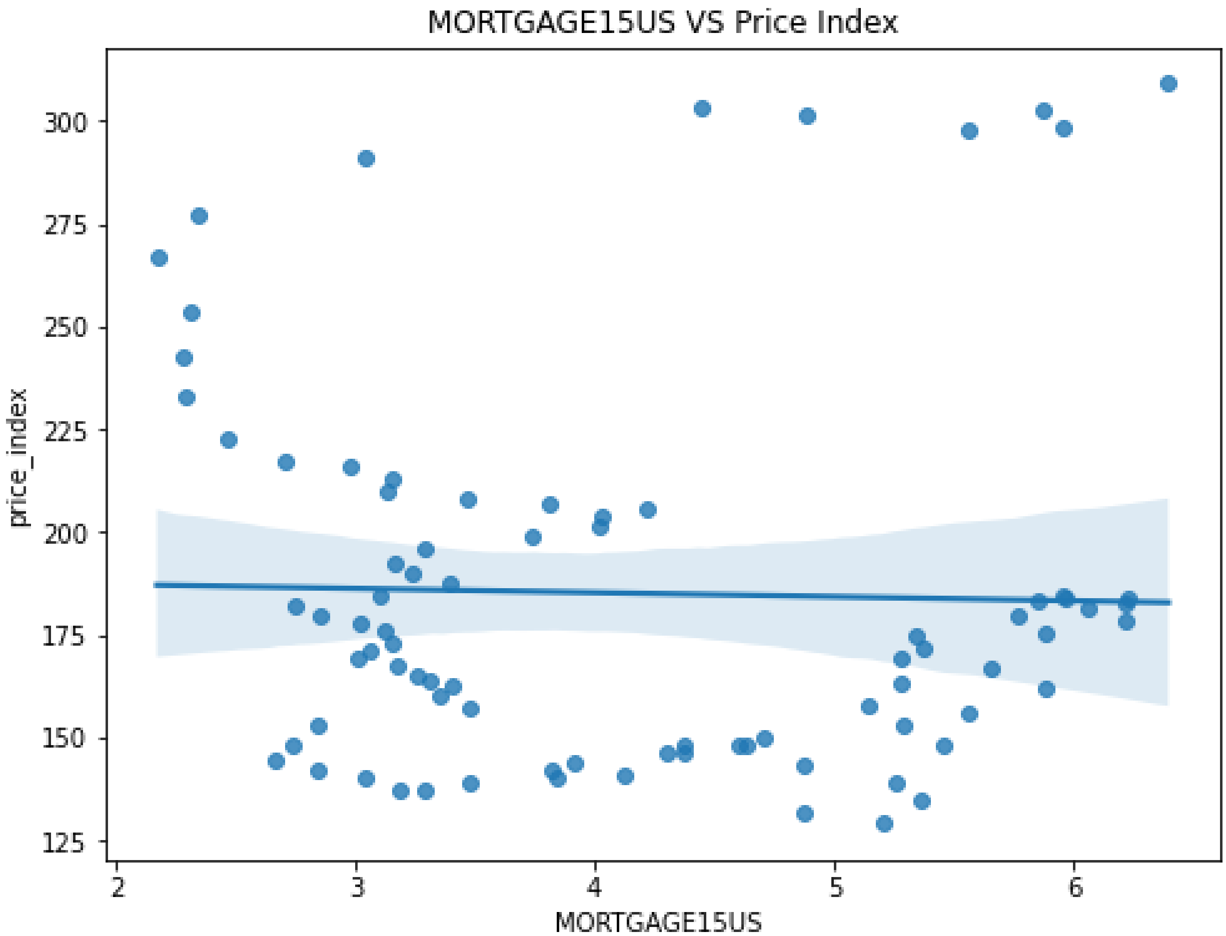
```
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='GDP', y='price_index',data=df)
# Add labels and title
plt.xlabel('GDP ')
plt.ylabel('price_index')
plt.title('GDP VS Price Index')



# Show the plot
plt.show()
```



- GDP(Gross Domestic Product) is Highly related to price_index means as soon as GDP is high Price index will always be higher

8-MORTGAGE15US VS price_index

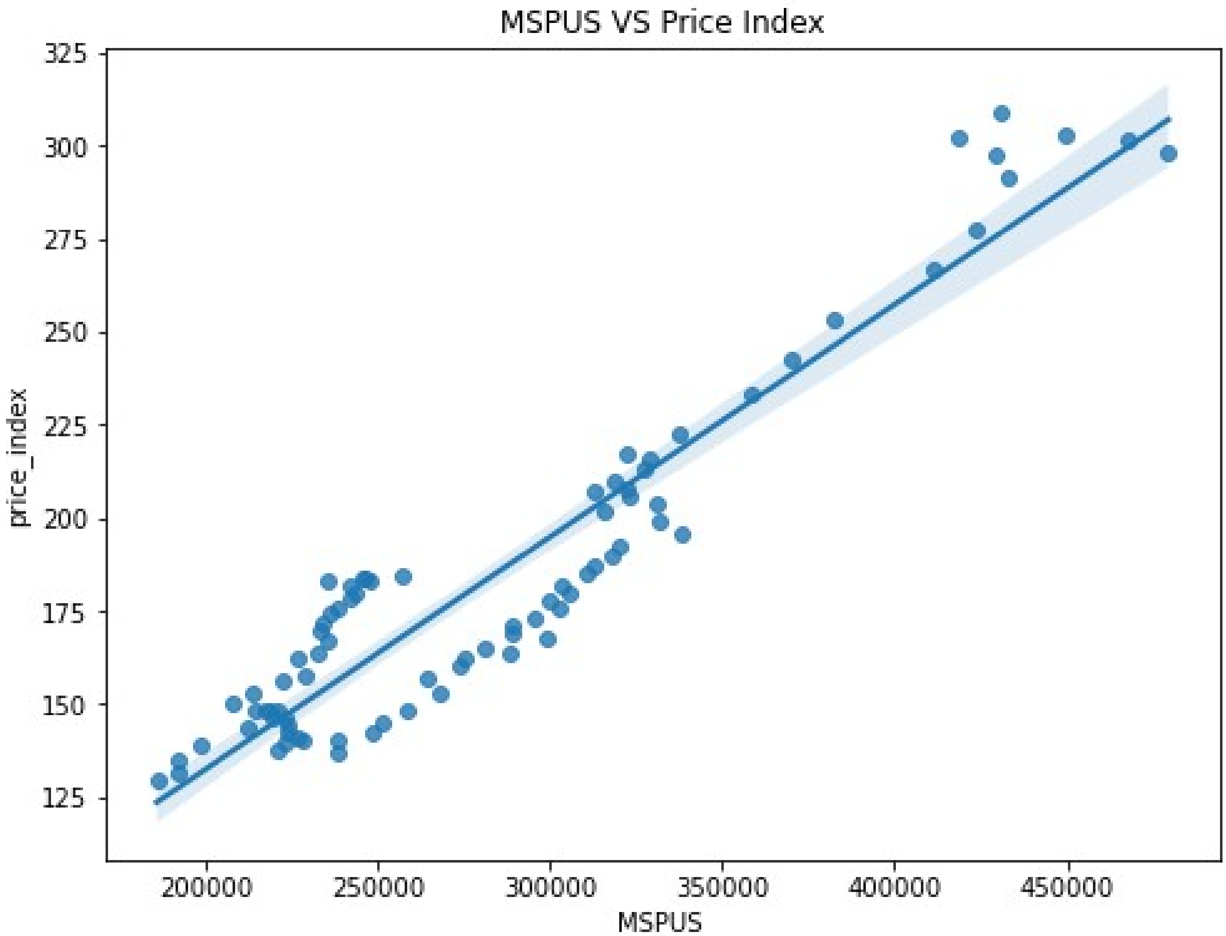
```
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='MORTGAGE15US', y='price_index',data=df)
# Add labels and title
plt.xlabel('MORTGAGE15US ')
plt.ylabel('price_index')
plt.title('MORTGAGE15US VS Price Index')



# Show the plot
plt.show()
```



- Home Price Index is decrasinf as soon as average interest rate is increasing there is decline in home price index

9- MSPUS VS price_index

```
plt.figure(figsize=(8,6),facecolor='white')
sns.regplot(x='MSPUS', y='price_index',data=df)
# Add labels and title
plt.xlabel('MSPUS ')
plt.ylabel('price_index')
plt.title('MSPUS VS Price Index')



# Show the plot
plt.show()
```

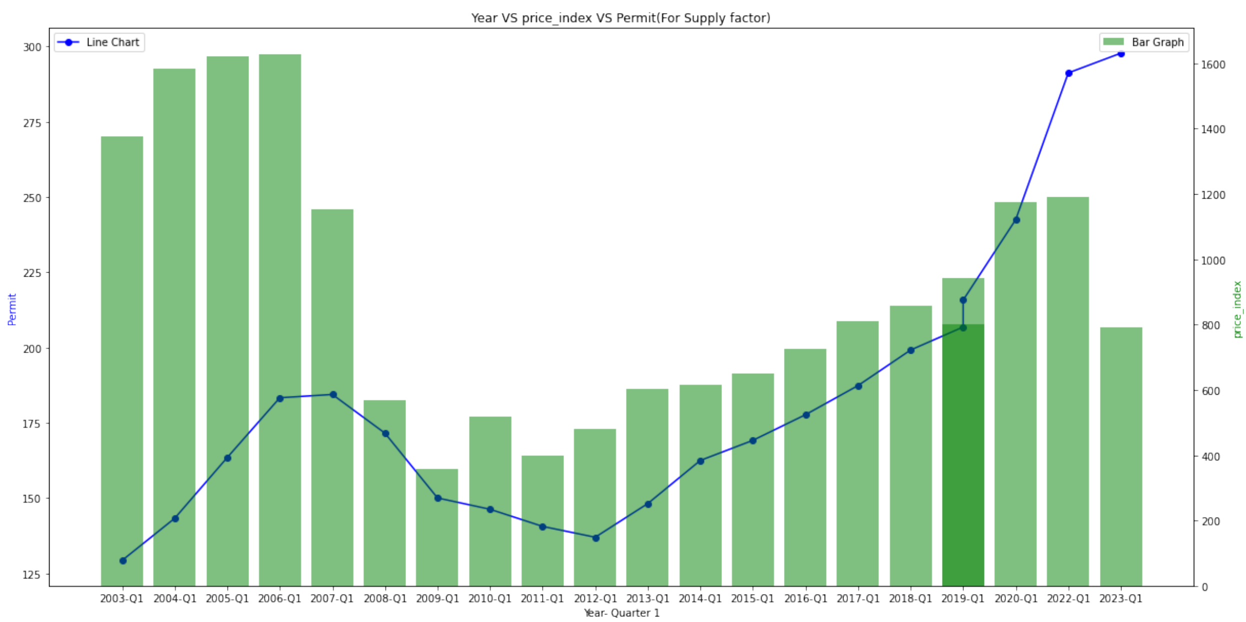


- MSPUS is strongly related with price index we can see in above regression plot

Multivariate Analysis

```
#creating different datframe for each Quarter i.e Q1,Q2,Q3,Q4

df_first_quarter = df[df['year'].apply(lambda x: x.endswith('-Q1'))]
df_second_quarter = df[df['year'].apply(lambda x: x.endswith('-Q2'))]
df_third_quarter = df[df['year'].apply(lambda x: x.endswith('-Q3'))]
df_fourth_quarter = df[df['year'].apply(lambda x: x.endswith('-Q4'))]


### Q1 of each year and Permit VS price_index

# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')
```

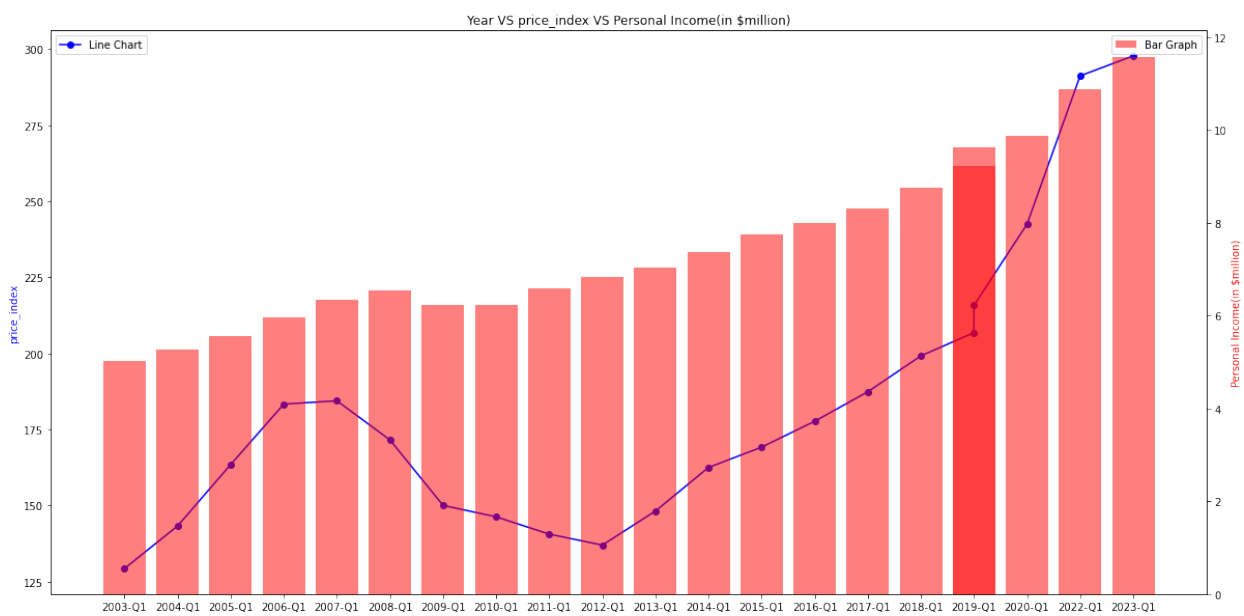
```python
# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['Permit'],
alpha=0.5, color='green', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('Permit', color='blue')
ax2.set_ylabel('price_index', color='green')
plt.title('Year VS price_index VS Permit(For Supply factor)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



- (PERMIT) is an economic factor that measures the number of new privately-owned housing units authorized by building permits in permit-issuing places. It is used to gauge the strength of the housing market and the overall economy. The issuance of residential building permits can be a barometer for consumer confidence and solvency.

- The number of new privately-owned housing units authorized has a moderate positive correlation with home prices. This suggests that the approval of the construction of more housing units tends to raise home prices. This is because a decrease in the supply of homes, workers and material causes an increase in price.

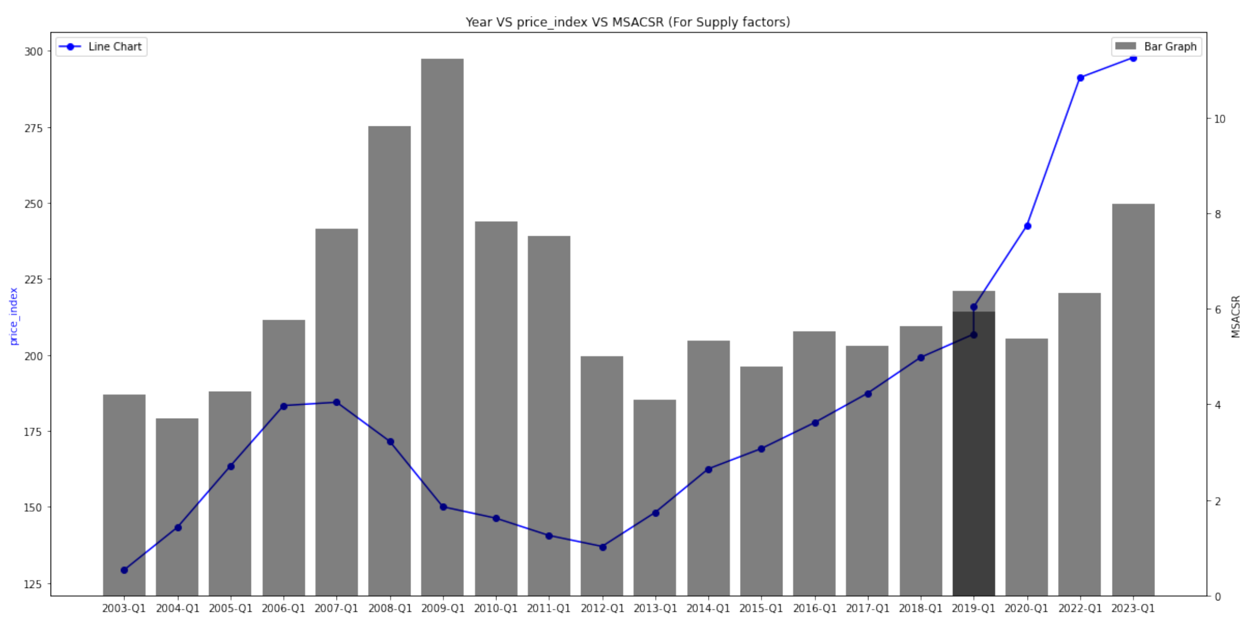
```python
# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')

# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['Personal Income(in
$million)'], alpha=0.5, color='red', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('price_index', color='blue')
ax2.set_ylabel('Personal Income(in $million)', color='red')
plt.title('Year VS price_index VS Personal Income(in $million)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



- We have also seen previuosly that personal Income has a big impact on price_index. if People of US has more income they will surely invest in better project which surely increase the price index for house

Q1 VS MSACSR VS price_index

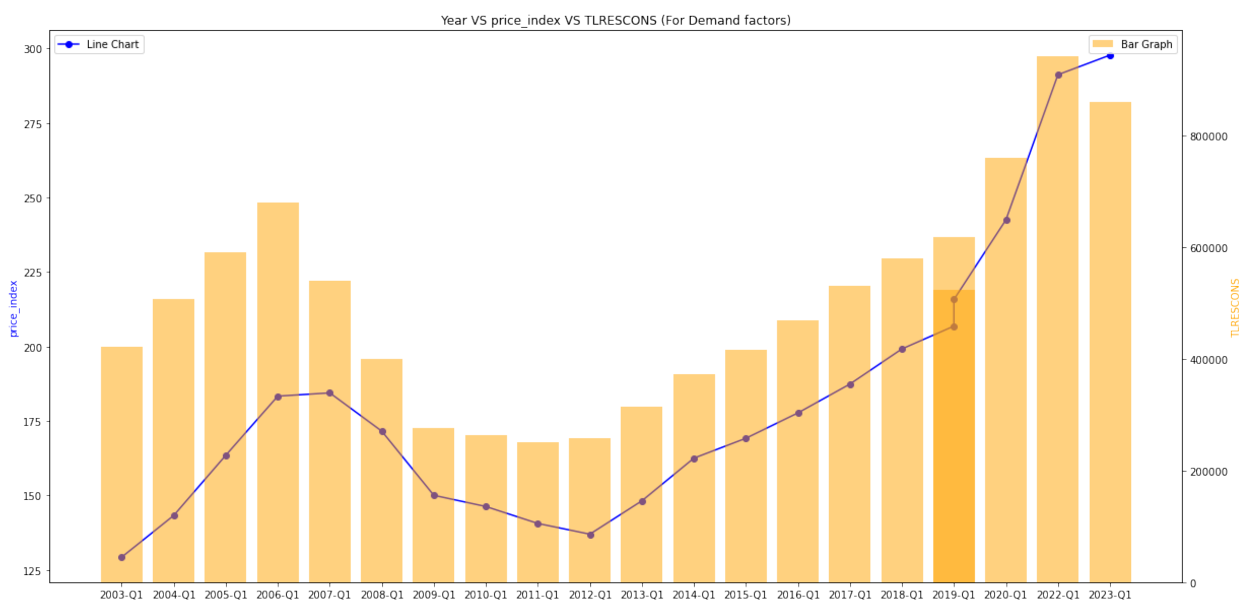
```
# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')

# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['MSACSR'],
alpha=0.5, color='black', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('price_index', color='blue')
ax2.set_ylabel('MSACSR', color='black')
plt.title('Year VS price_index VS MSACSR (For Supply factors)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



The monthly supply of new houses has a negative correlation with home prices. This means that increase in MSACSR could lead to a decrease in Price Index. This is because an increase in supply of new houses could lead to a decrease in demand which could lead to a decrease in prices.

Q1 VS price_index_VS TLRESCONS

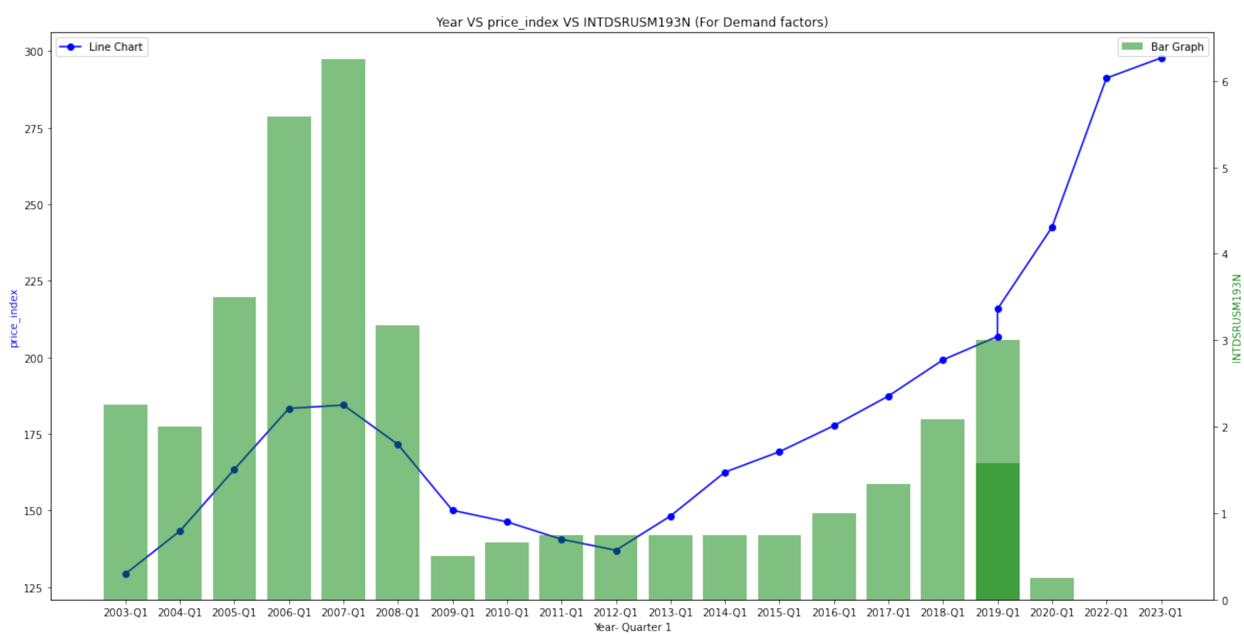
```
# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')

# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['TLRESCONS'],
alpha=0.5, color='orange', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('price_index', color='blue')
ax2.set_ylabel('TLRESCONS', color='orange')
plt.title('Year VS price_index VS TLRESCONS (For Demand factors)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



- As long as TLRESCONS(Total Cost spending on construction) keeps on icreasing price_index will also be incraesed.The reason for this is simple: construction costs include building materials, labor, and other charges. This raises overall house costs.

Q1 VS price_index VS INTDSRUSM193N

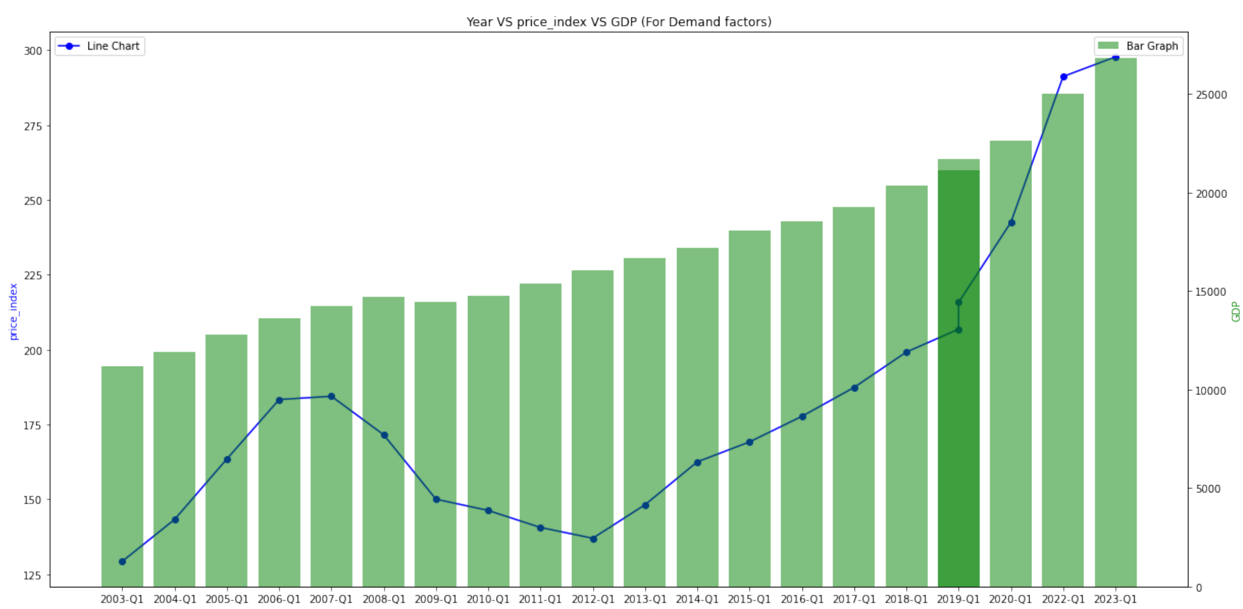
```
# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')

# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['INTDSRUSM193N'],
alpha=0.5, color='green', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('price_index', color='blue')
ax2.set_ylabel('INTDSRUSM193N', color='green')
plt.title('Year VS price_index VS INTDSRUSM193N (For Demand factors)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



- INTDSRUSM193N-interest rates or discount rates are weakly realted to home prices that means highr the interest rate lower the price index

Q1 VS price_index VS GDP

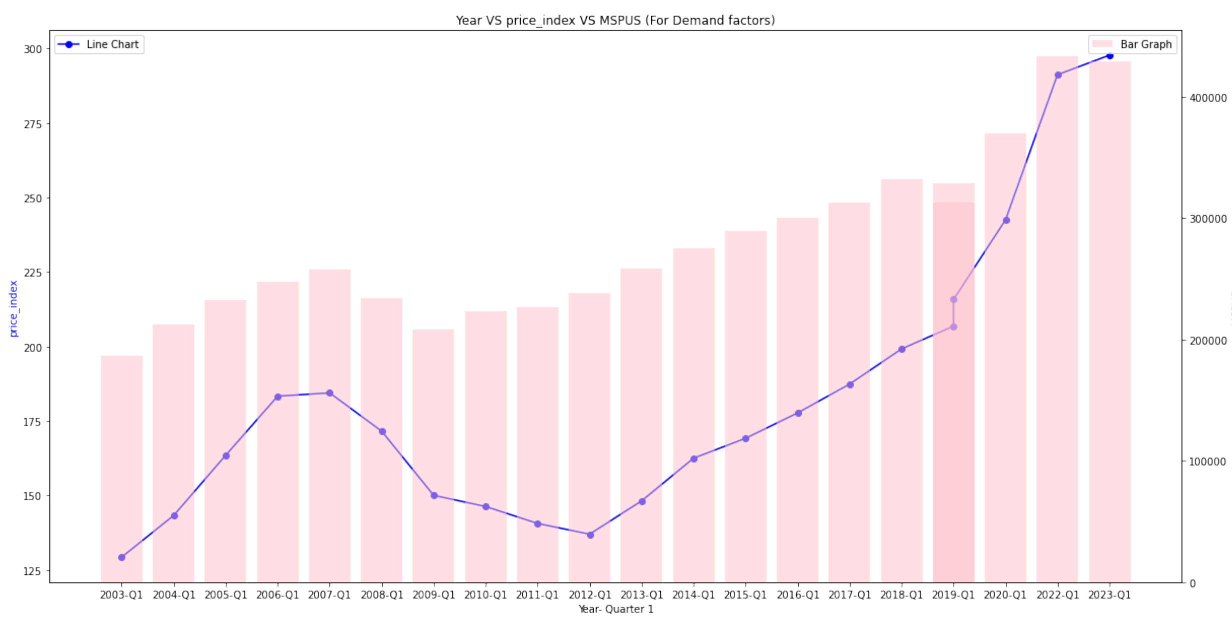
```
# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')

# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['GDP'], alpha=0.5,
color='green', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('price_index', color='blue')
ax2.set_ylabel('GDP', color='green')
plt.title('Year VS price_index VS GDP (For Demand factors)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



- As long as GDP is higher Price index will also rises.its strongly realted to Price_index

```
#### Q1 VS price_index VS MSPUS
```

```
# Create a figure and axis

fig, ax1 = plt.subplots(figsize=(20,10))

# Plot the line chart
ax1.plot(df_first_quarter['year'], df_first_quarter['price_index'],
color='blue', marker='o', label='Line Chart')

# Create a second y-axis for the bar graph
ax2 = ax1.twinx()
ax2.bar(df_first_quarter['year'], df_first_quarter['MSPUS'],
alpha=0.5, color='pink', label='Bar Graph')

# Add labels and title
ax1.set_xlabel('Year- Quarter 1')
ax1.set_ylabel('price_index', color='blue')
ax2.set_ylabel('MSPUS', color='pink')
plt.title('Year VS price_index VS MSPUS (For Demand factors)')

# Add legends
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

# Show the plot
plt.show()
```



- MSPUS i.e Median Sale Price are highly realted with price index as we can see taht in Q1 all the MSP are higher so is Price_index
- Now since we analyzed the data , we initially found some skewness in our columnsbefore proceedding further I need to make sure there is no o less skewness in our columns

Using Log Transform to fix skewness¶

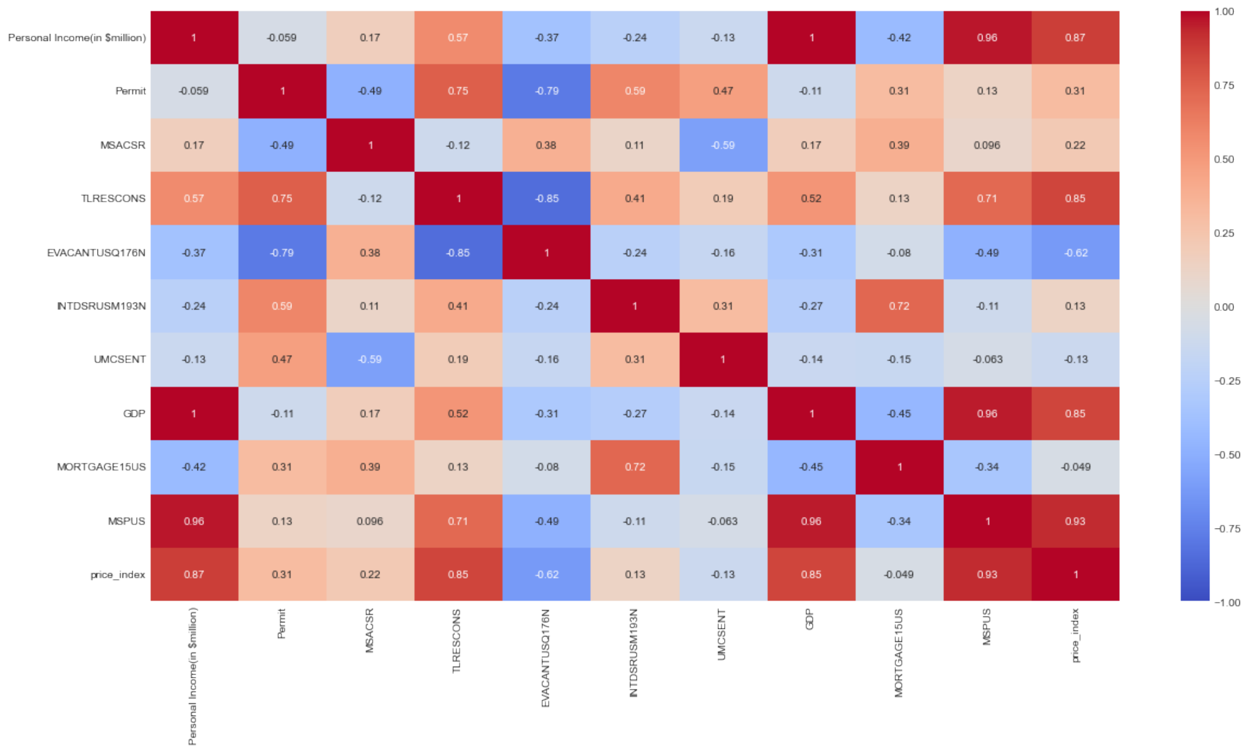
```
for col in numerical_cols:
    if df.skew().loc[col]>0.55:
        df[col]=np.log1p(df[col])

df[numerical_cols].skew()
```

```
Personal Income(in $million)    0.377310
Permit                          0.070731
MSACSR                          0.420000
TLRESCONS                      -0.120525
EVACANTUSQ176N                 -0.246561
INTDSRUSM193N                   0.557704
UMCSENT                        -0.397624
GDP                             0.215310
MORTGAGE15US                    0.269950
MSPUS                           0.574825
price_index                     0.899402
dtype: float64
```

- Here we have handled the skewness in continous data.now all the continous colums are in accetepable range of skewness(+/-.5)

Correlation using a Heatmap

```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.show()
```



Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together (MSPUS,Personal Income,TLRESCONS,GDP,Permit,MSACSR). Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down (INTDSRUSM193N,MORTGAGE15US,UMCSENT,EVACANTUSQ176N).

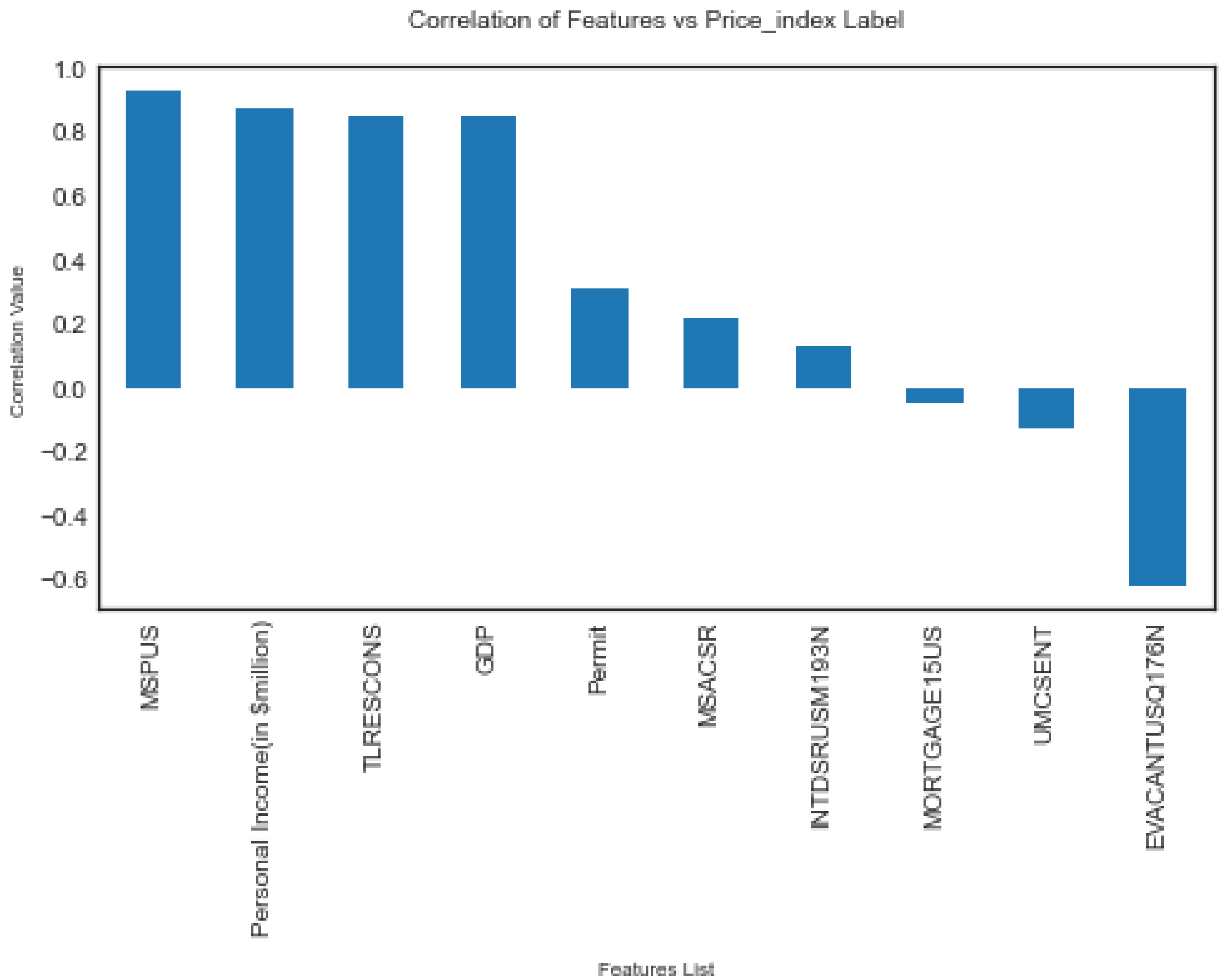
In the above heatmap we can see the correlation details plus we can determine that there is some multi colinearity issue between our columns like

- MSPUS & Personal Income(0.96)
- GDP & MSPUS (0.96) and others as well

but we have very less columns in our dataset so its not wise to drop these columns as of now.if there is no good Acuuray achieved definietly will do something

Correlation Bar Plot comparing features with our label

```
plt.style.use('seaborn-white')

df_corr = df.corr()
plt.figure(figsize=(8,4))
df_corr['price_index'].sort_values(ascending=False).drop('price_index'
).plot.bar()
plt.title("Correlation of Features vs Price_index Label\n",
fontsize=10)
plt.xlabel("\nFeatures List", fontsize=8)
plt.ylabel("Correlation Value", fontsize=8)
plt.show()
```



-Here Bar plot is giving us a clearer picture on positive and negative correlation columns we have generated this bar plot and we see that more than half the feature columns are positively correlated with our target label while all the remaining features are negatively correlated with our label column.

- postively realted columns with label column are - **MSPUS,Personal Income,TLRESCONS,GDP,Permit,MSACSR**
- Negatively related columns with Label column are - **INTDSRUSM193N,MORTGAGE15US,UMCSENT,EVACANTUSQ176N**

Splitting the dataset into 2 variables namely 'x' and 'y' for feature and label¶

```
x=df.drop('price_index',axis=1)
y=df['price_index']
y

y
```

```
0     4.870003
1     4.888513
2     4.912753
3     4.940461
4     4.971888
        ...
78    5.712307
79    5.701062
80    5.699556
81    5.715241
82    5.736677
Name: price_index, Length: 83, dtype: float64
```

```
years = df['year']

x=x.drop(['year'], axis=1)
```

Scaling the features

```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

X_scaled=scaler.fit_transform(x)
```

For Best Random State¶

```
lr=LinearRegression()

for i in range(1,1000):

x_train,x_test,y_train,y_test=train_test_split(X_scaled,y,test_size=.2
0)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)

if(round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test,pr
ed_test)*100,1)):
        print("At random_state",i,"the model performed very well")
        print("At random_state",i)
        print("Trainning Score",r2_score(y_train,pred_train)*100)
        print("Testing Score",r2_score(y_test,pred_test)*100,'\n\n')


At random_state 7 the model performed very well
At random_state 7
Trainning Score 98.50781148145131
Testing Score 98.49777954341579
```

```
At random_state 34 the model performed very well
At random_state 34
Trainning Score 99.03487768903587
Testing Score 99.01310501069105


At random_state 38 the model performed very well
At random_state 38
Trainning Score 98.98281071763574
Testing Score 99.01677128927346


At random_state 85 the model performed very well
At random_state 85
Trainning Score 98.9015811374059
Testing Score 98.86171545208583


At random_state 109 the model performed very well
At random_state 109
Trainning Score 99.0324695847905
Testing Score 99.03667896593356


At random_state 116 the model performed very well
At random_state 116
Trainning Score 98.99527205348755
Testing Score 98.99236817523328


At random_state 118 the model performed very well
At random_state 118
Trainning Score 98.98819664138408
Testing Score 98.98045699151113


At random_state 121 the model performed very well
At random_state 121
Trainning Score 98.981087888122
Testing Score 99.02670465633132


At random_state 159 the model performed very well
At random_state 159
Trainning Score 99.0256030420941
Testing Score 99.02637691409129


At random_state 175 the model performed very well
At random_state 175
Trainning Score 99.03650142192514
```

```
Testing Score 98.99036513719702


At random_state 196 the model performed very well
At random_state 196
Trainning Score 99.02054340460899
Testing Score 98.99265753628632


At random_state 224 the model performed very well
At random_state 224
Trainning Score 99.02489368950854
Testing Score 98.98614848936533


At random_state 250 the model performed very well
At random_state 250
Trainning Score 98.9682206552996
Testing Score 99.03436520939341


At random_state 269 the model performed very well
At random_state 269
Trainning Score 98.96772611434383
Testing Score 98.99448706204474


At random_state 271 the model performed very well
At random_state 271
Trainning Score 99.0278066529386
Testing Score 98.98754323736281


At random_state 272 the model performed very well
At random_state 272
Trainning Score 99.00711371677805
Testing Score 99.03717820633146


At random_state 283 the model performed very well
At random_state 283
Trainning Score 99.03411918477352
Testing Score 98.97321532847442


At random_state 304 the model performed very well
At random_state 304
Trainning Score 98.99620869281122
Testing Score 98.99439852329695
```

```
At random_state 336 the model performed very well
At random_state 336
Trainning Score 99.03300408710376
Testing Score 98.96494832947744


At random_state 341 the model performed very well
At random_state 341
Trainning Score 99.0118618484819
Testing Score 99.02976351970227


At random_state 358 the model performed very well
At random_state 358
Trainning Score 98.9935412065946
Testing Score 98.9842949314869


At random_state 363 the model performed very well
At random_state 363
Trainning Score 98.97500456377138
Testing Score 98.98183714520901


At random_state 371 the model performed very well
At random_state 371
Trainning Score 98.97470370777819
Testing Score 99.01018684831435


At random_state 376 the model performed very well
At random_state 376
Trainning Score 99.02283258399048
Testing Score 98.98385568526027


At random_state 383 the model performed very well
At random_state 383
Trainning Score 99.03715761159225
Testing Score 99.00303273019156


At random_state 403 the model performed very well
At random_state 403
Trainning Score 99.03857636926865
Testing Score 98.96311421452579


At random_state 406 the model performed very well
At random_state 406
Trainning Score 98.99343590795895
```

```
Testing Score 99.04195095728916


At random_state 439 the model performed very well
At random_state 439
Trainning Score 98.93494063925544
Testing Score 98.90274924148123


At random_state 480 the model performed very well
At random_state 480
Trainning Score 98.98654183370671
Testing Score 99.04868626168465


At random_state 481 the model performed very well
At random_state 481
Trainning Score 99.01096999136072
Testing Score 98.99815487066344


At random_state 482 the model performed very well
At random_state 482
Trainning Score 99.01217039742752
Testing Score 99.01640401821071


At random_state 487 the model performed very well
At random_state 487
Trainning Score 99.04238262739482
Testing Score 99.01065337508884


At random_state 499 the model performed very well
At random_state 499
Trainning Score 98.97044052739824
Testing Score 99.02885885130974


At random_state 502 the model performed very well
At random_state 502
Trainning Score 99.00560255720603
Testing Score 98.98707359581866


At random_state 504 the model performed very well
At random_state 504
Trainning Score 98.96001502302296
Testing Score 98.9844944927762
```

```
At random_state 561 the model performed very well
At random_state 561
Trainning Score 98.79398041586495
Testing Score 98.79896933829832


At random_state 562 the model performed very well
At random_state 562
Trainning Score 99.03161578127195
Testing Score 98.97183222741374


At random_state 614 the model performed very well
At random_state 614
Trainning Score 98.97606241878036
Testing Score 99.0393357081837


At random_state 658 the model performed very well
At random_state 658
Trainning Score 99.00404772375029
Testing Score 99.01811886913974


At random_state 733 the model performed very well
At random_state 733
Trainning Score 98.9769199350939
Testing Score 99.00851286256523


At random_state 779 the model performed very well
At random_state 779
Trainning Score 98.99156317489279
Testing Score 99.03562359926015


At random_state 781 the model performed very well
At random_state 781
Trainning Score 98.98034600633866
Testing Score 99.01734651951239


At random_state 792 the model performed very well
At random_state 792
Trainning Score 99.01464540275072
Testing Score 98.95900811598513


At random_state 794 the model performed very well
At random_state 794
Trainning Score 98.9851723953145
```

```
Testing Score 98.9789076797521


At random_state 798 the model performed very well
At random_state 798
Trainning Score 98.9421588409492
Testing Score 98.94152089957782


At random_state 816 the model performed very well
At random_state 816
Trainning Score 99.0119174449676
Testing Score 98.98085421022365


At random_state 848 the model performed very well
At random_state 848
Trainning Score 99.01293572762717
Testing Score 98.95689162056807


At random_state 875 the model performed very well
At random_state 875
Trainning Score 99.01837016038424
Testing Score 98.99384781534503


At random_state 894 the model performed very well
At random_state 894
Trainning Score 99.04351889784392
Testing Score 98.95990922576675


At random_state 920 the model performed very well
At random_state 920
Trainning Score 99.00219117964298
Testing Score 98.99536893892676


At random_state 922 the model performed very well
At random_state 922
Trainning Score 99.00008984194454
Testing Score 99.03153837934238


At random_state 952 the model performed very well
At random_state 952
Trainning Score 98.975243648335
Testing Score 99.04531471886298
```

```
At random_state 960 the model performed very well
At random_state 960
Trainning Score 99.02940591491321
Testing Score 98.97784531911384


At random_state 967 the model performed very well
At random_state 967
Trainning Score 99.01742580137869
Testing Score 98.99650931094274


At random_state 973 the model performed very well
At random_state 973
Trainning Score 99.0182828480453
Testing Score 98.97724554802859


At random_state 991 the model performed very well
At random_state 991
Trainning Score 99.02902522903074
Testing Score 98.97949948065275
```

```
# Selecting rando state 85
x_train,x_test,y_train,y_test=train_test_split(X_scaled,y,test_size=.2
0,random_state=85)

# Divided the dataset in 80:20 ratio that means 80% of the data is for
training and rest 20% data is for testing

print('Size of x_train : ', x_train.shape)
print('Size of y_train : ', y_train.shape)
print('Size of x_test : ', x_test.shape)
print('Size of Y_test : ', y_test.shape)
```

```
Size of x_train :  (66, 10)
Size of y_train :  (66,)
Size of x_test :  (17, 10)
Size of Y_test :  (17,)
```

Single functin for different Regression Model

```
def reg(model,X_scaled,y):
    X_train, X_test, Y_train,
Y_test=train_test_split(X_scaled,y,test_size=.30)


    # Training the model
    model.fit(X_train, Y_train)
```

```python
    # Predicting Y_test
    pred = model.predict(X_test)

    # MSE - a lower RMSE score is better than a higher one
    mse = mean_squared_error(Y_test, pred)
    print("MSE Score is:", mse)

    #RMSE
    print("RMSE Score is:", np.sqrt(mse))

    # R2 score
    r2 = r2_score(Y_test, pred)*100
    print("R2 Score is:", r2)

    # Cross Validation Score
    cv_score = (cross_val_score(model, X_scaled,y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of r2 score minus cv score
    result = r2 - cv_score
    print("R2 Score - Cross Validation Score is", result)
```

Linear Regression Model

```python
model=LinearRegression()
reg(model, X_scaled,y)
```

```
MSE Score is: 0.0009116650529550087
RMSE Score is: 0.03019379162932355
R2 Score is: 97.80619603404472
Cross Validation Score: 58.22251009238247
R2 Score - Cross Validation Score is 39.583685941662246
```

Ridge Regression

```python
model=Ridge(alpha=0.05)
reg(model, X_scaled,y)
```

```
MSE Score is: 0.0008020467935716945
RMSE Score is: 0.028320430674191637
R2 Score is: 98.6339050623801
Cross Validation Score: 59.315020463097724
R2 Score - Cross Validation Score is 39.31888459928237
```

Lasso Regression

```python
model=Lasso(0.01)
reg(model, X_scaled,y)
```

```
MSE Score is: 0.0010635420176043648
RMSE Score is: 0.03261199192941708
R2 Score is: 97.45443920348393
Cross Validation Score: 24.055181142224725
R2 Score - Cross Validation Score is 73.3992580612592
```

Support Vector Regression

```
model=SVR()
reg(model, X_scaled,y)
```

```
MSE Score is: 0.004048234182353822
RMSE Score is: 0.06362573522053652
R2 Score is: 89.11766152077358
Cross Validation Score: -281.88186617728
R2 Score - Cross Validation Score is 370.9995276980536
```

AdaBoostRegressor

```
model=AdaBoostRegressor()
reg(model, X_scaled,y)
```

```
MSE Score is: 0.0016519582743053506
RMSE Score is: 0.04064428956576004
R2 Score is: 97.71974807386923
Cross Validation Score: -162.5862199103026
R2 Score - Cross Validation Score is 260.30596798417184
```

Random Forest Regressor

```
rfc=RandomForestRegressor()
reg(model, X_scaled,y)
```

```
MSE Score is: 0.0007349304602298569
RMSE Score is: 0.027109600886583648
R2 Score is: 97.98272134420404
Cross Validation Score: 24.055181142224725
R2 Score - Cross Validation Score is 73.92754020197931
```

To select the best performing model, we used cross-validation with five folds. This technique helps assess the models' performance on different subsets of the training data. We used the mean squared error (MSE) as the evaluation metric, where lower values indicate better performance.

Definietly Ridge is performin better than all models

```
ridge = Ridge(alpha=1.0)

ridge.fit(x_train, y_train)
```

```
ridge_mse = mean_squared_error(y_test, ridge.predict(x_test))
ridge_rmse = np.sqrt(ridge_mse)
ridge_r2 = ridge.score(x_test, y_test)

# Perform cross-validation
ridge_cv_score = cross_val_score(ridge, X_scaled, y, cv=5,
scoring='r2').mean()

print("Ridge Regression Results:")
print(f"MSE Score: {ridge_mse}")
print(f"RMSE Score: {ridge_rmse}")
print(f"R2 Score: {ridge_r2}")
print(f"Cross Validation Score: {ridge_cv_score}")
```

```
Ridge Regression Results:
MSE Score: 0.0010558438249160414
RMSE Score: 0.032493750551699035
R2 Score: 0.977612982440919
Cross Validation Score: 0.5567246235350294
```

RidgeCV is a convenient way to apply Ridge Regression with cross-validated hyperparameter tuning. It combines the Ridge Regression model with the process of selecting the best alpha through cross-validationoverfirring issue so lets use Ridgecv model to reduce any overfitting of the model

```
from sklearn.linear_model import RidgeCV

alphas = [0.1, 1.0, 10.0]  # Specify the alpha values to consider
ridgecv_model = RidgeCV(alphas=alphas, store_cv_values=True)  # Set
store_cv_values=True to access cross-validation results

# Fit RidgeCV model
ridgecv_model.fit(x_train, y_train)

# Get the optimal alpha
best_alpha = ridgecv_model.alpha_

# Evaluate RidgeCV model
ridgecv_mse = mean_squared_error(y_test,
ridgecv_model.predict(x_test))
ridgecv_r2=r2_score(y_test, ridgecv_model.predict(x_test))
print(f"RidgeCV Regression MSE: {ridgecv_mse}")
print(f"RidgeCV r2 score: {ridgecv_r2}")
```

```
RidgeCV Regression MSE: 0.0012437523427187474
RidgeCV r2 score: 0.9736287650895653
```

RidgeCV Regression MSE: 0.0012437523427187474: This is the mean squared error, a measure of the average squared difference between the predicted and actual values. A lower MSE indicates better model performance.

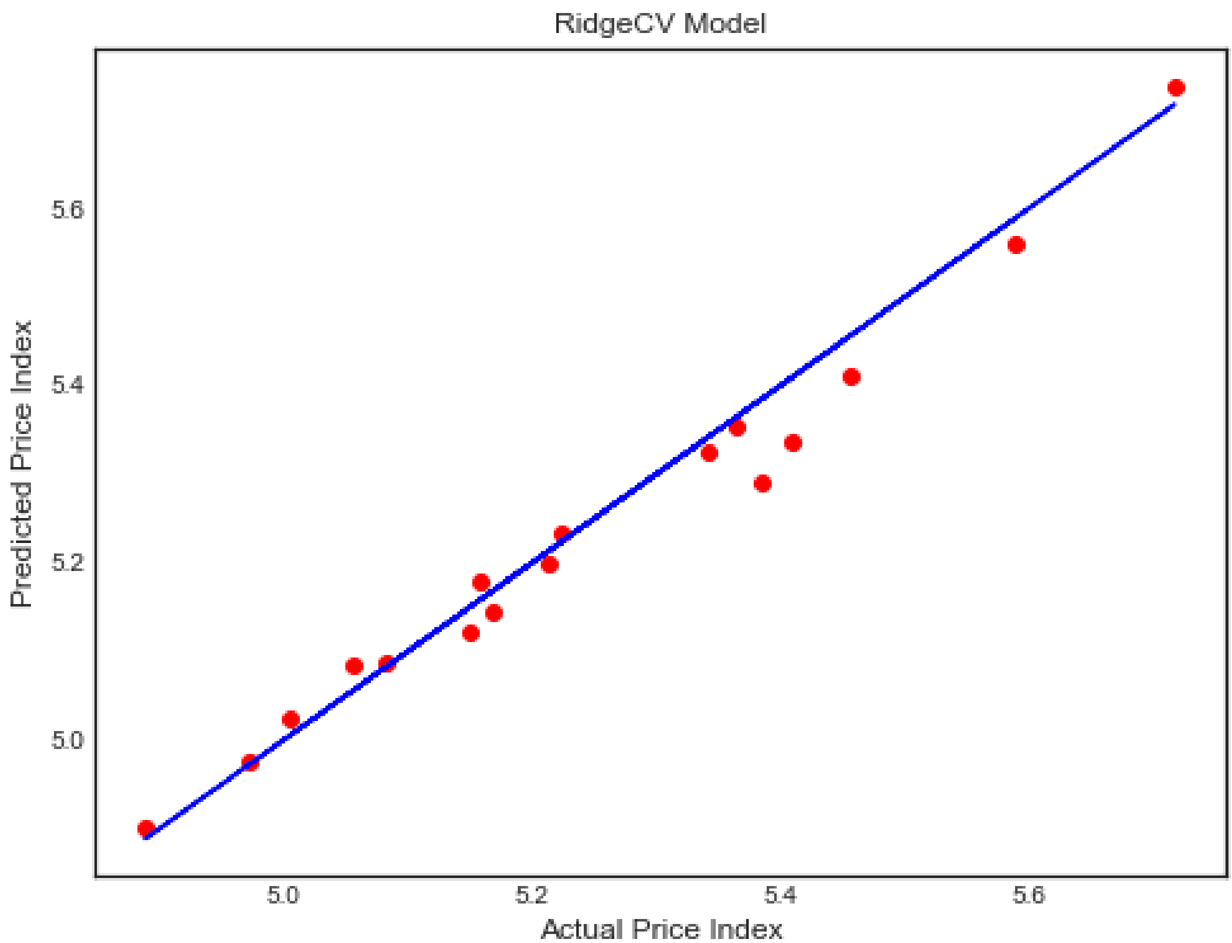
RidgeCV R2 Score: 0.9736287650895653: The R2 score, also known as the coefficient of determination, represents the proportion of the variance in the dependent variable that is predictable from the independent variables. An R2 score of 0.97 is excellent, indicating that your RidgeCV model captures a large portion of the variance in the target variable.

RidgeCV is perfroming better than Ridge model

Our final Model is RidgeCV

Visualising the final model

```
ridgecv_model.fit(x_train, y_train)
ridgecv_train=ridgecv_model.predict(x_train)
ridgecv_test=ridgecv_model.predict(x_test)

plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=ridgecv_test,color='r')
plt.plot(y_test,y_test,color='b')
plt.xlabel("Actual Price Index",fontsize=12)
plt.ylabel("Predicted Price Index",fontsize=12)
plt.title("RidgeCV Model")
plt.show()
```



Saving the final model

```
import pickle

filename=('homeprice_index.pkl')

pickle.dump(ridgecv_model,open(filename,'wb'))

#conclusion
loaded_model=pickle.load(open('homeprice_index.pkl','rb'))
result=loaded_model.score(x_test,y_test)
print(result*100)

97.36287650895653
```

our final model is giving me 97% approx accuracy

```
conclusion=pd.DataFrame([loaded_model.predict(x_test)
[:],ridgecv_test[:]],index=['Predcited','Orignal'])
conclusion
```

```
                 0          1          2          3          4          5  \
Predcited  5.337091  5.178016  5.291651  5.737626  5.233423  5.121596

Orignal    5.337091  5.178016  5.291651  5.737626  5.233423  5.121596


                 6          7          8          9         10         11  \
Predcited  4.901116  5.324261  5.145598  5.024913  5.411445  5.197731

Orignal    4.901116  5.324261  5.145598  5.024913  5.411445  5.197731


                12         13         14         15         16
Predcited  5.083528  5.086693  5.559857  5.35459  4.97714
Orignal    5.083528  5.086693  5.559857  5.35459  4.97714
```

Prediction For 2023-Q4 to 2025-Q4

on the basis of collected data , made the assumption for this new csv file which contains only input not the output and then accessed.

```
# Accessing csv file
data=pd.read_csv('prediction.csv')
data

      Year  Personal Income(in $million)   Permit  MSACSR    TLRESCONS  \
0  2023 Q4                         13.50  1758.33     5.2  521328.6667

1  2024-Q1                         14.50  1145.25     3.6  579308.6667

2  2024-Q2                         11.05  1123.74     8.5  459890.0000

3  2024-Q3                         15.50  1452.60     4.9  691437.3333

4  2024-Q4                         16.50  1784.25     7.8  516856.3333

5  2025-Q1                         18.90  1551.32     9.0  956483.3333

6  2025-Q2                         19.20  1785.32     8.0  752939.0000

7  2025-Q3                         11.50  1655.20     8.9  801413.3333

8  2025-Q4                         15.20  1922.30     9.8  902790.3333


   EVACANTUSQ176N  INTDSRUSM193N    UMCSENT        GDP  MORTGAGE15US
```

```
MSPUS
0           18208           0.75  84.766667  29828.973      6.171429
441200
1           19009           0.85  79.900000  28654.603      6.338462
443600
2           19184           2.50  33.133333  28029.116      7.044615
463100
3           17312           2.75  77.866667  30044.273      4.450769
479300
4           19340           2.85  55.125110  30259.639      6.880769
478000
5           18593           0.75  57.800000  29408.405      7.159231
489500
6           16102           1.93  69.600000  28813.601      6.815385
489000
7           14049           2.00  68.300000  29063.012      7.070769
499500
8           14172           2.32  71.600000  30644.463      7.396154
501000
```

```
data.shape
```

```
(9, 11)
```

it has 9 rows and 11 columns

Applying Preprocessing steps on this data

```
data.isnull().sum()
```

```
Year                            0
Personal Income(in $million)    0
Permit                          0
MSACSR                          0
TLRESCONS                       0
EVACANTUSQ176N                  0
INTDSRUSM193N                   0
UMCSENT                         0
GDP                             0
MORTGAGE15US                    0
MSPUS                           0
dtype: int64
```

```
X=data.drop('Year',axis=1)
```

```
scaler=StandardScaler()
X=scaler.fit_transform(X)
X
```

```
array([[-0.5903001 ,  0.67948317, -1.0255093 , -0.98152996,
0.44586525,
```

```
        -1.36521611,  1.22850433,  0.51517428, -0.50017577, -
1.65786786],
       [-0.22007704, -1.59736687, -1.80684971, -0.63789645,
0.85257695,
        -1.24172923,  0.90201279, -0.95064746, -0.30150547, -
1.54360511],
       [-1.4973466 , -1.67725048,  0.58600531, -1.34566215,
0.94143406,
         0.79580437, -2.23543666, -1.73136601,  0.53839936, -
0.61522027],
       [ 0.15014602, -0.45593368, -1.17201062,  0.02666317, -
0.00908317,
         1.10452158,  0.76560195,  0.78390681, -2.54674125,
0.15605329],
       [ 0.52036908,  0.7757446 ,  0.24416888, -1.0080364 ,
1.02064383,
         1.22800846, -0.76006772,  1.05272172,  0.34351949,
0.09416097],
       [ 1.40890442, -0.08930838,  0.83017419,  1.59752704,  0.6413509
,
        -1.36521611, -0.58061657, -0.00976907,  0.67472379,
0.64166998],
       [ 1.51997134,  0.77971835,  0.34183643,  0.39116875, -
0.62346664,
         0.09192913,  0.21101359, -0.7521898 ,  0.26575098,
0.61786524],
       [-1.33074622,  0.29648007,  0.78134042,  0.67846446, -
1.66588751,
         0.17836994,  0.1238001 , -0.4408807 ,  0.56950695,
1.11776477],
       [ 0.0390791 ,  1.28843323,  1.2208444 ,  1.27930155, -
1.60343366,
         0.57352798,  0.34518819,  1.53305022,  0.95652192,
1.18917899]])

#prediction using loaded_model
predictions = loaded_model.predict(X)

#prediction
predictions

array([5.08137823, 4.79185222, 4.85285986, 5.14141754, 5.34865057,
       5.4216637 , 5.27667721, 5.24876754, 5.60488918])

data['Predicted price Index']=predictions

data

      Year  Personal Income(in $million)   Permit  MSACSR    TLRESCONS
\
0  2023 Q4                          13.50  1758.33     5.2  521328.6667
```

```
1  2024-Q1                                  14.50  1145.25      3.6  579308.6667
2  2024-Q2                                  11.05  1123.74      8.5  459890.0000
3  2024-Q3                                  15.50  1452.60      4.9  691437.3333
4  2024-Q4                                  16.50  1784.25      7.8  516856.3333
5  2025-Q1                                  18.90  1551.32      9.0  956483.3333
6  2025-Q2                                  19.20  1785.32      8.0  752939.0000
7  2025-Q3                                  11.50  1655.20      8.9  801413.3333
8  2025-Q4                                  15.20  1922.30      9.8  902790.3333

   EVACANTUSQ176N  INTDSRUSM193N    UMCSENT        GDP  MORTGAGE15US
MSPUS  \
0           18208           0.75  84.766667  29828.973      6.171429
441200
1           19009           0.85  79.900000  28654.603      6.338462
443600
2           19184           2.50  33.133333  28029.116      7.044615
463100
3           17312           2.75  77.866667  30044.273      4.450769
479300
4           19340           2.85  55.125110  30259.639      6.880769
478000
5           18593           0.75  57.800000  29408.405      7.159231
489500
6           16102           1.93  69.600000  28813.601      6.815385
489000
7           14049           2.00  68.300000  29063.012      7.070769
499500
8           14172           2.32  71.600000  30644.463      7.396154
501000

   Predicted price Index
0               5.081378
1               4.791852
2               4.852860
3               5.141418
4               5.348651
5               5.421664
6               5.276677
7               5.248768
8               5.604889
```

Conclusion

- Supply factors, such as house inventory and the number of authorized housing units, have a positive influence on home prices. Higher construction spending on residential projects also contributes significantly to higher home prices.
- Demand factor, such as mortgage interest rates, have a negative impact on home prices. Higher mortgage rates and lower consumer sentiment are associated with slightly lower home prices.
- Economic factors, including GDP and interest rates, play a crucial role in determining home prices. A strong economy with higher GDP and slightly lower interest rates tends to support higher home prices.
- The median sales price of houses sold is strongly correlated with home prices, reflecting the importance of market dynamics and buyer behaviour in determining home price movements.
- These insights can be valuable for various stakeholders in the real estate market, including home buyers, sellers, developers, and policymakers. Understanding the factors that influence home prices can help make informed decisions related to investments, financing, and economic policies.