1. Data Collection

The information gathered for this task has been separated into many files. These dataset include annual data on the key factor influencing US home prices across the country. Here is a link for the data.

https://www.macrotrends.net/countries/USA/united-states/gdp-growth-rate (https://www.macrotrends.net/countries/USA/united-states/gdp-growth-rate)

https://www.macrotrends.net/countries/USA/united-states/inflation-rate-cpi (https://www.macrotrends.net/countries/USA/united-states/inflation-rate-cpi)

https://www.macrotrends.net/countries/USA/united-states/unemployment-rate (https://www.macrotrends.net/countries/USA/united-states/unemployment-rate)

https://themortgagereports.com/61853/30-year-mortgage-rates-chart#current (https://themortgagereports.com/61853/30-year-mortgage-rates-chart#current)

https://fred.stlouisfed.org/series/MSACSR (https://fred.stlouisfed.org/series/MSACSR)

https://fred.stlouisfed.org/series/TLRESCONS (https://fred.stlouisfed.org/series/TLRESCONS)

2. Data Preparation and Preprocessing

In [1]:
```
#importing libary

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

import warnings
warnings.filterwarnings('ignore')
```

Importing different key factors csv file and clean the data for analysis

In [2]:
```
data_mortgage = pd.read_csv("D:/Sumit/data_files/AVG_MORTGAGE_20US.csv")
data_mortgage
```

Out[2]:

	Year	Average_Mortgage_Rate(%)
0	2003	5.83%
1	2004	5.84%
2	2005	5.87%
3	2006	6.41%
4	2007	6.34%
5	2008	6.03%
6	2009	5.04%
7	2010	4.69%
8	2011	4.45%
9	2012	3.66%
10	2013	3.98%
11	2014	4.17%
12	2015	3.85%
13	2016	3.65%
14	2017	3.99%
15	2018	4.54%
16	2019	3.94%
17	2020	3.10%
18	2021	2.96%
19	2022	5.34%

In [3]:
```
# Remove '%' symbol from 'Average_Mortgage_Rate'
data_mortgage['Average_Mortgage_Rate(%)'] = data_mortgage['Average_Mortgage_Rate(%)'].str.replace(
data_mortgage
```

Out[3]:

	Year	Average_Mortgage_Rate(%)
0	2003	5.83
1	2004	5.84
2	2005	5.87
3	2006	6.41
4	2007	6.34
5	2008	6.03
6	2009	5.04
7	2010	4.69
8	2011	4.45
9	2012	3.66
10	2013	3.98
11	2014	4.17
12	2015	3.85
13	2016	3.65
14	2017	3.99
15	2018	4.54
16	2019	3.94
17	2020	3.10
18	2021	2.96
19	2022	5.34

In [4]:
```
data_gdp = pd.read_csv("D:/Sumit/data_files/GDP_Growth.csv")
data_gdp
```

Out[4]:

	Year	GDP_Growth(%)
0	2003	2.80%
1	2004	3.85%
2	2005	3.48%
3	2006	2.78%
4	2007	2.01%
5	2008	0.12%
6	2009	-2.60%
7	2010	2.71%
8	2011	1.55%
9	2012	2.28%
10	2013	1.84%
11	2014	2.29%
12	2015	2.71%
13	2016	1.67%
14	2017	2.24%
15	2018	2.95%
16	2019	2.29%
17	2020	-2.77%
18	2021	5.95%
19	2022	2.06%

In [5]:
```
# Remove '%' symbol from 'GDP_Growth'
data_gdp['GDP_Growth(%)'] = data_gdp['GDP_Growth(%)'].str.replace('%', '').astype(float)
```

In [6]:
```
data_gdp.head()
```

Out[6]:

	Year	GDP_Growth(%)
0	2003	2.80
1	2004	3.85
2	2005	3.48
3	2006	2.78
4	2007	2.01

In [7]:
```
data_unemp = pd.read_csv("D:/Sumit/data_files/Unemployment_rate.csv")
data_unemp
```

Out[7]:

	Year	Unemployment_Rate(%)
0	2003	5.99%
1	2004	5.53%
2	2005	5.08%
3	2006	4.62%
4	2007	4.62%
5	2008	5.78%
6	2009	9.25%
7	2010	9.63%
8	2011	8.95%
9	2012	8.07%
10	2013	7.37%
11	2014	6.17%
12	2015	5.28%
13	2016	4.87%
14	2017	4.36%
15	2018	3.90%
16	2019	3.67%
17	2020	8.05%
18	2021	5.35%
19	2022	3.61%

In [8]:
```
# Remove '%' symbol
data_unemp['Unemployment_Rate(%)'] = data_unemp['Unemployment_Rate(%)'].str.replace('%', '').astyp
```

In [9]:
```
data_unemp.head()
```

Out[9]:

	Year	Unemployment_Rate(%)
0	2003	5.99
1	2004	5.53
2	2005	5.08
3	2006	4.62
4	2007	4.62

In [10]:
```
data_in = pd.read_csv("D:/Sumit/data_files/US_Inflation_rate.csv")
data_in
```

Out[10]:

	Year	Inflation_Rate(%)
0	2003	2.27%
1	2004	2.68%
2	2005	3.39%
3	2006	3.23%
4	2007	2.85%
5	2008	3.84%
6	2009	-0.36%
7	2010	1.64%
8	2011	3.16%
9	2012	2.07%
10	2013	1.46%
11	2014	1.62%
12	2015	0.12%
13	2016	1.26%
14	2017	2.13%
15	2018	2.44%
16	2019	1.81%
17	2020	1.23%
18	2021	4.70%
19	2022	8.00%

In [11]:
```
# Remove '%' symbol
data_in['Inflation_Rate(%)'] = data_in['Inflation_Rate(%)'].str.replace('%', '').astype(float)
```

In [12]:
```
data_in.head()
```

Out[12]:

	Year	Inflation_Rate(%)
0	2003	2.27
1	2004	2.68
2	2005	3.39
3	2006	3.23
4	2007	2.85

In [13]:
```
data_msa = pd.read_csv("D:/Sumit/data_files/MSACSR.csv") # Monthly Supply of New Houses in the U.S
data_msa
```

Out[13]:

	DATE	MSACSR
0	01-01-2003	4.0
1	01-02-2003	4.5
2	01-03-2003	4.1
3	01-04-2003	4.1
4	01-05-2003	3.9
...	...	...
235	01-08-2022	8.7
236	01-09-2022	9.7
237	01-10-2022	9.7
238	01-11-2022	9.4
239	01-12-2022	8.5

240 rows × 2 columns

The data records in MSACSR are organized on a monthly basis, and for a yearly analysis, it is necessary to group the records by year. In this process, the mean of each month within a specific year will be calculated, providing a consolidated yearly overview. Same process take place for TLRESCONS (Total Construction Spending in Residential) dataset and CSUSHPISA dataset.

In [14]:
```
# Convert string to datetime object
data_msa['DATE'] = pd.to_datetime(data_msa['DATE'], format='%d-%m-%Y')
```

In [15]:
```
data_msa['Year'] = data_msa['DATE'].dt.year
```

In [16]:
```
data_msa = data_msa.drop(columns=['DATE'])
```

In [17]:
```
data_msa.head()
```

Out[17]:

	MSACSR	Year
0	4.0	2003
1	4.5	2003
2	4.1	2003
3	4.1	2003
4	3.9	2003

In [18]:
```
# Group by year and calculate the mean of 'MSACSR'
data_msa = data_msa.groupby(data_msa['Year'])['MSACSR'].mean().round(2).reset_index()
```

In [19]:
```
data_msa
```

Out[19]:

	Year	MSACSR
0	2003	3.91
1	2004	4.00
2	2005	4.45
3	2006	6.43
4	2007	8.38
5	2008	10.68
6	2009	9.03
7	2010	8.00
8	2011	6.58
9	2012	4.76
10	2013	4.74
11	2014	5.48
12	2015	5.17
13	2016	5.21
14	2017	5.38
15	2018	6.18
16	2019	5.82
17	2020	4.62
18	2021	5.50
19	2022	8.45

In [20]:
```
data_tlcons = pd.read_csv("D:/Sumit/data_files/TLRESCONS.csv")
data_tlcons
```

Out[20]:

	DATE	TLRESCONS
0	2003-01-01	423049.0
1	2003-02-01	422705.0
2	2003-03-01	418232.0
3	2003-04-01	425493.0
4	2003-05-01	426270.0
...	...	...
245	2023-06-01	870655.0
246	2023-07-01	865747.0
247	2023-08-01	885776.0
248	2023-09-01	884184.0
249	2023-10-01	895130.0

250 rows × 2 columns

In [21]:
```
# Convert string to datetime object
data_tlcons['DATE'] = pd.to_datetime(data_tlcons['DATE'], format='%Y-%m-%d')
```

In [22]:
```
data_tlcons['Year'] = data_tlcons['DATE'].dt.year
data_tlcons = data_tlcons.drop(columns=['DATE'])
```

In [23]:
```
data_tlcons.head()
```

Out[23]:

	TLRESCONS	Year
0	423049.0	2003
1	422705.0	2003
2	418232.0	2003
3	425493.0	2003
4	426270.0	2003

In [24]:
```
# Group by year and calculate the mean of 'TLRESCONS'
data_tlcons = data_tlcons.groupby(data_tlcons['Year'])['TLRESCONS'].mean().round(2).reset_index()
```

In [25]:
```
data_tlcons = data_tlcons.loc[data_tlcons['Year'] != 2023].reset_index(drop=True)
```

In [26]:
```
data_tlcons
```

Out[26]:

	Year	TLRESCONS
0	2003	450241.17
1	2004	536923.00
2	2005	628863.50
3	2006	617260.17
4	2007	497164.00
5	2008	367000.08
6	2009	261395.50
7	2010	256535.67
8	2011	255208.58
9	2012	278995.58
10	2013	335207.33
11	2014	382812.25
12	2015	437998.08
13	2016	486102.25
14	2017	545873.58
15	2018	564342.50
16	2019	552999.25
17	2020	644425.00
18	2021	808891.17
19	2022	927137.58

In [27]:
```
data_cus = pd.read_csv("D:/Sumit/data_files/CSUSHPISA.csv")
data_cus
```

Out[27]:

	DATE	CSUSHPISA
0	2003-01-01	128.461
1	2003-02-01	129.355
2	2003-03-01	130.148
3	2003-04-01	130.884
4	2003-05-01	131.735
...	...	...
235	2022-08-01	301.473
236	2022-09-01	299.353
237	2022-10-01	298.873
238	2022-11-01	298.269
239	2022-12-01	297.413

240 rows × 2 columns

In [28]:
```
# Convert string to datetime object
data_cus['DATE'] = pd.to_datetime(data_cus['DATE'], format='%Y-%m-%d')
```

In [29]:
```
data_cus['Year'] = data_cus['DATE'].dt.year
data_cus = data_cus.drop(columns=['DATE'])
```

In [30]:
```
data_cus.head()
```

Out[30]:

	CSUSHPISA	Year
0	128.461	2003
1	129.355	2003
2	130.148	2003
3	130.884	2003
4	131.735	2003

In [31]:
```
# Group by year and calculate the mean of 'CSUSHPISA'
data_cus = data_cus.groupby(data_cus['Year'])['CSUSHPISA'].mean().round(2).reset_index()
```

In [32]:
```
data_cus
```

Out[32]:

	Year	CSUSHPISA
0	2003	133.73
1	2004	150.44
2	2005	171.74
3	2006	183.45
4	2007	179.92
5	2008	164.06
6	2009	148.55
7	2010	144.67
8	2011	139.26
9	2012	140.99
10	2013	154.52
11	2014	164.70
12	2015	172.18
13	2016	180.93
14	2017	191.40
15	2018	202.48
16	2019	209.46
17	2020	222.14
18	2021	260.05
19	2022	298.49

Here next step is to merge all clean dataset into a final dataset which we can use for analysis and model bulding.

In [33]:
```
from functools import reduce

data = [data_gdp, data_in, data_mortgage, data_msa, data_tlcons, data_unemp, data_cus]

# Merge DataFrames
df = reduce(lambda left, right: pd.merge(left, right, on='Year'), data)

df
```

Out[33]:

	Year	GDP_Growth(%)	Inflation_Rate(%)	Average_Mortgage_Rate(%)	MSACSR	TLRESCONS	Unemployment_Rate(%)
0	2003	2.80	2.27	5.83	3.91	450241.17	5.99
1	2004	3.85	2.68	5.84	4.00	536923.00	5.53
2	2005	3.48	3.39	5.87	4.45	628863.50	5.08
3	2006	2.78	3.23	6.41	6.43	617260.17	4.62
4	2007	2.01	2.85	6.34	8.38	497164.00	4.62
5	2008	0.12	3.84	6.03	10.68	367000.08	5.78
6	2009	-2.60	-0.36	5.04	9.03	261395.50	9.25
7	2010	2.71	1.64	4.69	8.00	256535.67	9.63
8	2011	1.55	3.16	4.45	6.58	255208.58	8.95
9	2012	2.28	2.07	3.66	4.76	278995.58	8.07
10	2013	1.84	1.46	3.98	4.74	335207.33	7.37
11	2014	2.29	1.62	4.17	5.48	382812.25	6.17
12	2015	2.71	0.12	3.85	5.17	437998.08	5.28
13	2016	1.67	1.26	3.65	5.21	486102.25	4.87
14	2017	2.24	2.13	3.99	5.38	545873.58	4.36
15	2018	2.95	2.44	4.54	6.18	564342.50	3.90
16	2019	2.29	1.81	3.94	5.82	552999.25	3.67
17	2020	-2.77	1.23	3.10	4.62	644425.00	8.05
18	2021	5.95	4.70	2.96	5.50	808891.17	5.35
19	2022	2.06	8.00	5.34	8.45	927137.58	3.61

Here is our final data files contains features and label:

Year: Observation year.

GDP_Growth: Gross Domestic Product growth rate for last 20 years.

Inflation_Rate: Inflation is the rate of increase in prices over a given period of time. Calculated yearly growth rate.

Average_Mortgage_Rate: A mortgage rate is the interest rate charged for a home loan.

MSACSR: Monthly Supply of New Houses in the U.S. rate.

TLRESCONS: Total Construction Spending in Residential. Calculated in $.

Unemployment_Rate: Unemployment rate in U.S.

CSUSHPISA: S&P/Case-Shiller U.S. National Home Price Index. This represents the home price index for the U.S.

In [34]:
```
df.describe()
```

Out[34]:

	Year	GDP_Growth(%)	Inflation_Rate(%)	Average_Mortgage_Rate(%)	MSACSR	TLRESCONS	Unemploymer
count	20.00000	20.000000	20.000000	20.000000	20.000000	20.000000	
mean	2012.50000	2.010500	2.477000	4.684000	6.138500	491768.812000	
std	5.91608	1.952926	1.765682	1.083646	1.849191	181834.053952	
min	2003.00000	-2.770000	-0.360000	2.960000	3.910000	255208.580000	
25%	2007.75000	1.797500	1.580000	3.917500	4.755000	359051.892500	
50%	2012.50000	2.285000	2.200000	4.495000	5.490000	491633.125000	
75%	2017.25000	2.785000	3.177500	5.832500	6.935000	577571.917500	
max	2022.00000	5.950000	8.000000	6.410000	10.680000	927137.580000	

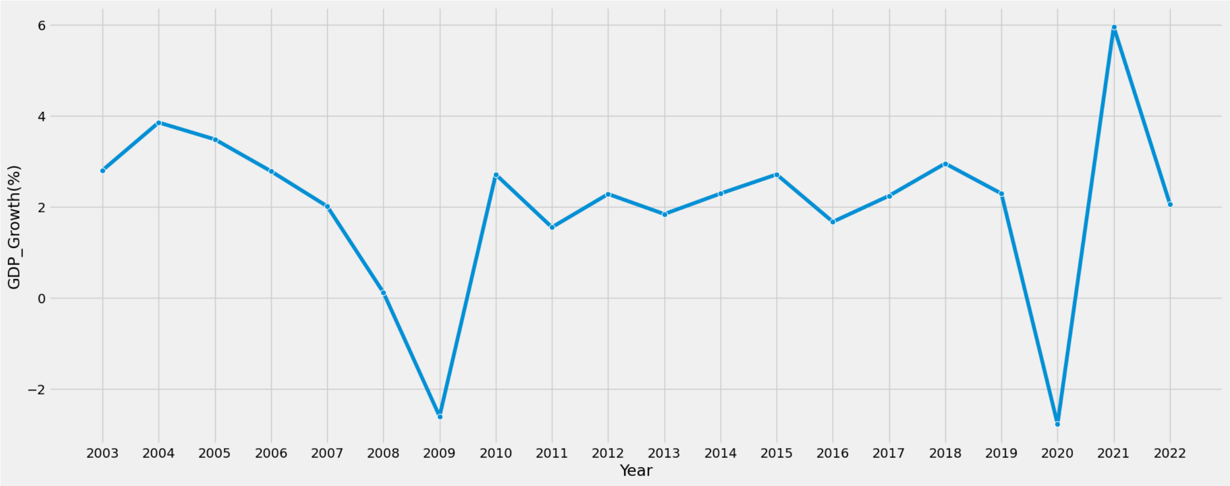
In [35]:
```
df.info()
```

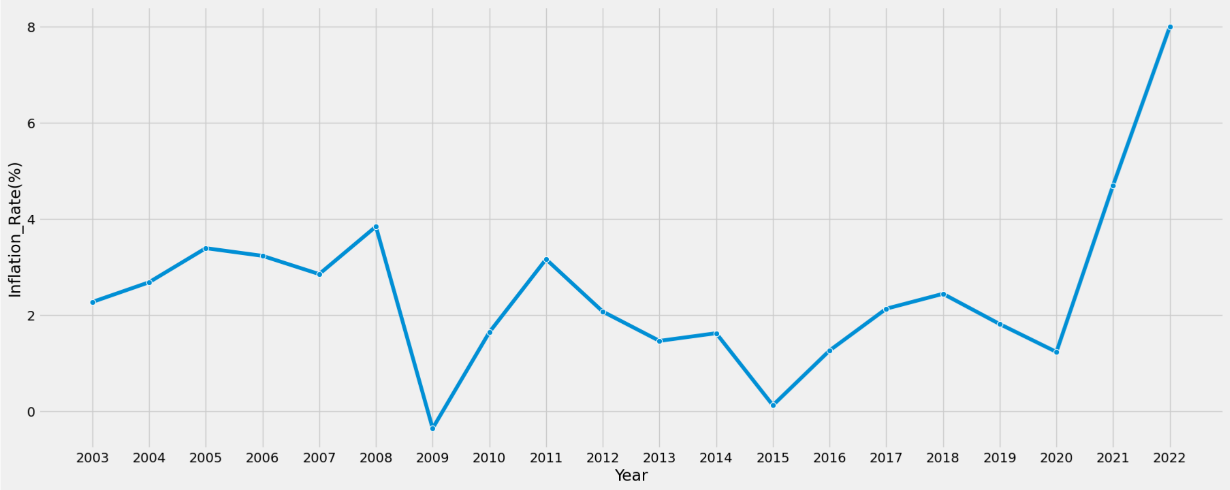
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Year                      20 non-null     int64  
 1   GDP_Growth(%)             20 non-null     float64
 2   Inflation_Rate(%)         20 non-null     float64
 3   Average_Mortgage_Rate(%)  20 non-null     float64
 4   MSACSR                    20 non-null     float64
 5   TLRESCONS                 20 non-null     float64
 6   Unemployment_Rate(%)      20 non-null     float64
 7   CSUSHPISA                 20 non-null     float64
dtypes: float64(7), int64(1)
memory usage: 1.4 KB
```

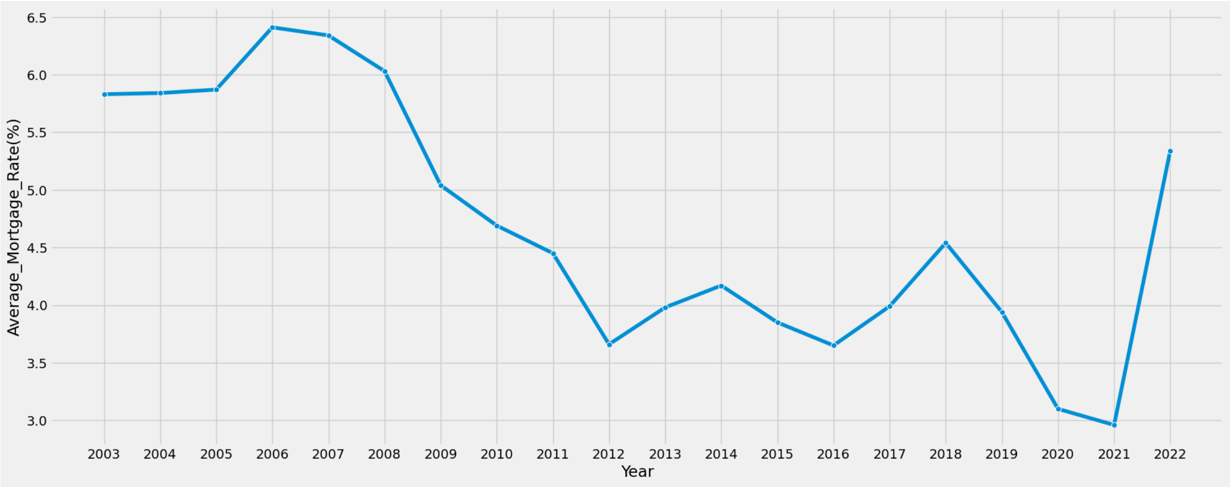
3. Exploratory Data Analysis and Visualization analysis

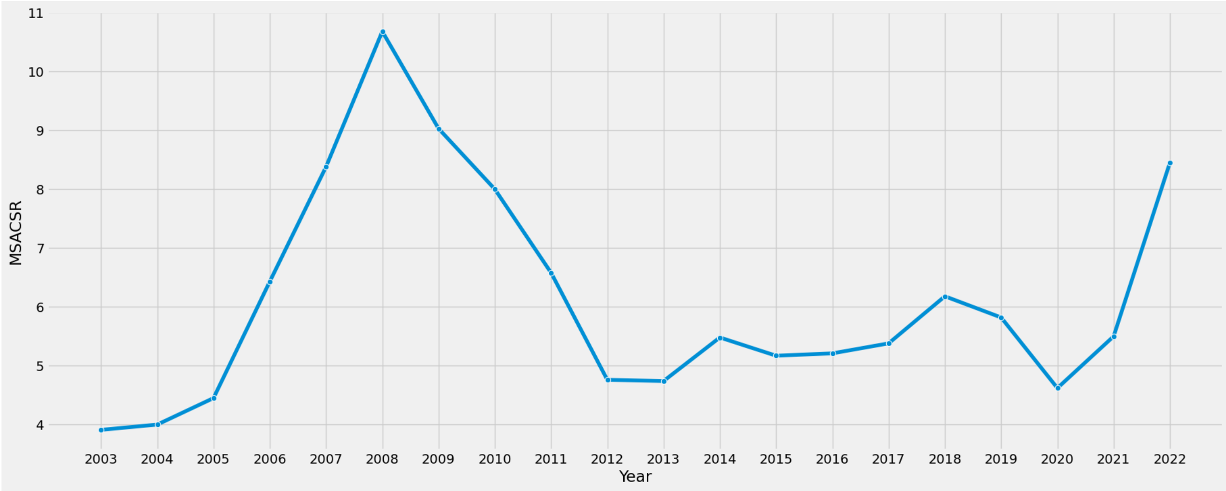
Year vs all Key Factors

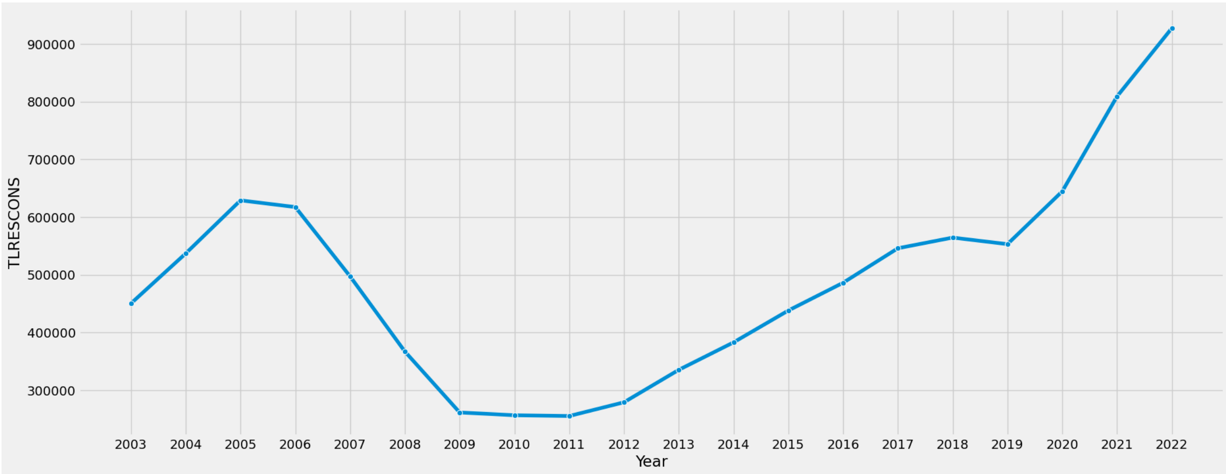
In [130]:
```
for column in df:
    if column!= 'Year':
        plt.figure(figsize=(20,8))
        plt.style.use('fivethirtyeight')
        sns.lineplot(x='Year',y=column, data = df, marker='o')
        plt.xticks([2003, 2004, 2005, 2006, 2007, 2008, 2009,2010,
                    2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022])
    plt.show()
```

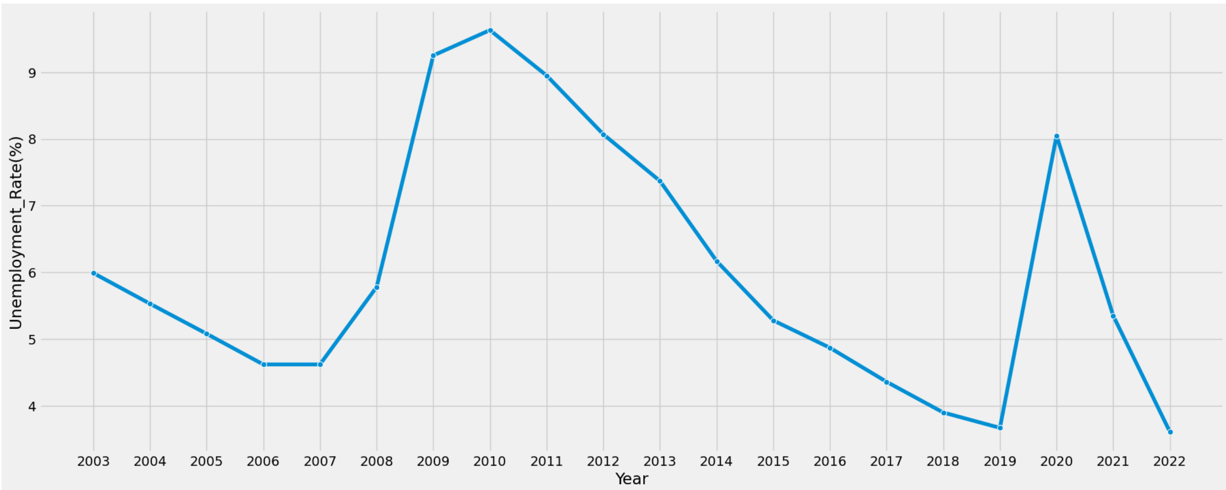


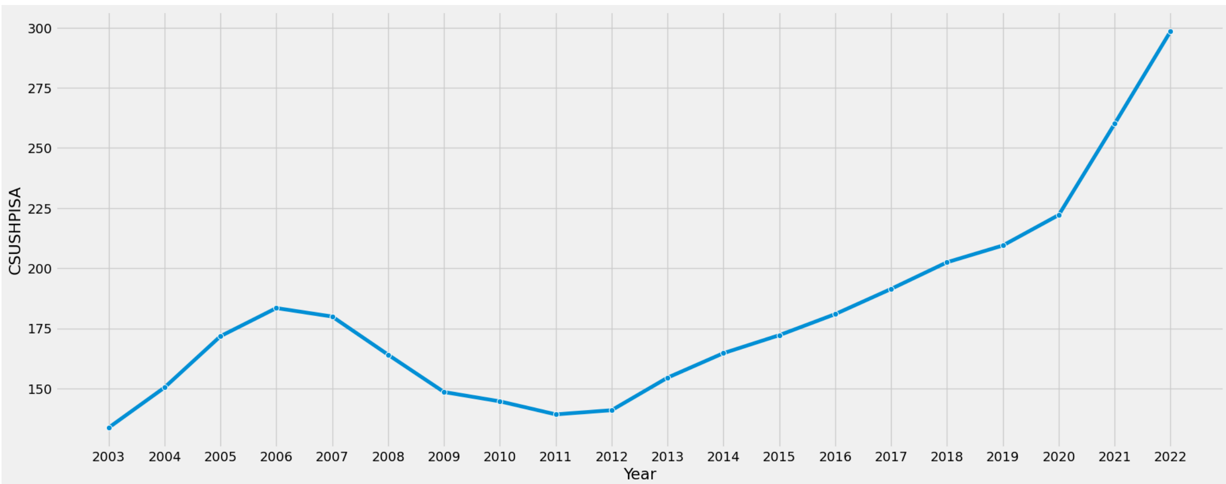





MSACSR
Year
2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
4 5 6 7 8 9 10 11

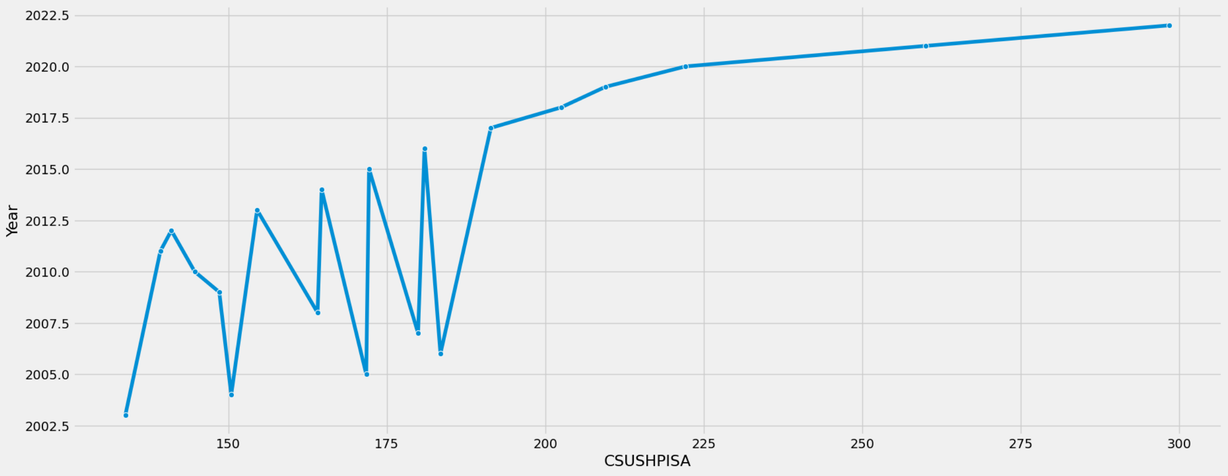

TLRESCONS
Year
2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
300000 400000 500000 600000 700000 800000 900000

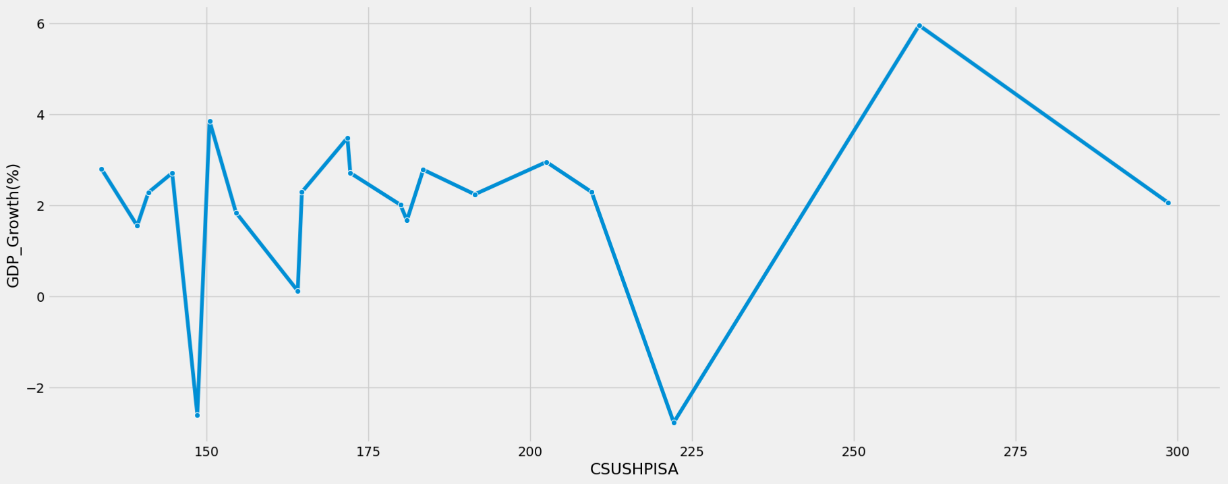

Unemployment_Rate(%)
Year
2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
4 5 6 7 8 9

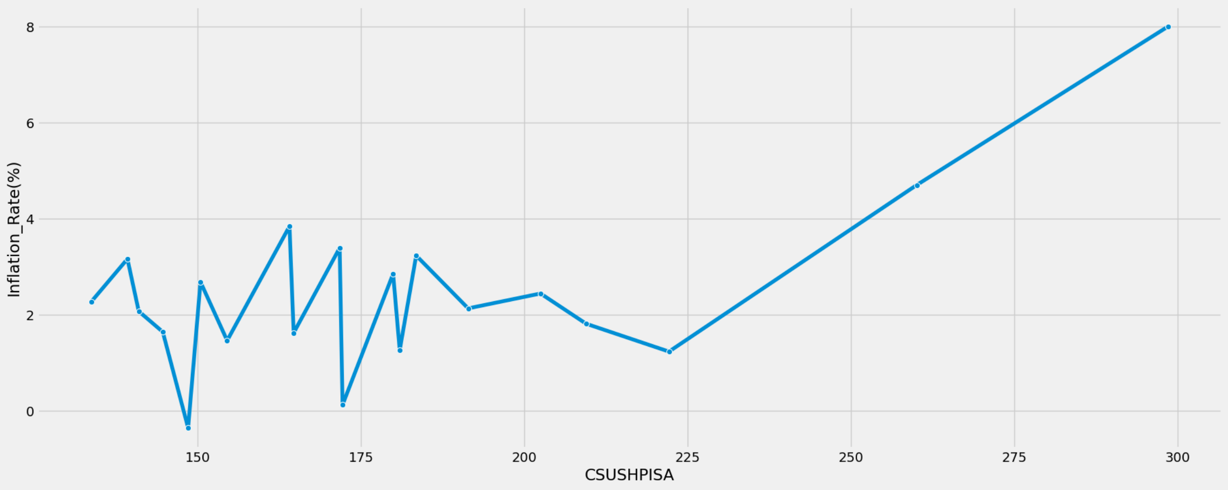

CSUSHPISA
Year
2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
150 175 200 225 250 275 300

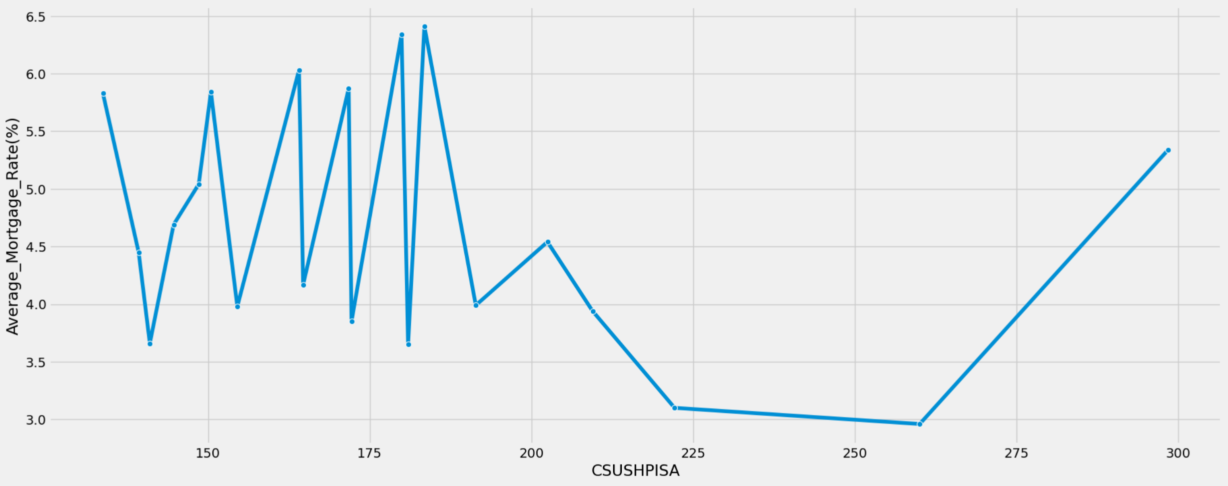
CSUSHPISA vs Key factors

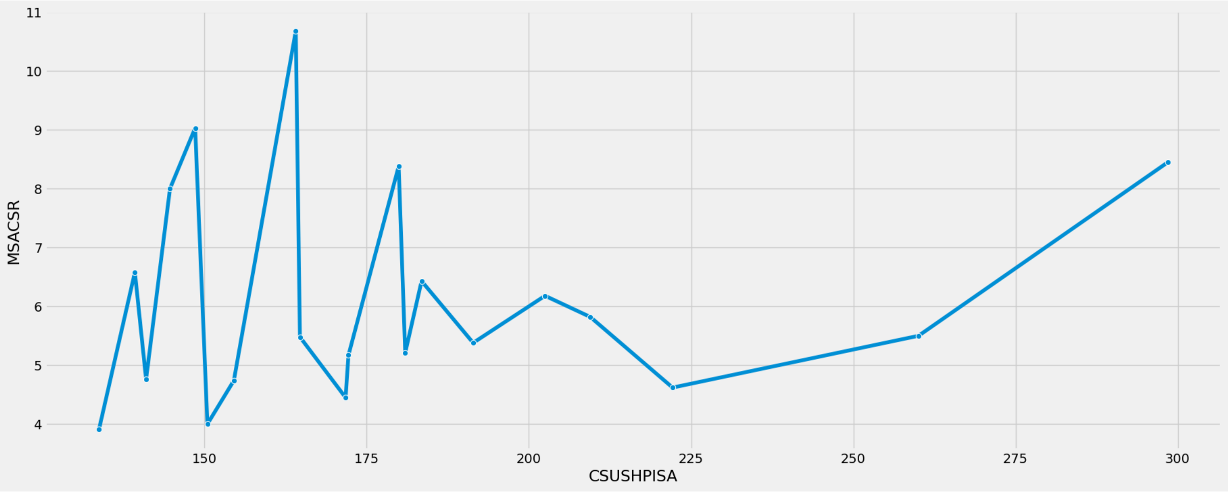
In [57]:
```
for column in df:
    if column!= 'CSUSHPISA':
        plt.figure(figsize=(20,8))
        sns.lineplot(x='CSUSHPISA',y=column, data = df, marker='o')
    plt.show()
```

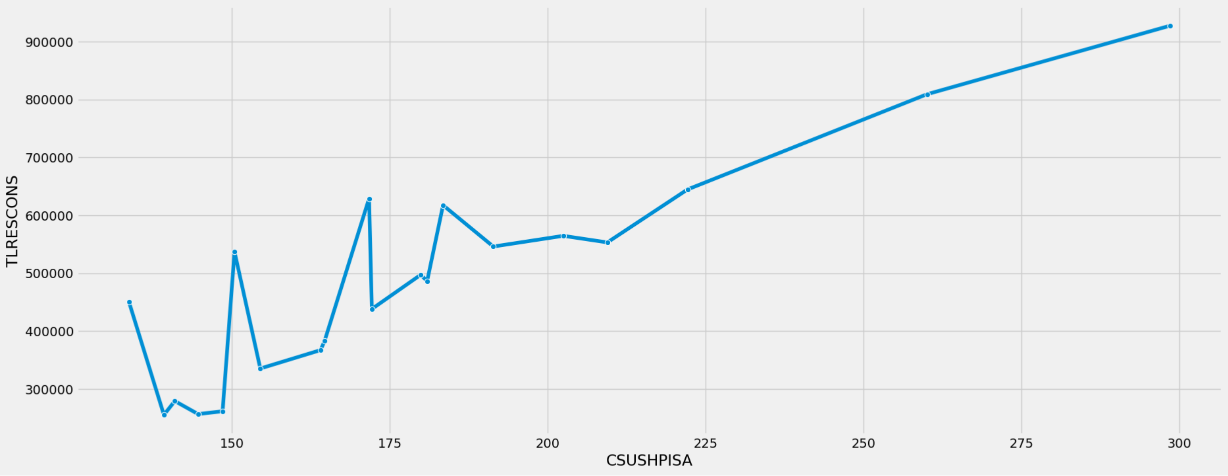


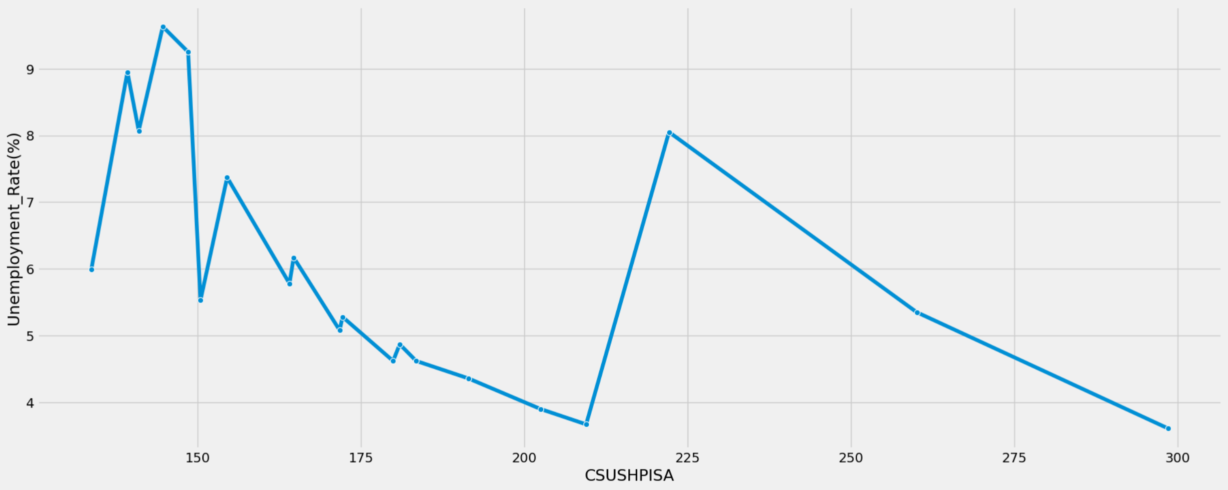





Average_Mortgage_Rate(%)
CSUSHPISA


MSACSR
CSUSHPISA


TLRESCONS
CSUSHPISA

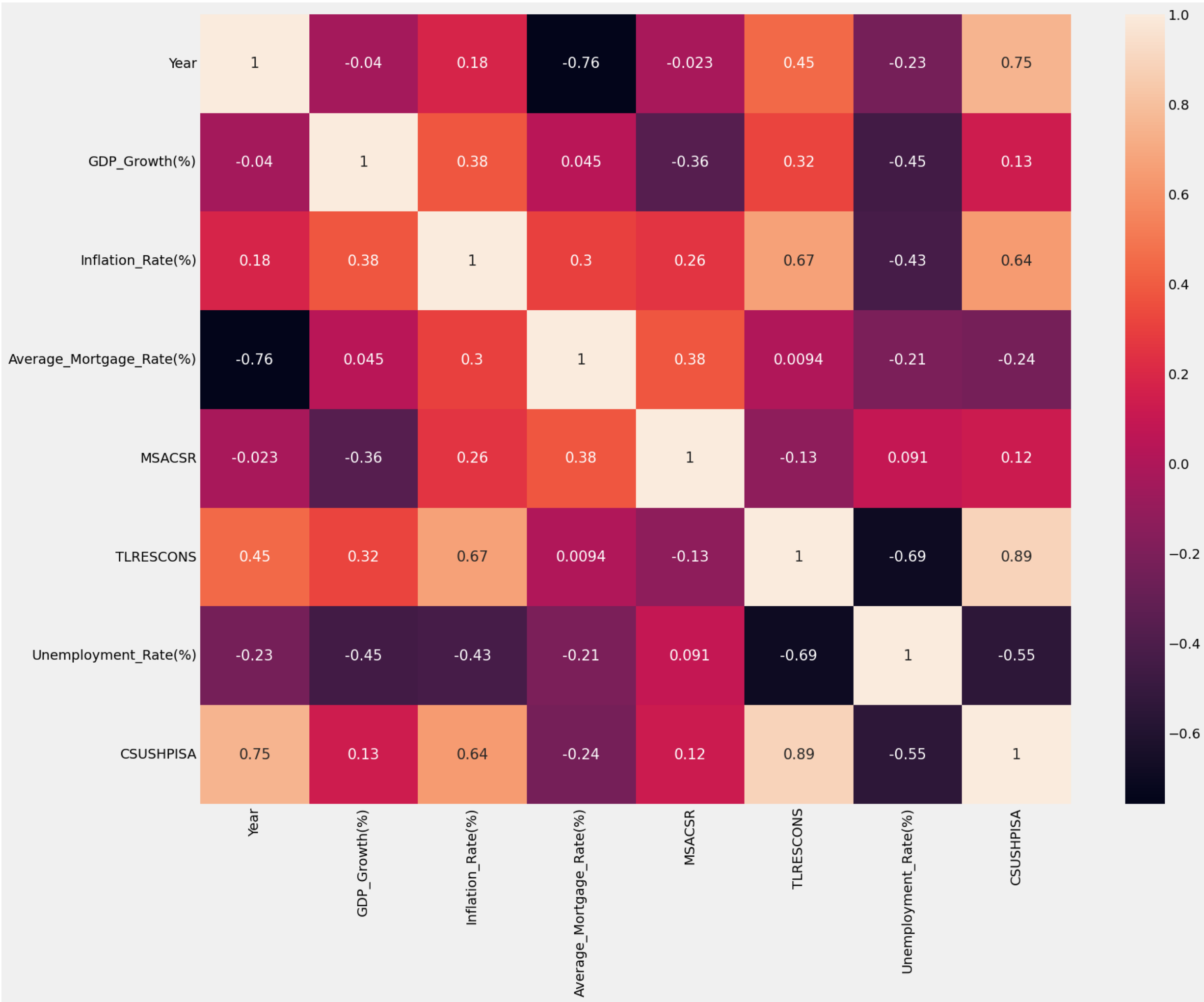

Unemployment_Rate(%)
CSUSHPISA

Ploting Heatmap for correlation CSUSHPISA vs all factors

In [58]:
```
# plotting Heatmap correlation

df_cor= df.corr() # checking relationship

plt.figure(figsize=(18,14))
sns.heatmap(df_cor, annot=True, annot_kws={'size':15})
plt.show()
```



In [103]:
```
correlation = df.corr()['CSUSHPISA']

correlation_df = pd.DataFrame(correlation).reset_index()

correlation_df.columns = ['Key_Factor', 'Correlation with CSUSHPISA']

correlation_df
```

Out[103]:

	Key_Factor	Correlation with CSUSHPISA
0	Year	0.746143
1	GDP_Growth(%)	0.131261
2	Inflation_Rate(%)	0.636309
3	Average_Mortgage_Rate(%)	-0.235378
4	MSACSR	0.123425
5	TLRESCONS	0.889444
6	Unemployment_Rate(%)	-0.547620
7	CSUSHPISA	1.000000

Here in our dataset TLRESCONS(Total Construction Spending in Residential) has strong positive correlation (0.889444). This means that there is a strong correlation between increased construction spending and higher housing prices.

Inflation_Rate has positive correlation (0.636309), higher rates and greater housing values.

MSACSR(Supply of New Houses in the United States) (0.123425) has positive correlation it means that the supply of new homes increases, it have a positive impact on home prices. Correleation is not strong so it may slightly increases on the prices.

Average_Mortgage_Rate (-0.235378) has a negative correlation it means that the supply of new homes could a bit raise housing prices as they increase.

GDP_Growth (0.131261) has week positive correlation it means that GDP increases, it may slightly increase the prices.

Unemployment_Rate(-0.547620) has a negative correlation it means that higher unemployment rates are slightly lower the prices.

4. Model building and Model evaluation

In [59]:
```
X=df.drop('CSUSHPISA',axis=1)
y= df.CSUSHPISA
```

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [61]:
```
# applying standard scaler
std_scalar = StandardScaler()

X_scaler= std_scalar.fit_transform(X)
```

In [62]:
```
x_train, x_test, y_train, y_test = train_test_split(X_scaler, y, test_size=0.2, random_state=42)
```

Importing different regressor models to find the best model.

In [116]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb

from sklearn.metrics import mean_squared_error as mse, mean_absolute_error as mae, r2_score as r2
#from sklearn.model_selection import cross_val_score
```

Linear Regression Model

In [105]:
```
Lin_reg = LinearRegression()
Lin_reg.fit(x_train, y_train) #training the algorithm
```

Out[105]:
```
▾ LinearRegression
LinearRegression()
```

In [106]:
```
y_predL = Lin_reg.predict(x_test)
```

In [107]:
```
print('Mean Absolute Error:', mae(y_test, y_predL))
print('Mean Squared Error:', mse(y_test, y_predL))
print('Root Mean Squared Error:', r2(y_test, y_predL))
```

```
Mean Absolute Error: 4.9872431874214485
Mean Squared Error: 33.42631944979831
Root Mean Squared Error: 0.9115800029443337
```

Decision Tree Regressor Model

In [108]:
```
# create a regressor object
DT_reg= DecisionTreeRegressor()

DT_reg.fit(x_train, y_train)

y_predD = DT_reg.predict(x_test)

print('Mean Absolute Error:', mae(y_test, y_predD))
print('Mean Squared Error:', mse(y_test, y_predD))
print('Root Mean Squared Error:', r2(y_test, y_predD))
```

```
Mean Absolute Error: 7.2775000000000105
Mean Squared Error: 58.69157500000018
Root Mean Squared Error: 0.8447478222516733
```

Random Forest Regressor Model

In [109]:
```
# create regressor object
Rf_reg = RandomForestRegressor()

# fit the regressor with x and y data
Rf_reg.fit(x_train, y_train)

y_predR = Rf_reg.predict(x_test)

print('Mean Absolute Error:', mae(y_test, y_predR))
print('Mean Squared Error:', mse(y_test, y_predR))
print('Root Mean Squared Error:', r2(y_test, y_predR))
```

```
Mean Absolute Error: 3.8205999999999705
Mean Squared Error: 19.647951634999636
Root Mean Squared Error: 0.948026828729209
```

Suppot Vector Regressor Model

In [110]:
```
# create the model object
SV_reg = SVR()

# fit the model on the data
SV_reg.fit(x_train, y_train)

y_predS = SV_reg.predict(x_test)

print('Mean Absolute Error:', mae(y_test, y_predS))
print('Mean Squared Error:', mse(y_test, y_predS))
print('Root Mean Squared Error:', r2(y_test, y_predS))
```

```
Mean Absolute Error: 18.691637492524663
Mean Squared Error: 489.40697723450467
Root Mean Squared Error: -0.29458953897356377
```

K-NN Model

In [111]:
```
KNN_reg = KNeighborsRegressor()
# fit the model using the training data and training targets
KNN_reg.fit(x_train, y_train)

y_predK = KNN_reg.predict(x_test)

print('Mean Absolute Error:', mae(y_test, y_predK))
print('Mean Squared Error:', mse(y_test, y_predK))
print('Root Mean Squared Error:', r2(y_test, y_predK))
```

```
Mean Absolute Error: 17.027999999999999
Mean Squared Error: 424.41769799999986
Root Mean Squared Error: -0.12267854269426914
```

XGBoost Model

In [117]:
```
xgb = xgb.XGBRegressor()

xgb.fit(x_train,y_train)
```

Out[117]:
```
XGBRegressor
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, device=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=None, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
multi_strategy=None, n_estimators=None, n_jobs=None,
num_parallel_tree=None, random_state=None, ...)
```

In [118]:
```
y_predX =xgb.predict(x_test)

print('Mean Absolute Error:', mae(y_test, y_predX))
print('Mean Squared Error:', mse(y_test, y_predX))
print('Root Mean Squared Error:', r2(y_test, y_predX))
```

```
Mean Absolute Error: 5.385199127197275
Mean Squared Error: 36.921702768325254
Root Mean Squared Error: 0.9023339421210131
```

In [119]:
```
result_data= pd.DataFrame(columns=['MAE','MSE','R2-score'])
```

In [120]:
```
result_data.loc['LinearRegression']=[mae(y_test,y_predL), mse(y_test,y_predL), r2(y_test,y_predL)]

result_data.loc['DecisionTreeRegressor']=[mae(y_test,y_predD), mse(y_test,y_predD), r2(y_test,y_pr

result_data.loc['Random Forest']=[mae(y_test,y_predR), mse(y_test,y_predR), r2(y_test,y_predR)]

result_data.loc['Support Vector Machines']=[mae(y_test,y_predS), mse(y_test,y_predS), r2(y_test,y_

result_data.loc['K-nearest Neighbors']=[mae(y_test,y_predK), mse(y_test,y_predK), r2(y_test,y_pred

result_data.loc['XGBoost']=[mae(y_test,y_predX), mse(y_test,y_predX), r2(y_test,y_predX)]
```

In [121]:
```
result_data
```

Out[121]:

	MAE	MSE	R2-score
LinearRegression	4.987243	33.426319	0.911580
DecisionTreeRegressor	7.277500	58.691575	0.844748
Random Forest	3.820600	19.647952	0.948027
Support Vector Machines	18.691637	489.406977	-0.294590
K-nearest Neighbors	17.028000	424.417698	-0.122679
XGBoost	5.385199	36.921703	0.902334

The average squared difference between the predicted and actual target values is measured by the Mean Squared Error (MSE). Better performance is indicated by a lower MSE because it represents smaller prediction errors.

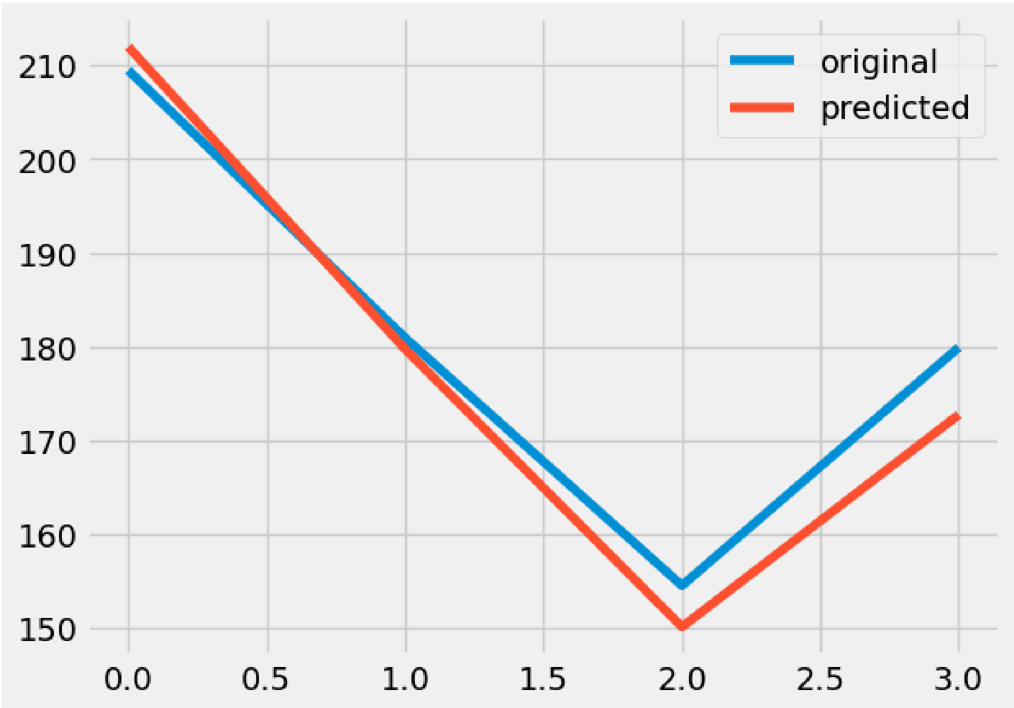
And, the R-2score, which evaluates the percentage of the target variable's variance that the model may be responsible for higher values denote a better fit. The range is 0 to 1.

In our analysis showed that the Random Forest Regression model worked well. With an MSE of 19.64 on the testing set, the prediction errors were quite small. Also, 0.948027 was the R-squared score. Pointing that the model is capable of accounting for approx. 97.23% of the target variable.

Actual vs Predicted line for Random Forest model.

In [129]:
```
#Random Forest

x_ax = range(len(y_test))
#plt.figure(figsize=(10,5))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, y_predR, label="predicted")
plt.legend()
plt.show()
```



Accept user input for the new data for the prediction.

In [131]:
```
# user input for the new data
user_input = {
    'Year': int(input('Enter Year: ')),
    'GDP_Growth(%)': float(input('Enter GDP Growth: ')),
    'Inflation_Rate(%)': float(input('Enter Inflation Rate: ')),
    'Average_Mortgage_Rate(%)': float(input('Enter Average Mortgage Rate: ')),
    'MSACSR': float(input('Enter MSACSR: ')),
    'TLRESCONS': float(input('Enter TLRESCONS: ')),
    'Unemployment_Rate(%)': float(input('Enter Unemployment Rate: '))
}

user_df = pd.DataFrame([user_input])

# Standardize the new data
user_df_scaled = std_scalar.fit_transform(user_df)

# Predict the new data
prediction = Rf_reg.predict(user_df_scaled)
print(f'\n Predicted S&P/Case-Shiller U.S. National Home Price Index (CSUSHPISA) for {user_input["
```

```
Enter Year: 2024
Enter GDP Growth: 2.2
Enter Inflation Rate: 2.3
Enter Average Mortgage Rate: 4.1
Enter MSACSR: 3.0
Enter TLRESCONS: 24000
Enter Unemployment Rate: 4.3
Predicted CSUSHPISA for 2024: 149.81470000000004
```

Saving Random Forest model, it give the best result

In [132]:
```
import pickle
pickle.dump(Rf_reg, open('Assessment_Home.LLC.pkl', 'wb'))
```

In [133]:
```
pickled_model = pickle.load(open('Assessment_Home.LLC.pkl', 'rb'))
pickled_model.predict(x_test)
```

Out[133]:
```
array([211.9619, 179.7318, 150.1112, 172.7465])
```

Conclusion

The analysis found that factors such as increased construction spending, inflation rate, and the greater supply of new houses tend to have positive correlations with higher housing prices. On the other side, factors like higher mortgage rates and unemployment rates may exert a slight downward pressure on housing prices. Economic indicators like GDP growth may have a modest positive influence on home prices.

In [ ]:

In [ ]: