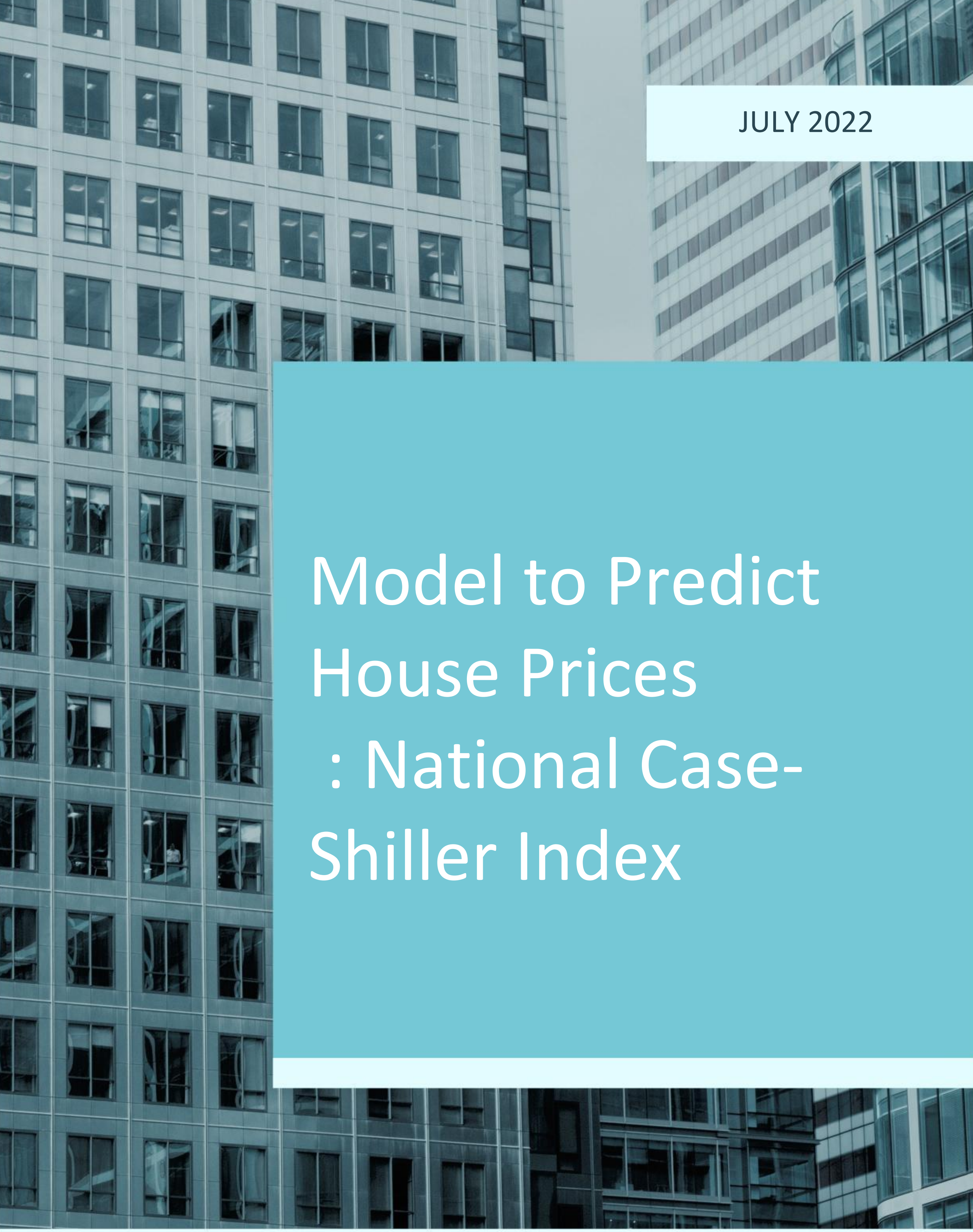


JULY 2022

Model to Predict House Prices : National Case-Shiller Index

By – Sarth Mirashi

Introduction

The primary objective of this study is to build a model to predict the nation-wide US house prices with national Case -Shiller Index as a reference pointer for overall house price trend.

Publicly available Data Used :

1. **Consumer Price Index CPI** - https://fred.stlouisfed.org/series/CPIAUCSL
2. **Unemployment rate** - https://fred.stlouisfed.org/series/UNRATE
3. **30-Year Fixed Rate Mortgage** - https://fred.stlouisfed.org/series/MORTGAGE30US
4. **Gross Domestic Product** - https://fred.stlouisfed.org/series/GDP
5. **Total Unit Labor Cost** - https://fred.stlouisfed.org/series/LCULMN01USQ661S
6. **Personal current taxes: State and local: Property taxes-** https://fred.stlouisfed.org/series/S210401A027NBEA
7. **Employment-Population Ratio** - https://fred.stlouisfed.org/series/EMRATIO
8. **NASDAQ Composite Index -** https://fred.stlouisfed.org/series/NASDAQCOM
9. **Disposable Personal Income -** https://fred.stlouisfed.org/series/DSPI
10. **Rental Vacancy Rate -** https://fred.stlouisfed.org/series/RRVRUSQ156N
11. **Monthly Supply of New Houses -** https://fred.stlouisfed.org/series/MSACSR
12. **New Privately-Owned Housing Units Completed** - https://fred.stlouisfed.org/series/COMPUTSA

Code - https://colab.research.google.com/drive/1fwbMUFUTI8ewy2tdCv0Ro452XS9DcQ8U?usp=sharing

DATA:

The above mentioned data was imported and the date was parsed and converted to panda datetime and used as the index of the dataframe. The time-step of the target i.e. Case shiller index is a month thus all the data is preferably collected in this format, if the frequency of the data is higher than monthly, monthly data is used and the rest is discarded. However if the frequency is lower than monthly. Monthly data was filled in using linear interpolation. Quarterly data like GDP were linearly interpolated.

```
factor_quat = {
         'GDP':GDP,
         'Labour_cost':Labour_cost,
         'Rental_vacancy':Rental_vacancy,
         'prop_taxes':prop_taxes,
    }
```

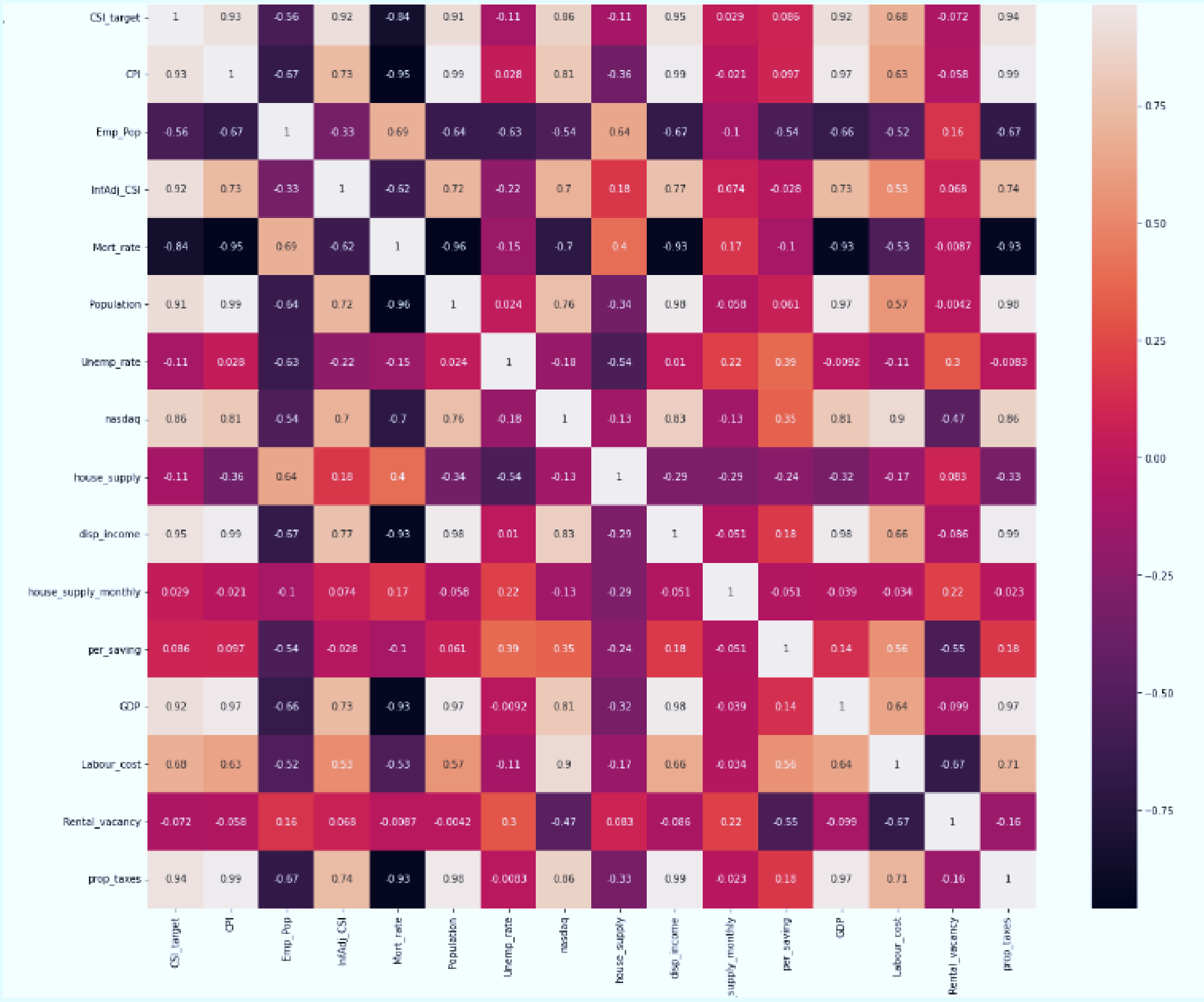
```
for name,fact in factor_quat.items():                    #interpolation
  df_quat[name].interpolate(method='linear', inplace=True)
```

	CSI_target	CPI	Emp_Pop	InfAdj_CSI	Mort_rate	Population	Unemp_rate	nasdaq	house_supply	disp_income	house_supply_monthly	per_saving	GDP	Labour_cost	Rental_vacancy	prop_taxes
date																
1987-01-01	63.735	111.400	61.0	62.19553	9.2040	241857.0	6.6	384.227142	136.7	6159.5	6.0	9.7	4722.156000	88.040121	7.400000	2409.000000
1987-02-01	64.134	111.800	61.1	62.41771	9.0825	242005.0	6.6	411.712846	117.8	6192.1	6.2	8.5	4750.157333	87.713226	7.433333	2420.000000
1987-03-01	64.470	112.200	61.2	62.49543	9.0350	242166.0	6.6	432.204559	126.4	6200.0	6.0	8.5	4778.158667	87.386331	7.466667	2431.000000
1987-04-01	64.974	112.700	61.3	62.59973	9.8325	242338.0	6.3	422.771423	135.6	5967.2	6.0	4.5	4806.160000	87.059437	7.500000	2442.000000
1987-05-01	65.549	113.000	61.6	62.84859	10.5960	242516.0	6.3	416.634003	131.4	6209.1	6.7	8.2	4832.291667	87.039587	7.700000	2459.666667
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2021-12-01	278.694	280.126	59.5	108.32838	3.0980	332640.0	3.9	15474.431641	127.3	15442.7	5.6	8.7	24258.761000	125.694729	5.733333	11485.666667
2022-01-01	282.069	281.933	59.7	109.40715	3.4450	332684.0	4.0	14531.377930	87.1	15163.5	5.7	5.8	24386.734000	126.274703	5.800000	11501.000000
2022-02-01	287.304	284.182	59.9	110.70095	3.7625	332750.0	3.8	13898.727539	98.6	15173.6	6.0	5.8	18350.841500	125.875439	5.797037	11501.000000
2022-03-01	294.721	287.708	60.1	111.65551	4.1720	332812.0	3.6	13623.262695	112.7	15119.6	6.9	5.3	12314.949000	125.476175	5.794074	11501.000000
2022-04-01	300.845	288.663	60.0	112.95519	4.9825	332863.0	3.6	13394.163086	107.4	15154.4	8.3	5.2	6279.056500	125.076911	5.791111	11501.000000

424 rows × 16 columns

Correlation Heatmap :

High inter correlation is seen. This may be attributed to the fact that most of the features considered have a general upward trend with respect to time.



Regression Approach :

Lagged features expect for the past target variable itself. This will help analyse the target variables dependence on the lagged features (past 1 month).

Converting Timeseries forecasting problem to Supervised Learning.:

Target feature:

Inflation adjusted National Case Shiller index was used as the target variable in the regression approach as it showed better results than using vanilla National CS index.

$$\frac{National\ Case - Shiller\ Index}{CPI} X\ 100$$

Lagged feature:

All the features used in the regression analysis were lagged features all shift by two months. That is second last month's features were used to predict the target variable. This will allow us to predict two months into the future.

The previous value of the target variable were not used so as to better study the impact of the factors on the Case shiller index.

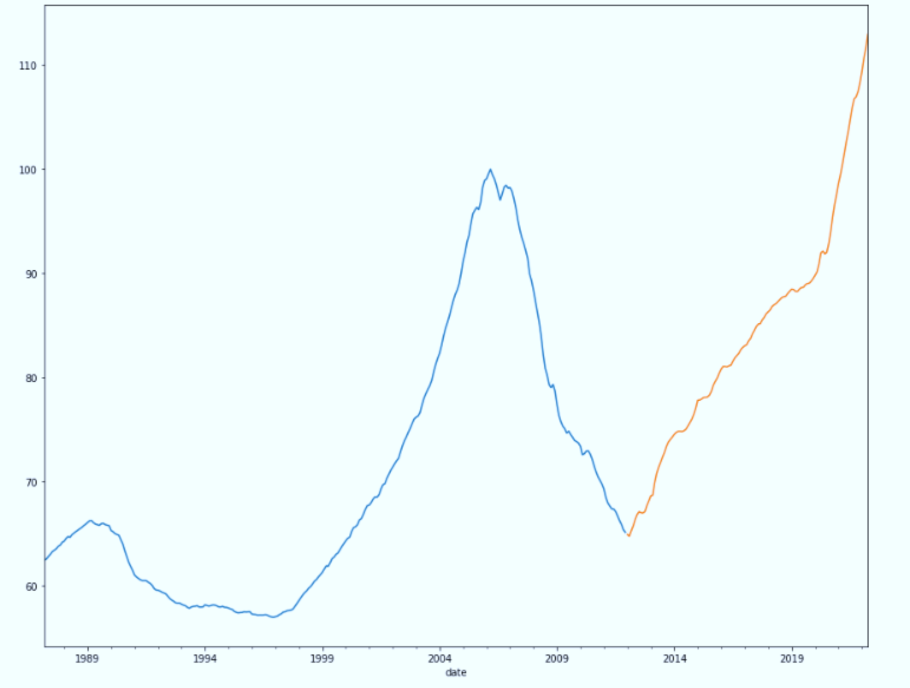
```
target = df1.InfAdj_CSI
df1 = df1.shift(2)
```

Train- test split :

The data was split into train and test set to validate the performance. And since this problem is semi - time series problem random sampling for the split is ill advised. This is done to avoid data leakage and overfit.

```
X_train = df1.loc[df1.index < '2012-01-01']
X_test  = df1.loc[df1.index >= '2012-01-01']
Y_train = target.loc[df1.index < '2012-01-01']
Y_test = target.loc[df1.index >= '2012-01-01']
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fddc43126d0>
```



This split from 2012 stays consistent through all the models.

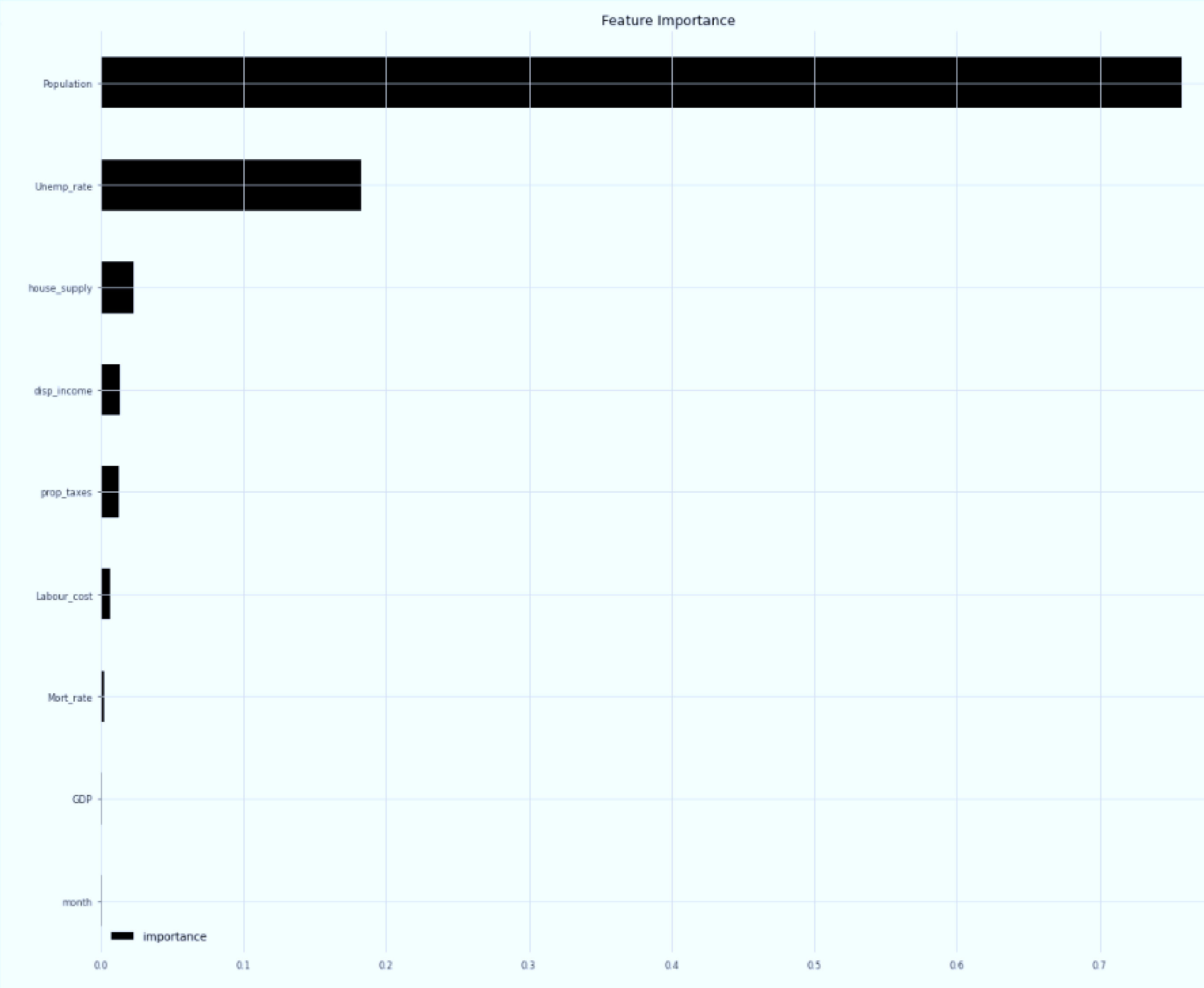
XGBoost :

XGboost Algorithm was my go to choice for a semi time series problem. As It is versatile and has seen success in the multivariate time series forecasting.

```
reg = xgb.XGBRegressor(n_estimators=3000, early_stopping_rounds = 50, learning_rate = 0.01)
reg.fit(X_train, Y_train,
        eval_set = [(X_train,Y_train), (X_test,Y_test)],
       verbose = 1000 )
```

```
[14:57:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[0]	validation_0-rmse:69.8338	validation_1-rmse:83.5104
[1000]	validation_0-rmse:0.234793	validation_1-rmse:15.4801
[2000]	validation_0-rmse:0.112942	validation_1-rmse:15.5299
[2999]	validation_0-rmse:0.067642	validation_1-rmse:15.4966
XGBRegressor(early_stopping_rounds=50, learning_rate=0.01, n_estimators=3000)
```

Feature Importance:

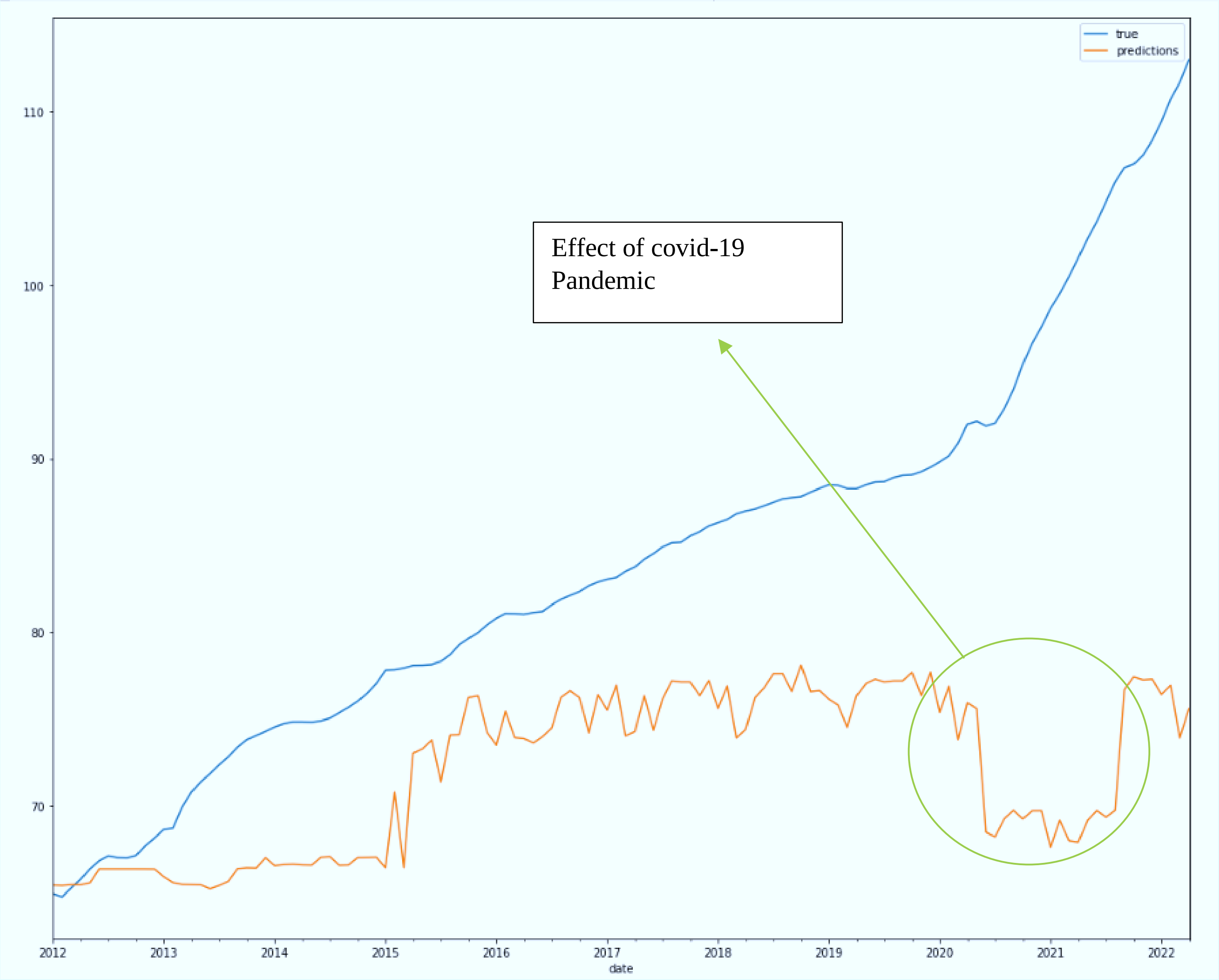


XGBoost allocates feature importance to the features that do well in predicting the target. It was found population is the best in predicting CS index. Considering the feature importance and After iterating through a bunch of feature combinations. Following features were selected in the final model :

	Mort_rate	Population	Unemp_rate	house_supply	disp_income	GDP	Labour_cost	prop_taxes	month
date									
1987-03-01	9.2040	241857.0	6.6	136.7	6159.5	4722.156000	88.040121	2409.000000	3
1987-04-01	9.0825	242005.0	6.6	117.8	6192.1	4750.157333	87.713226	2420.000000	4
1987-05-01	9.0350	242166.0	6.6	126.4	6200.0	4778.158667	87.386331	2431.000000	5
1987-06-01	9.8325	242338.0	6.3	135.6	5967.2	4806.160000	87.059437	2442.000000	6
1987-07-01	10.5960	242516.0	6.3	131.4	6209.1	4832.291667	87.039587	2459.666667	7

Predictions :

The predictions of the model were compared to the test data labelled as true. Root mean squared error of the prediction is **15.5** compared to a standard deviation of 11.5 of the target variable InfAdj_CSI



Auto ML:

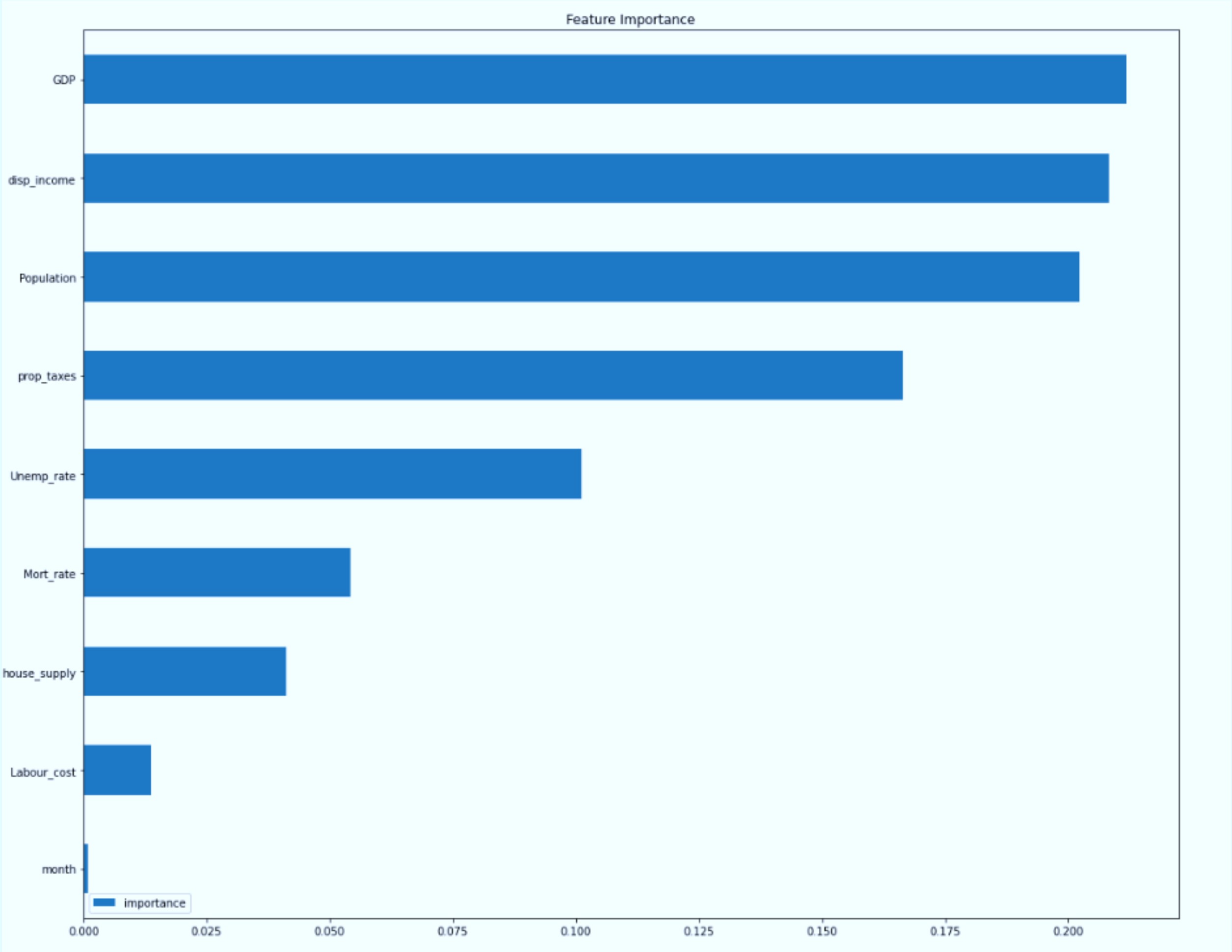
Pycaret - Is an autoML python library that tests a bunch of different algorithms on the data and lists the best algorithms. It was found that extra trees regressor was best for the purpose.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	0.3579	4.857000e-01	0.6255	0.9977	0.0071	0.0044	0.456
knn	K Neighbors Regressor	0.4676	1.777100e+00	0.9981	0.9924	0.0107	0.0056	0.062
rf	Random Forest Regressor	0.7098	1.756800e+00	1.2467	0.9910	0.0148	0.0089	0.542
gbr	Gradient Boosting Regressor	0.9729	2.377400e+00	1.4543	0.9882	0.0174	0.0123	0.110
dt	Decision Tree Regressor	0.8447	2.954500e+00	1.5406	0.9840	0.0185	0.0106	0.016
lightgbm	Light Gradient Boosting Machine	1.1133	3.754600e+00	1.8152	0.9816	0.0200	0.0134	0.070
ada	AdaBoost Regressor	1.8507	5.176600e+00	2.2668	0.9729	0.0302	0.0255	0.108
lr	Linear Regression	3.0086	1.691670e+01	4.0013	0.9144	0.0501	0.0400	0.323

Extra Trees Regressor :

As suggest by Pycaret Extra trees regressor was applied to the problem.

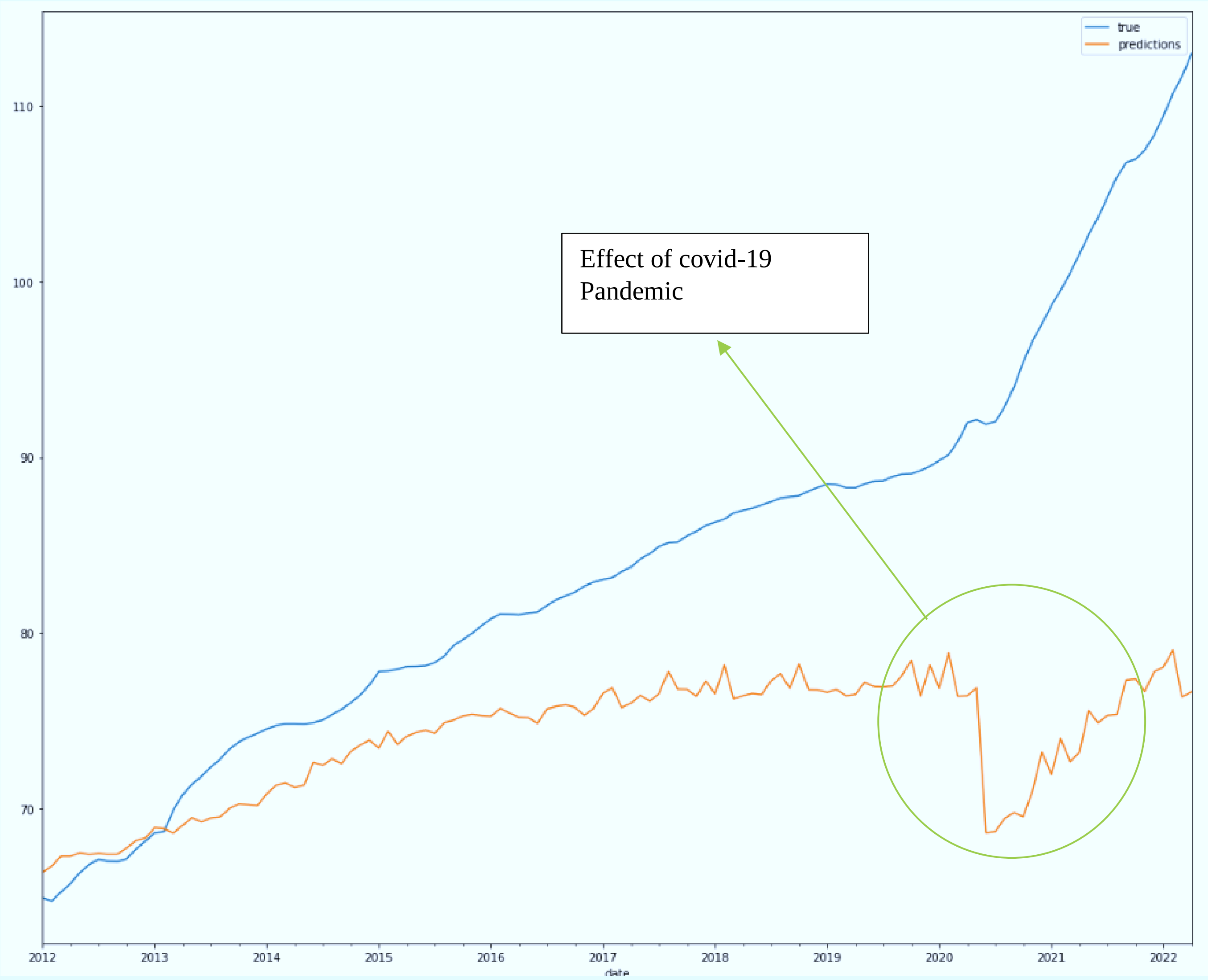
Feature importance:



The feature importance observed in Extra trees was different from XGBoost

Predictions :

The predictions of the model were compared to the test data labelled as true. Root mean squared error of the prediction is **13.9** compared to a standard deviation of 11.5 of the target variable InfAdj_CSI



Bagging Ensemble :

In this technique Trained models are assigned weights according to their performance and weighted average of their predictions is considered as the prediction of the ensemble. The models used in the ensemble are as follows :

```
[57] models = {'gbr':GradientBoostingRegressor(n_estimators=10000, learning_rate = 0.01),
               'br':BayesianRidge(),
               'xgb': xgb.XGBRegressor(n_estimators=3000, early_stopping_rounds = 50, learning_rate = 0.001,verbose = 0),
               'etr':ExtraTreesRegressor(n_estimators=3000)}
```

```
[58] for name, model in models.items():
       model.fit(X_train, Y_train)
```

Cross validation:

KFold cross validation was used. The root mean square value for each individual model is as mentioned in the figure

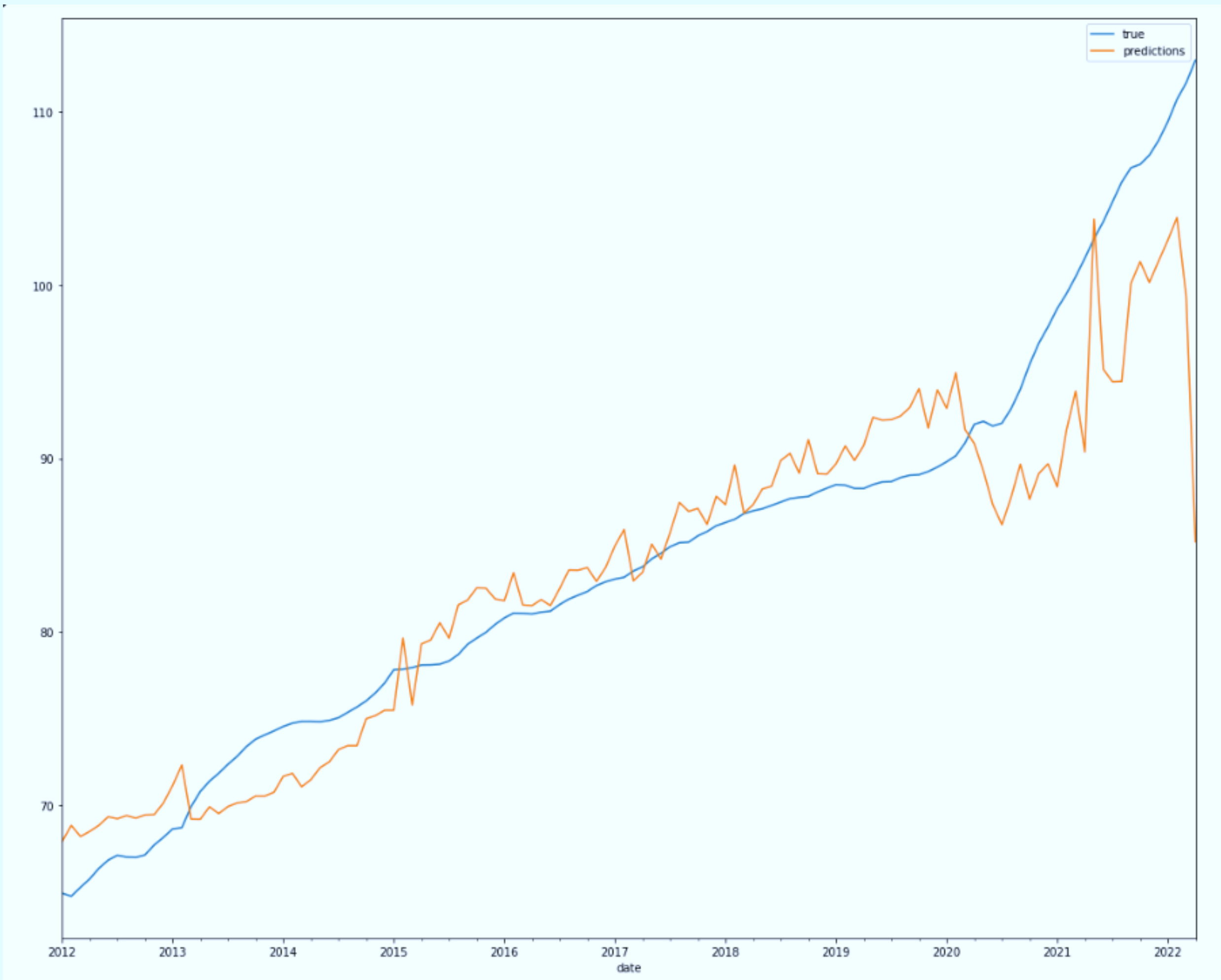
```
results = {}

kf = KFold(n_splits=10)

for name, model in models.items():
    result = np.sqrt(-cross_val_score(model, df1, target, scoring='neg_mean_squared_error', cv=kf))
    results[name] = result
```

```
----------
gbr
5.190255777086366
4.482449281626742
----------
br
7.346395731053842
3.8106547123155434
----------
xgb
7.452391132553402
6.537198307354843
----------
etr
4.208012381326437
4.504610177953215
```

```
y_pred = (
    0.15 * models['gbr'].predict(X_test) +
    0.25 *models['br'].predict(X_test) +
    0.4 *models['etr'].predict(X_test) +
    0.2 * models['xgb'].predict(X_test))
```

Predictions :

The predictions of the model were compared to the test data labelled as true. Root mean squared error of the prediction is **4.64** compared to a standard deviation of 11.5 of the target variable InfAdj_CSI



Vector autoregression:

Vector Autoregression (VAR) is a forecasting algorithm that can be used when two or more time series influence each other. That is, the relationship between the time series involved is bi-directional

Darts Library :

Darts is a Python library for easy manipulation and forecasting of time series. It contains a variety of models, from classics such as ARIMA to deep neural networks.

ADfuller test : Adfuller test is used to check for stationarity of the data. The ADfuller test was performed .The data as is not stationary however , after differencing the data once. The data is not stationary

Differencing :

```
def adftest(df):

  for fact in df:
    print('\n-------{}-------\n'.format(fact))
    adf = adfuller(df[fact])
    print(f'ADF Statistic: {adf[0]}')
    print(f'p-value: {adf[1]}')

adftest(df2.diff(1)[1:])
```

ADfuller test results :

```
-------InfAdj_CSI-------

ADF Statistic: -2.9127271274156596
p-value: 0.043887896852186636

-------Mort_rate-------

ADF Statistic: -10.322781972846803
p-value: 2.9788996059807623e-18

-------Unemp_rate-------

ADF Statistic: -12.1275063492022
p-value: 1.7667530500682174e-22

-------disp_income-------

ADF Statistic: -4.462838431231033
p-value: 0.00022938490894836274

-------GDP-------

ADF Statistic: -8.749766128154748
p-value: 2.857550554251797e-14

-------prop_taxes-------

ADF Statistic: -3.7979600886659948
p-value: 0.002928255045692676
```

Thus stationary

Varima Model :

After some hyperparameter tuning following parameters were selected for the final model :

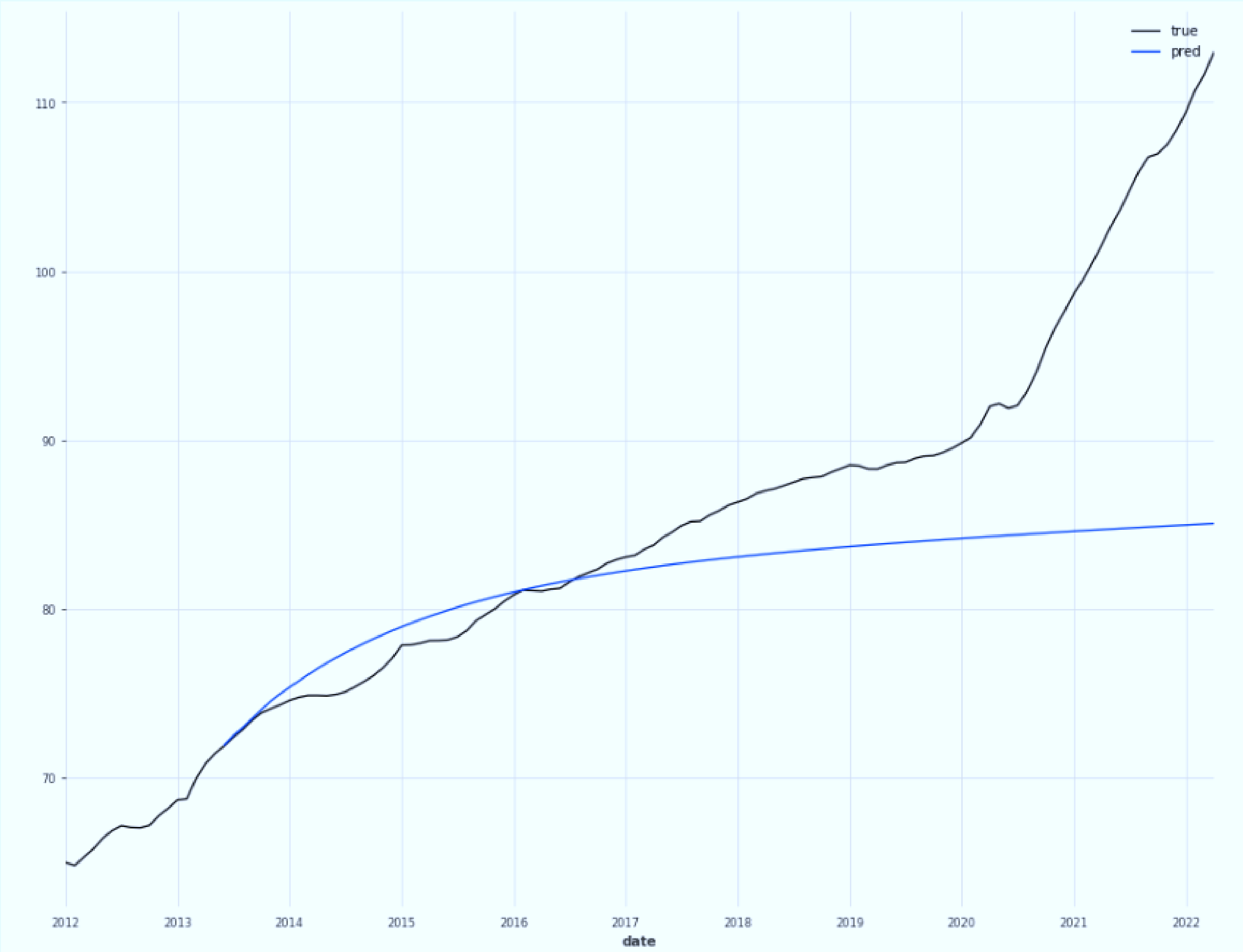
P = 3, d = 1, q = 0.Since q was zero the moving average component was not used making the model a VAR model.

```
model1 = VARIMA(p = 4, d = 1, q = 0)
model1.fit(X_train)
pred1 = model1.predict(len(X_test))
pred1.to_csv('varima')
final = pd.read_csv('/content/varima')
final.index = pd.to_datetime(final['date'])
```

- **p** (*int*) – Order (number of time lags) of the autoregressive model (AR)
- **d** (*int*) – The order of differentiation; i.e., the number of times the data have had past values subtracted. (I) Note that Darts only supports d <= 1 because for d > 1 the optimizer often does not result in stable predictions. If results are not stable for d = 1 try to set d = 0 and enable the trend parameter to account for possible non-stationarity.
- **q** (*int*) – The size of the moving average window (MA).

Predictions :

The predictions of the model were compared to the test data labelled as true. In this case predictions were made about **10 years in the future**. Root mean squared error of the prediction is 9.06 compared to a standard deviation of 11.5 of the target variable InfAdj_CSI



LSTM :

LSTM model of the following specifications was developed however failed to delivered due to some errors.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 4, 128)            70144

 leaky_re_lu_4 (LeakyReLU)   (None, 4, 128)            0

 lstm_7 (LSTM)               (None, 4, 128)            131584

 leaky_re_lu_5 (LeakyReLU)   (None, 4, 128)            0

 dropout_4 (Dropout)         (None, 4, 128)            0

 lstm_8 (LSTM)               (None, 64)                49408

 dropout_5 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 251,201
Trainable params: 251,201
Non-trainable params: 0
_________________________________________________________________
```