APS1052 Project

Option 3 - Yield Curve Prediction

Fengyi Xiao 1002956232

Tongjing Weng 1002920413

Xuanting Liu 1003017638

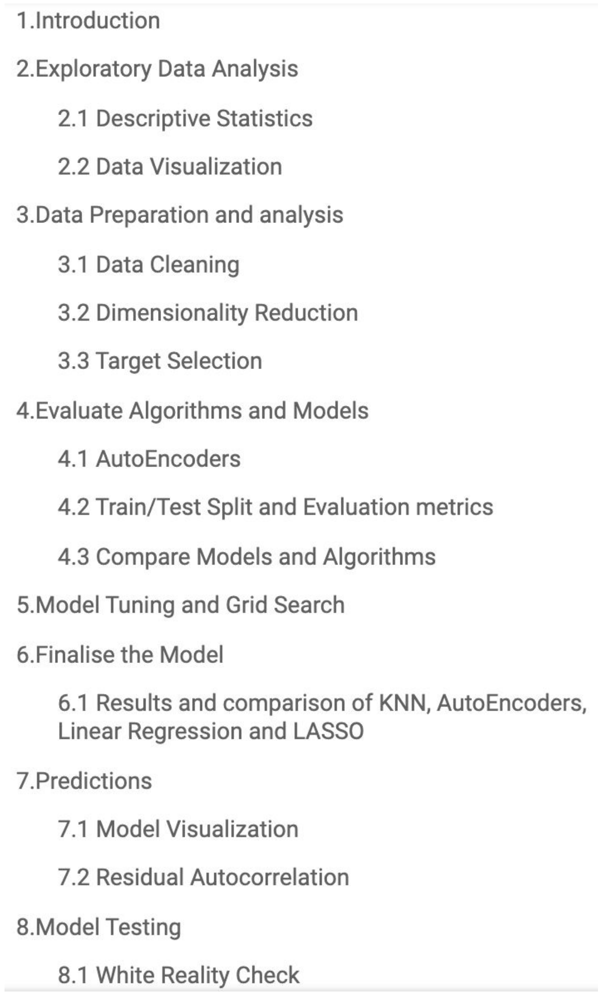
Yushi Chang 1003886529

Sources

- Seed programs
    - https://github.com/tatsath/fin-ml/blob/master/Chapter%205%20-%20Sup.%20Learning%20-%20Regression%20and%20Time%20Series%20models/Case%20Study%204%20-%20Yield%20Curve%20Prediction/YieldCurvePrediction.ipynb
    - https://github.com/tatsath/fin-ml/blob/master/Chapter%207%20-%20Unsup.%20Learning%20-%20Dimensionality%20Reduction/CaseStudy2%20-%20Yield%20Curve%20Construction%20and%20Interest%20Rate%20Modeling/YieldCurveConstruction.ipynb
    - 12-AutoEncoders_spot_rates.ipynb
- Source of the data
    - https://wrds-www.wharton.upenn.edu/pages/get-data/center-research-security-prices-crsp/annual-update/index-treasury-and-inflation/us-treasury-and-inflation-indexes/

Improvements

- Perform dimensionality reduction by using Autoencoders instead of PCA.
- Identify the most reasonable targets for prediction.
- Evaluate algorithms and models by comparing their train and validation metrics.
- Apply Grid Search to select best models.
- Predict the target and visualize the models performance.
- Test models by using White Reality Check.



Inputs and Targets

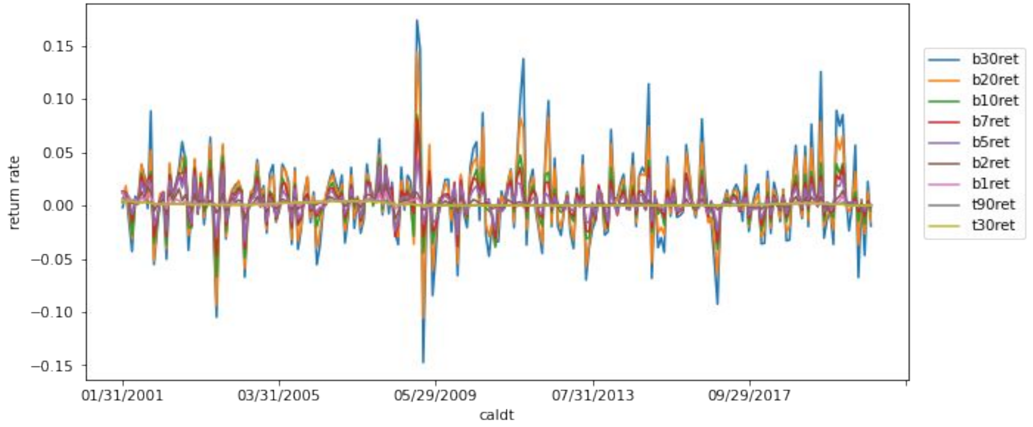
- The data inputs are the previous changes in the yield curve at the following tenors: 30 Year Bond (b30ret), 20 Year Bond(b20ret), 10 Year Bond (b10ret), 7 Year Bond (b7ret), 5 Year Bond (b5ret), 2 Year Bond (b2ret), 1 Year Bond (b1ret), 90 Day Bill (t90ret), 30 Day Bill (t30ret).
- The targets of the model(prediction variables) are three tensors(i.e b1, t90 and t30) of the yield curve. These tenors are most relative to the changes in the yield curve.
- The dataset is downloaded from US Treasury and inflation Indexes in the WRDS.

Data Description

- Our data comes from US Treasury and Inflation Indexes in WRDS. The time interval of data is from January 2001(minimum allowed date) to December 2020(maximum allowed date). When it comes to the query, we choose annual return.
- We found that there is no null values in our dataset. Also, the data are normalized which means we don’t need to standardize or smooth the data.

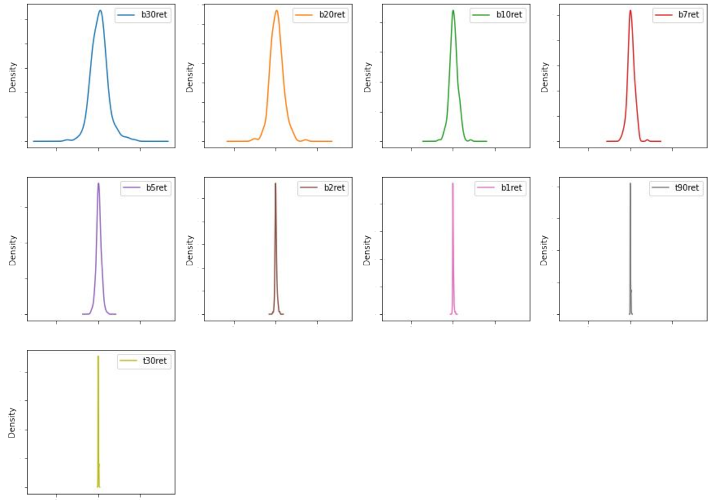
Data Visualization

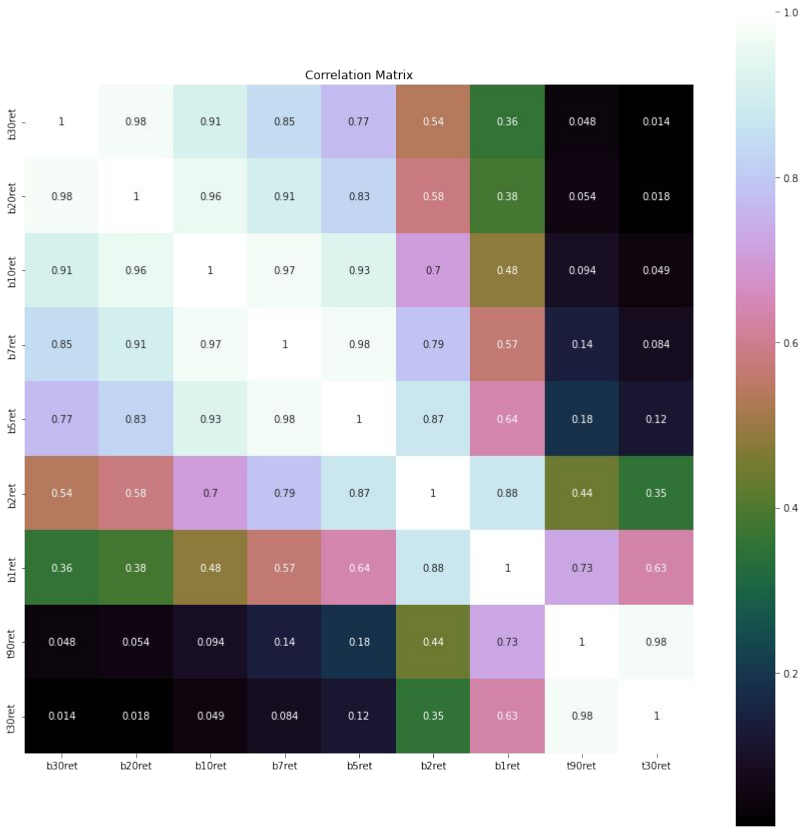
- The graph visualizes the return curves of treasuries with different maturity date vs the timeline.
- We notice that the changes of the return rates of different treasury bonds all follow the similar wave pattern.



Data Visualization

- The left graph shows the density distribution for each treasury and the right one is the correlation matrix of all the treasuries we choose. The lighter the colour of the intersection grid is, the greater the correlation the corresponding two treasuries have.

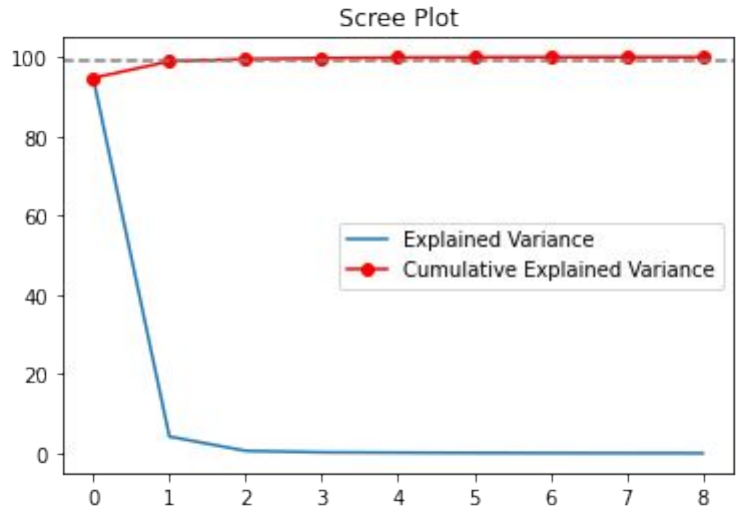




	b30ret	b20ret	b10ret	b7ret	b5ret	b2ret	b1ret	t90ret	t30ret
b30ret	1	0.98	0.91	0.85	0.77	0.54	0.36	0.048	0.014
b20ret	0.98	1	0.96	0.91	0.83	0.58	0.38	0.054	0.018
b10ret	0.91	0.96	1	0.97	0.93	0.7	0.48	0.094	0.049
b7ret	0.85	0.91	0.97	1	0.98	0.79	0.57	0.14	0.084
b5ret	0.77	0.83	0.93	0.98	1	0.87	0.64	0.18	0.12
b2ret	0.54	0.58	0.7	0.79	0.87	1	0.88	0.44	0.35
b1ret	0.36	0.38	0.48	0.57	0.64	0.88	1	0.73	0.63
t90ret	0.048	0.054	0.094	0.14	0.18	0.44	0.73	1	0.98
t30ret	0.014	0.018	0.049	0.084	0.12	0.35	0.63	0.98	1

Feature Selection

- First we compile PCA and generate the scree plot to find the turning point. From the cumulative explained variance line, we notice that 2 principal components should be enough to cover 99% of the dataset's variation.
- The next step is to perform Autoencoder in order to compress the encoding dimension into two. The details will be present in next section.
- After finalising the Autoencoders, the analysis results illustrate that all the input features should be retained. It is reasonable because the dataset is small and only contains 12 features.



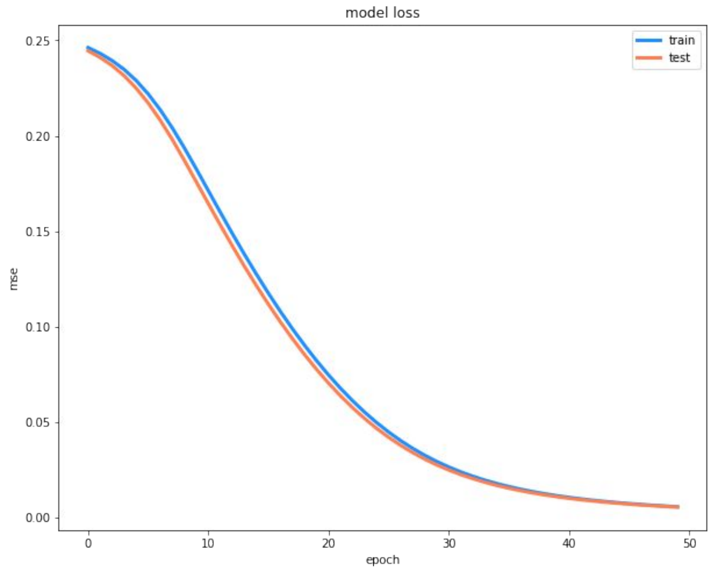
Autoencoder Analysis

- Then we set up the Autoencoder by adding encoded layers, middle layer and decoded layers. After compiling and fitting autoencoder to the data, we find the loss of the model is pretty small.

```
input_img = Input(shape=(12,))
final_activation = 'sigmoid'
encoded = Dense(8, activation='tanh')(input_img)
encoded = Dense(4, activation='relu')(encoded)
encoded = Dense(encoding_dim, activation='tanh')(encoded)  #Middle layer
decoded = Dense(4, activation='tanh')(encoded)
decoded = Dense(8, activation='relu')(decoded)
decoded = Dense(12, activation=final_activation)(decoded)
```

Autoencoder Analysis

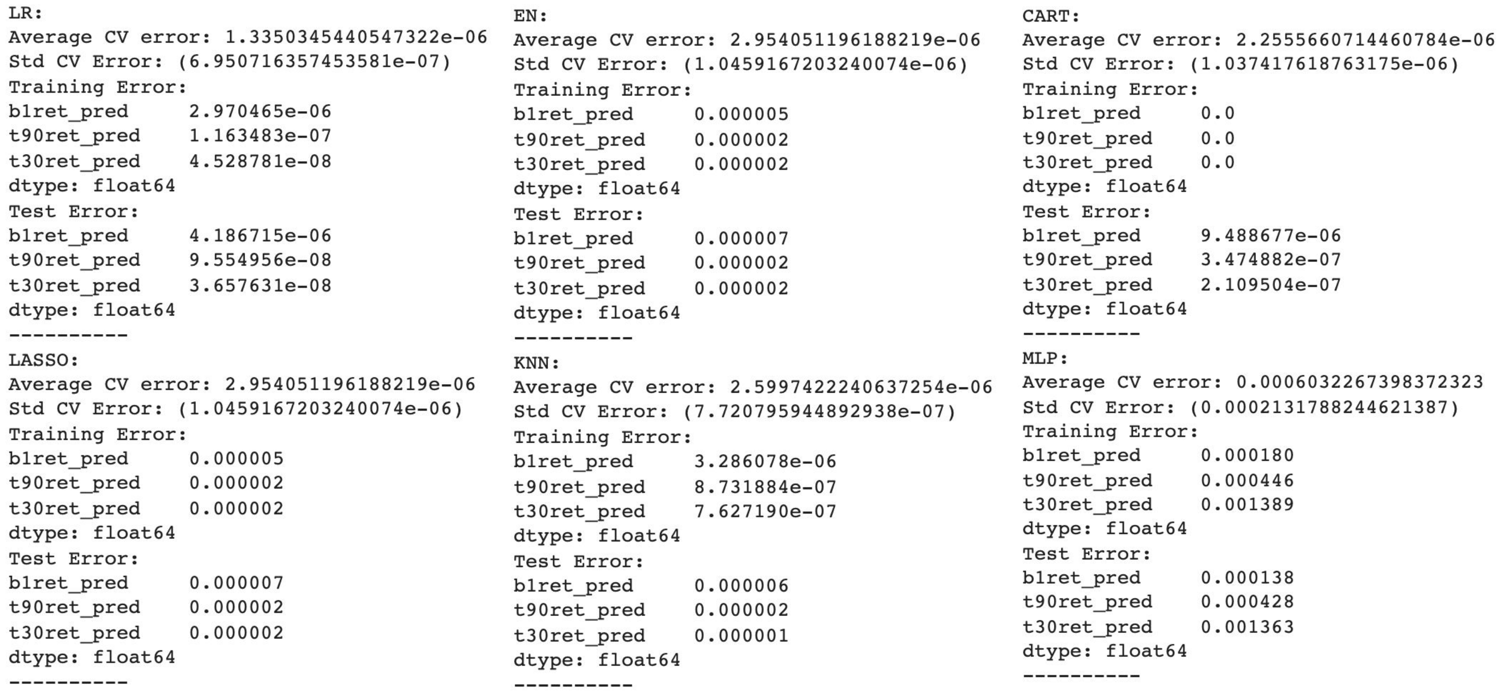
- Both train and test error of Autoencoder are reduced and converged over epochs. By observing the loss plot, we deem that the model is good fit to our dataset.
- However, Autoencoder will be omitted due to its negative R2 score.



```
MSE Knn = 0.000003, MSE AE = 0.000908, MSE OLS = 0.000001, MSE Lasso = 0.000001
R2 Knn = 0.313402,  R2 AE = -225.896027 , R2 OLS = 0.762891, R2 Lasso = 0.762893
```

Models

- Linear Regression
- LASSO
- ElasticNet
- KNN
- CART
- MLP

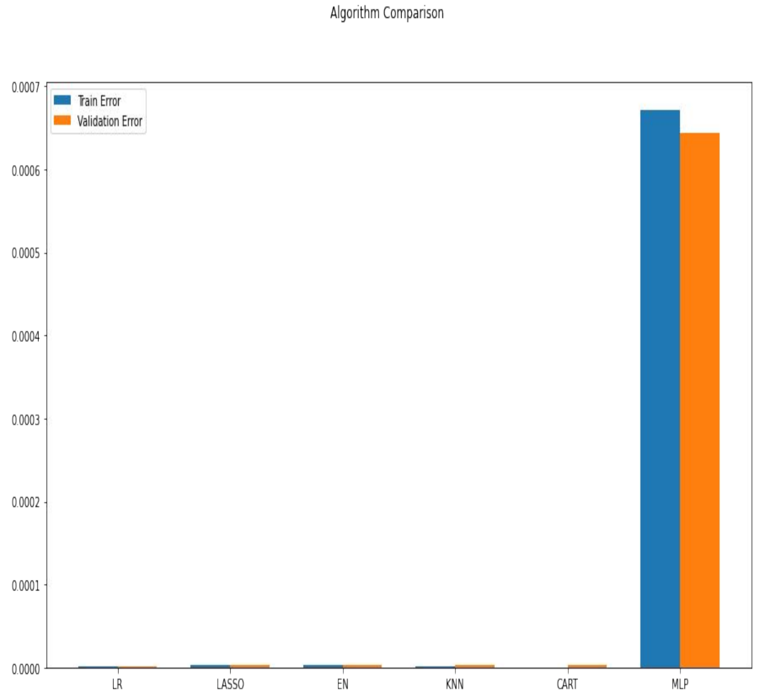


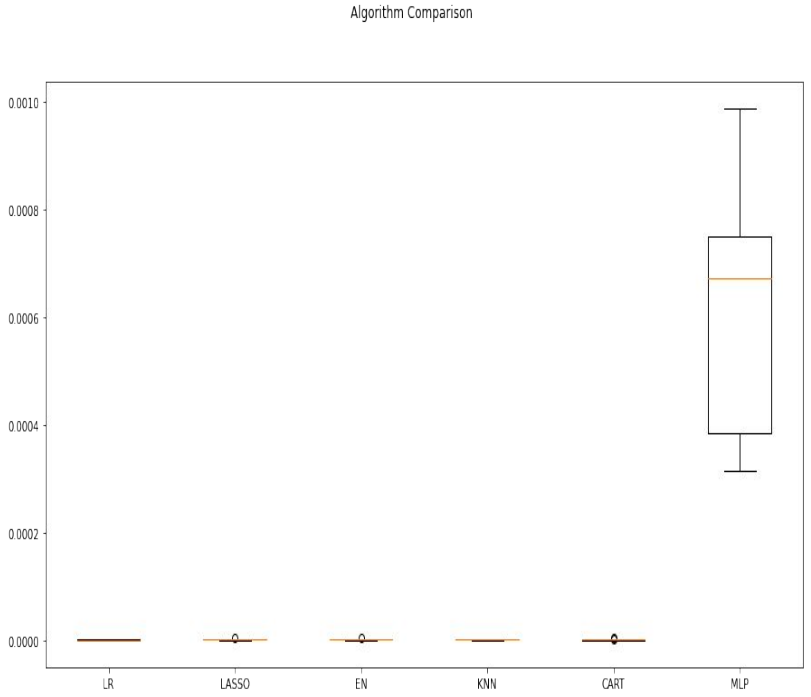
```
LR:
Average CV error: 1.3350345440547322e-06
Std CV Error: (6.950716357453581e-07)
Training Error:
b1ret_pred     2.970465e-06
t90ret_pred    1.163483e-07
t30ret_pred    4.528781e-08
dtype: float64
Test Error:
b1ret_pred     4.186715e-06
t90ret_pred    9.554956e-08
t30ret_pred    3.657631e-08
dtype: float64
----------
LASSO:
Average CV error: 2.954051196188219e-06
Std CV Error: (1.0459167203240074e-06)
Training Error:
b1ret_pred     0.000005
t90ret_pred    0.000002
t30ret_pred    0.000002
dtype: float64
Test Error:
b1ret_pred     0.000007
t90ret_pred    0.000002
t30ret_pred    0.000002
dtype: float64
----------
```

```
EN:
Average CV error: 2.954051196188219e-06
Std CV Error: (1.0459167203240074e-06)
Training Error:
b1ret_pred     0.000005
t90ret_pred    0.000002
t30ret_pred    0.000002
dtype: float64
Test Error:
b1ret_pred     0.000007
t90ret_pred    0.000002
t30ret_pred    0.000002
dtype: float64
----------
KNN:
Average CV error: 2.5997422240637254e-06
Std CV Error: (7.720795944892938e-07)
Training Error:
b1ret_pred     3.286078e-06
t90ret_pred    8.731884e-07
t30ret_pred    7.627190e-07
dtype: float64
Test Error:
b1ret_pred     0.000006
t90ret_pred    0.000002
t30ret_pred    0.000001
dtype: float64
----------
```

```
CART:
Average CV error: 2.2555660714460784e-06
Std CV Error: (1.037417618763175e-06)
Training Error:
b1ret_pred     0.0
t90ret_pred    0.0
t30ret_pred    0.0
dtype: float64
Test Error:
b1ret_pred     9.488677e-06
t90ret_pred    3.474882e-07
t30ret_pred    2.109504e-07
dtype: float64
----------
MLP:
Average CV error: 0.0006032267398372323
Std CV Error: (0.0002131788244621387)
Training Error:
b1ret_pred     0.000180
t90ret_pred    0.000446
t30ret_pred    0.001389
dtype: float64
Test Error:
b1ret_pred     0.000138
t90ret_pred    0.000428
t30ret_pred    0.001363
dtype: float64
----------
```

Model Comparison





- Based on the graphs, we choose Linear Regression, Knn and Lasso as our final models.

Model Tuning and Grid Search

- KNN
  - n_neighbors: np.arange(1,31)
  - weights: uniform, distance
  - metric: euclidean, manhattan
  - Best model: `{'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'distance'}`
- Lasso
  - alpha: np.linspace(0,0.2,21)
  - selection: cyclic, random
  - Best model: `{'alpha': 0.0, 'selection': 'random'}`

Train vs Validation Error

LR:

```
Average CV error: 1.3350345440547322e-06

Std CV Error: (6.950716357453581e-07)

Training Error:

b1ret_pred     2.970465e-06

t90ret_pred    1.163483e-07

t30ret_pred    4.528781e-08

dtype: float64

Test Error:

b1ret_pred     4.186715e-06

t90ret_pred    9.554956e-08

t30ret_pred    3.657631e-08

dtype: float64
```

LASSO:

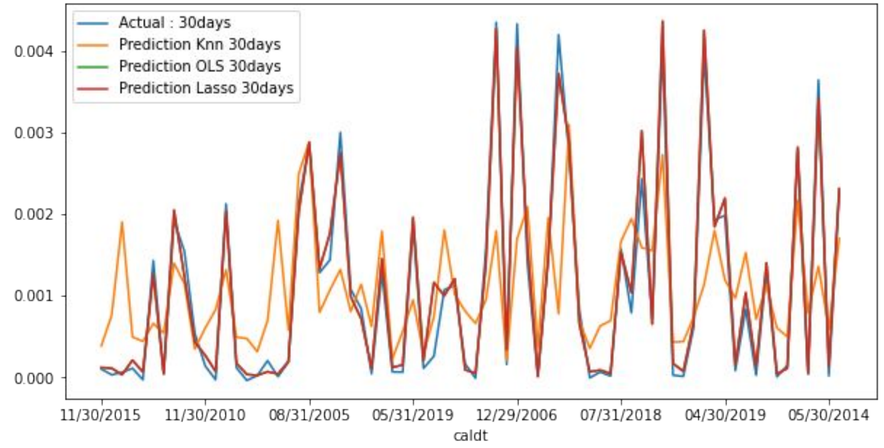
```
Average CV error: 2.954051196188219e-06

Std CV Error: (1.0459167203240074e-06)

Training Error:

b1ret_pred     0.000005

t90ret_pred    0.000002

t30ret_pred    0.000002

dtype: float64

Test Error:

b1ret_pred     0.000007

t90ret_pred    0.000002

t30ret_pred    0.000002

dtype: float64
```

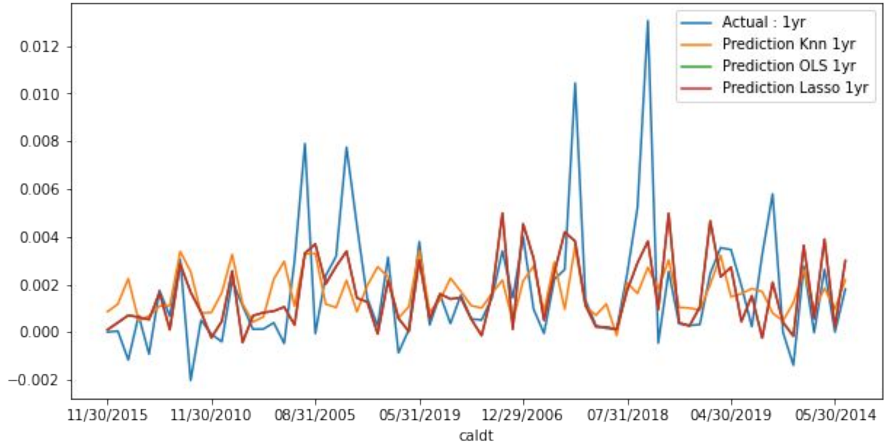
KNN:

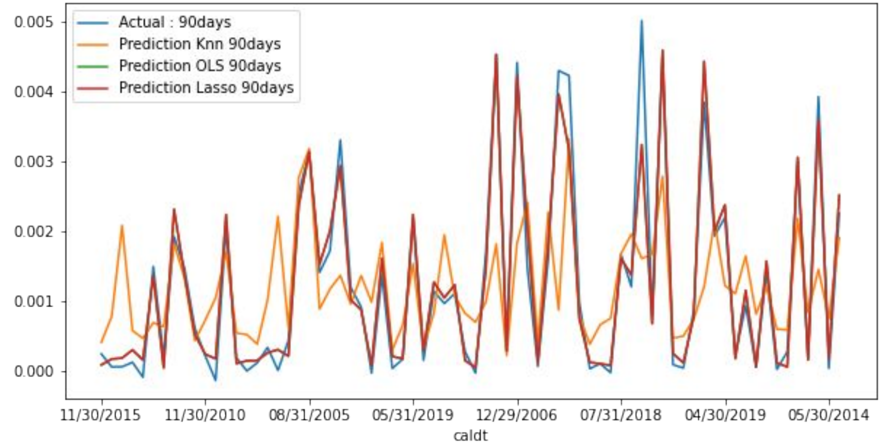
```
Average CV error: 2.5997422240637254e-06

Std CV Error: (7.720795944892938e-07)

Training Error:

b1ret_pred     3.286078e-06

t90ret_pred    8.731884e-07

t30ret_pred    7.627190e-07

dtype: float64

Test Error:

b1ret_pred     0.000006

t90ret_pred    0.000002

t30ret_pred    0.000001

dtype: float64
```

Multi-target Plots

- Overall, the Lasso and Linear Regression are comparable. The fitting with KNN is slightly poor as compared to the Lasso and Linear Regression. Also, the multitask learning with neural network is more intuitive for modeling many time series simultaneously.

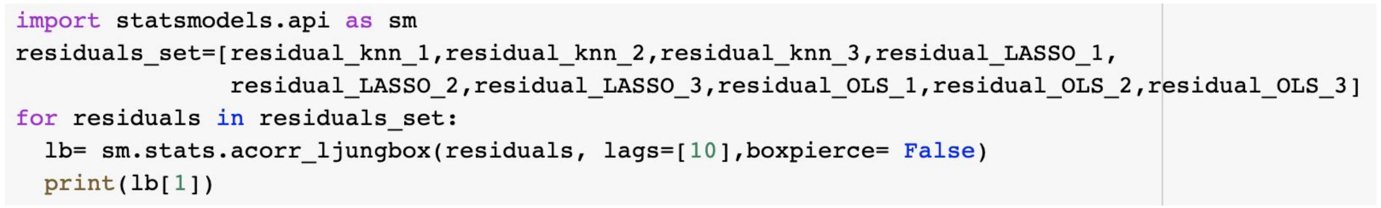




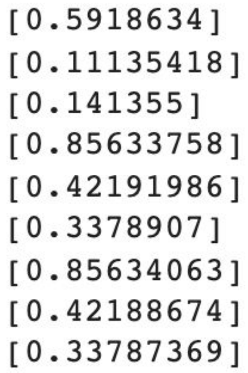


Train and Test Metrics - Statistical

- We use residual autoencoder to check if the residuals are independent. We find that the p_value of each set is greater than the required significance level(>0.05), which indicates that residuals are independent.



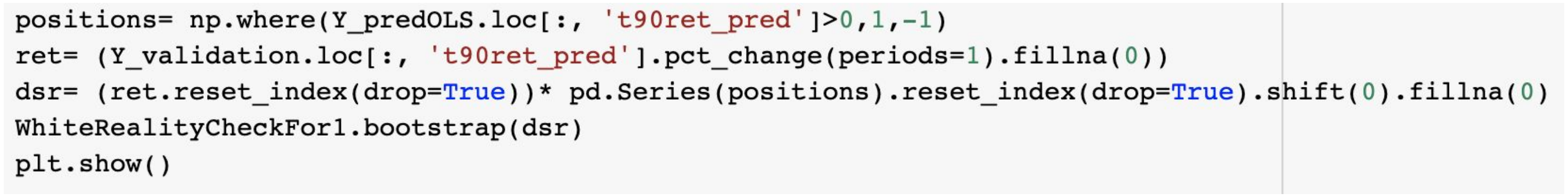
```
import statsmodels.api as sm
residuals_set=[residual_knn_1,residual_knn_2,residual_knn_3,residual_LASSO_1,
               residual_LASSO_2,residual_LASSO_3,residual_OLS_1,residual_OLS_2,residual_OLS_3]
for residuals in residuals_set:
  lb= sm.stats.acorr_ljungbox(residuals, lags=[10],boxpierce= False)
  print(lb[1])
```



```
[0.5918634]
[0.11135418]
[0.141355]
[0.85633758]
[0.42191986]
[0.3378907]
[0.85634063]
[0.42188674]
[0.33787369]
```

White Reality Check

- We are dealing with multi-target(b1, t90 and t30), therefore we use White Reality Check for each one of them.



```
positions= np.where(Y_predOLS.loc[:, 't90ret_pred']>0,1,-1)
ret= (Y_validation.loc[:, 't90ret_pred'].pct_change(periods=1).fillna(0))
dsr= (ret.reset_index(drop=True))* pd.Series(positions).reset_index(drop=True).shift(0).fillna(0)
WhiteRealityCheckFor1.bootstrap(dsr)
plt.show()
```

Sample Code

White Reality Check

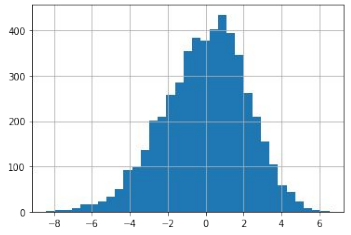
- All p-values are greater than 0.05. Thus we do not reject Ho. The population distribution of rule returns has an expected value of zero or less.

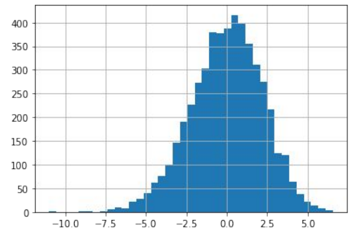
LR

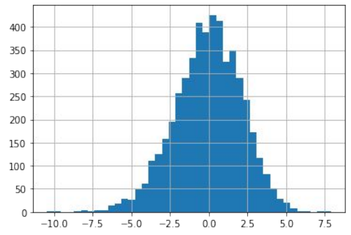
Knn

Lasso

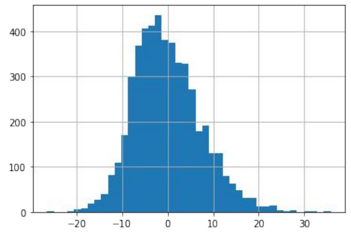
t90ret_pred

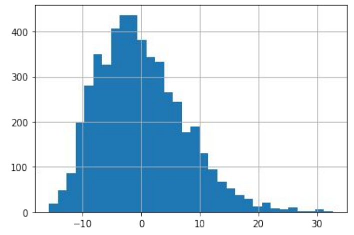

P_value: 0.5854

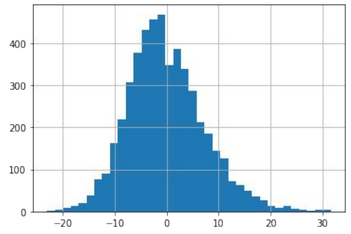

P_value: 0.5762


P_value: 0.5764

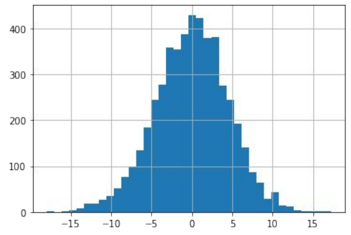
b1ret_pred

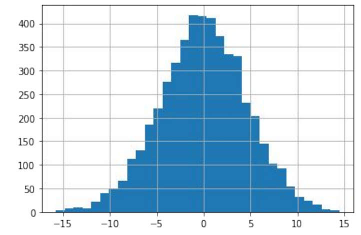

P_value: 0.4346

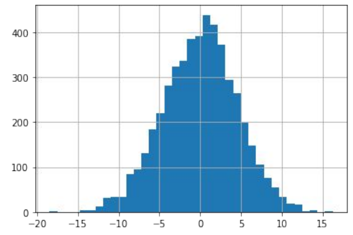

P_value: 0.1642


P_value: 0.4306

t30ret_pred


P_value: 0.1036


P_value: 0.1148


P_value: 0.1104

Conclusion

- The case study 2 of Chapter 7 used PCA to reconstruct the yield curve dataset.
- The case study 4 of Chapter 5 used supervised learning-based models to predict the yield curve.
- In order to find the best yield curve prediction model, we combine both cases and make some improvements on the dimensionality reduction and model selection perspectives. We find Lasso and Linear Regression model work relatively better among all models we analyzed.