In [4]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd, numpy as np
```

CQF Exam 3 - 2021/5/23

delegate: Lu, Zhenyang, zzhenyanglu@gmail.com (mailto:zzhenyanglu@gmail.com), +1-312-804-2532

Section 0 - data preprocessing

Group A Data Source:

Hong Kong Heng Seng Index (HSI) daily price(open, high, low, close) from 1999/10/25 to 2021/05/21

Source: Proprietary database

for Group A we create the following features:

1. sign: sign of daily return
2. return_t: T-t return, where t = 1,2,3,4,5
3. momentum_t: price change period t days, where t= 1,2,3,4,5
4. ewma: from https://en.wikipedia.org/wiki/EWMA_chart (https://en.wikipedia.org/wiki/EWMA_chart), $ewma_t = P_t + \lambda * ewma_{t-1}$, where $\lambda = 0.8$
5. O-C: Open - Close
6. H-L: High - Low
7. ewma-price: sing of ewma-price

dependent variable (sign_1 in the actual data below): T+1 daily return's sign, 1 if postive return, 0 otherwise

below is quick peek at group A data

In [5]:

```python
# group_a
group_a = pd.read_csv(r'./HSI.csv')
shift_days = 5
_lambda = 0.4 #EWMA model's argument

for i in range(1,1+shift_days):
    group_a[f'return_{i}'] = np.log(group_a['close']/group_a['close'].shift(i))
    group_a[f'momentum_{i}'] = group_a['close'] - group_a['close'].shift(i)
    group_a[f'sign_{i}'] = group_a[f'return_{i}'] >= 0
    #group_a[f'sign_{i}'] = group_a[f'sign_{i}'].astype(int)
    group_a[f'sign_{i}'] = group_a[f'sign_{i}'].apply(lambda x:1 if x else 0)

    group_a[f'return_{i}'] = group_a[f'return_{i}'].shift(1)
    group_a[f'momentum_{i}'] = group_a[f'momentum_{i}'].shift(1)
    group_a[f'sign_{i}'] = group_a[f'sign_{i}'].shift(1)

group_a['O-C'] = group_a['open'] - group_a['close']
group_a['H-L'] = group_a['high'] - group_a['low']

# EWMA
close_list = group_a['close'].tolist()
ewma_list = [close_list[0]]
for i in range(1, group_a['close'].shape[0]):
    ewma_list.append(_lambda*group_a['close'].iloc[i] + (1-_lambda)*ewma_list[i-1])

group_a['ewma'] = ewma_list
group_a['ewma-price'] = (group_a['ewma'] - group_a['close']) >= 0
group_a['ewma-price'] = group_a['ewma-price'].astype(int)
group_a = group_a.iloc[shift_days+2:]
group_a.head(10)
```

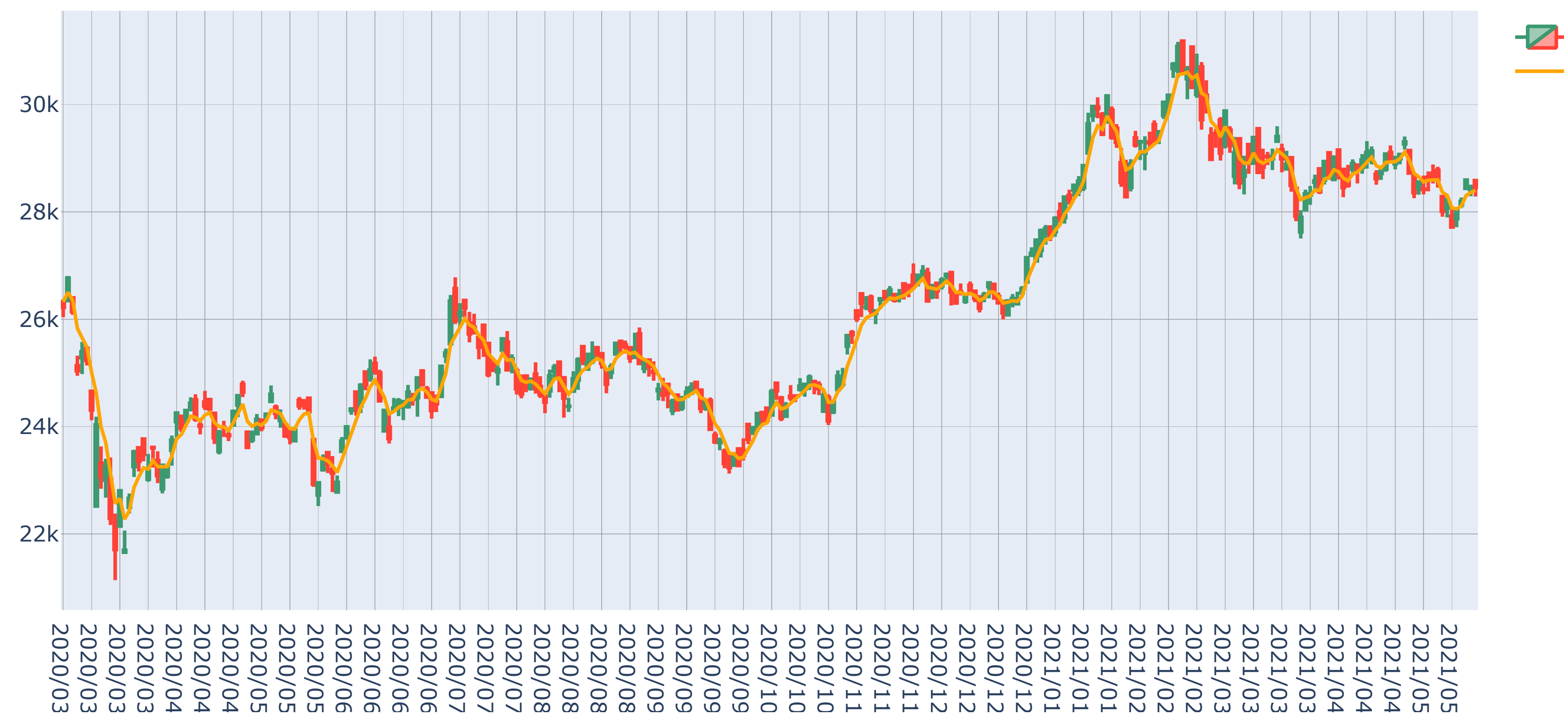
Out[5]:

	date	open	high	low	close	return_1	momentum_1	sign_1	return_2	momentum_2	...	return_4	momentum_4	sign
7	1999/11/03	13366.12	13497.36	13230.72	13257.33	0.001030	13.73	1.0	0.005933	78.89	...	0.048139	626.77	[illegible]
8	1999/11/04	13272.30	13693.39	13272.30	13651.51	-0.005905	-78.51	0.0	-0.004874	-64.78	...	0.038323	498.45	[illegible]
9	1999/11/05	13698.30	13730.89	13533.26	13610.27	0.029300	394.18	1.0	0.023395	315.67	...	0.029328	394.56	[illegible]
10	1999/11/08	13659.26	13756.84	13513.66	13521.11	-0.003025	-41.24	0.0	0.026274	352.94	...	0.021400	288.16	[illegible]
11	1999/11/09	13585.65	13735.07	13574.42	13669.70	-0.006572	-89.16	0.0	-0.009598	-130.40	...	0.013797	185.27	[illegible]
12	1999/11/10	13734.41	14005.26	13734.41	13975.54	0.010930	148.59	1.0	0.004357	59.43	...	0.030631	412.37	[illegible]
13	1999/11/11	14053.47	14217.36	13916.14	14105.71	0.022127	305.84	1.0	0.033056	454.43	...	0.023459	324.03	[illegible]
14	1999/11/12	14110.83	14268.39	13977.95	14189.67	0.009271	130.17	1.0	0.031398	436.01	...	0.035755	495.44	[illegible]
15	1999/11/15	14251.29	14634.99	14251.29	14562.22	0.005935	83.96	1.0	0.015206	214.13	...	0.048262	668.56	[illegible]
16	1999/11/16	14619.42	14726.65	14491.39	14689.46	0.025916	372.55	1.0	0.031851	456.51	...	0.063249	892.52	[illegible]

10 rows × 24 columns

GROUP A: below is a candlestick plot with EWMA average line of the most recent 500 trading days

In [6]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

fig = go.Figure(data=[go.Candlestick(x=group_a['date'][-300:],
                open=group_a['open'][-300:],
                high=group_a['high'][-300:],
                low=group_a['low'][-300:],
                close=group_a['close'][-300:], name = 'HSI'),
                go.Scatter(x=group_a['date'][-300:], y=group_a['ewma'][-300:], line=dict(color='orange', width=2), r
               )

fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```



Group B:

St Louis Fed Relative Midwest Economy Index from 1990-01-01 to 2021-03-01

Source: Fred St Louis. [https://fred.stlouisfed.org/series/RMEIM683SFRBCHI (https://fred.stlouisfed.org/series/RMEIM683SFRBCHI)](https://fred.stlouisfed.org/series/RMEIM683SFRBCHI)

for Group B we create the following features:

1. sign: sign of monthly return
2. return_t: T-t return, where t=1,2,3,4,5,6
3. momentum_t: price change period t months, where t=1,2,3,4,5,6
4. ma_t: t month simple moving average, where t=1,2,3,4,5,6

dependent variable (return_1 in the actual data below): T+1 monthly return, with 10 bin bucketing (i.e. labels to be predicted is 0 thru 9) based on quantile

In [7]:

```python
group_b = pd.read_csv(r'./RMEIM683SFRBCHI.csv')
shift_days = 6

for i in range(1,1+shift_days):
    group_b[f'return_{i}'] = (group_b['value']/group_b['value'].shift(i)) -1
    group_b[f'momentum_{i}'] = group_b['value'] - group_b['value'].shift(i)
    group_b[f'sign_{i}'] = group_b[f'return_{i}'] >= 0
    group_b[f'sign_{i}'] = group_b[f'sign_{i}'].apply(lambda x:1 if x else -1)
    #group_b[f'sign_{i}'] = group_b[f'sign_{i}'].astype(int)
    group_b[f'return_{i}']

# MA(5)
for i in range(1,1+shift_days):
    group_b[f'ma_{i}'] = group_b['value'].rolling(i).mean()

group_b = group_b.iloc[shift_days:]
group_b.tail(10)
```

Out[7]:

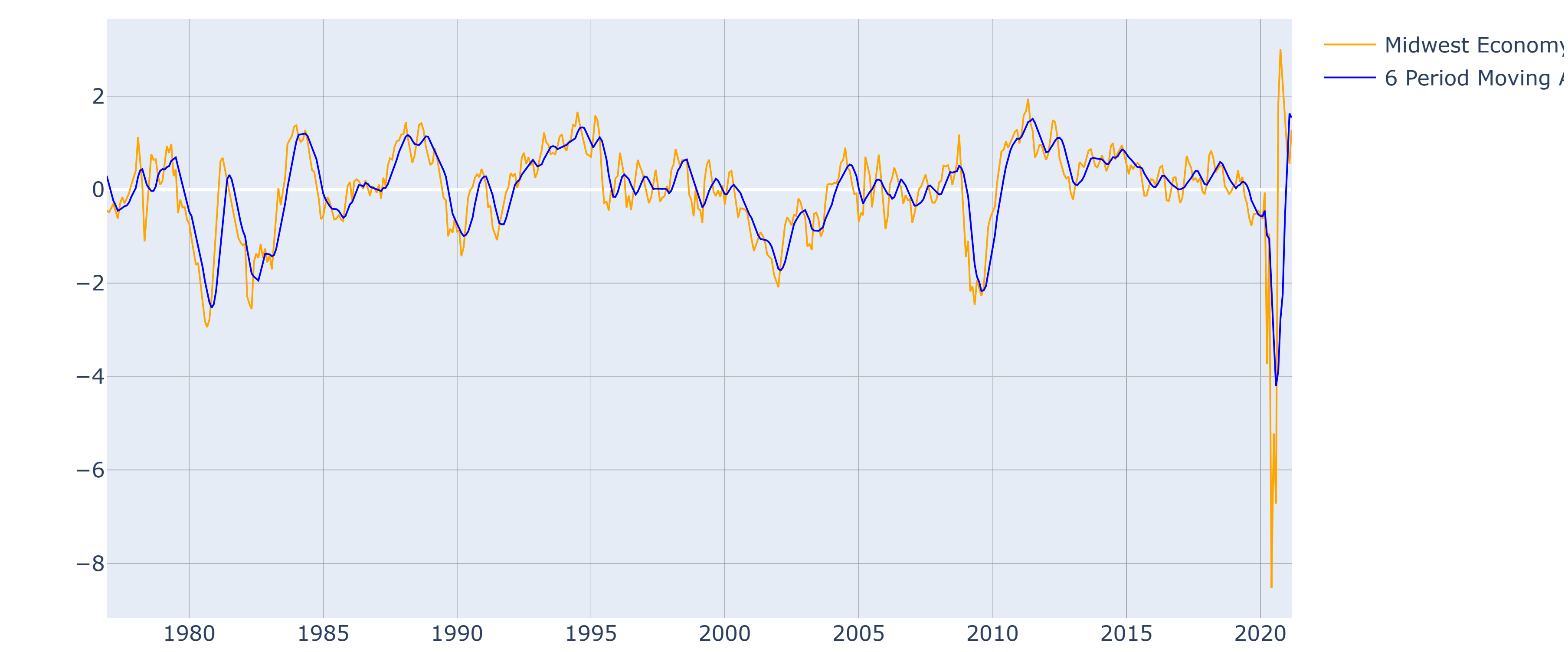
	date	value	return_1	momentum_1	sign_1	return_2	momentum_2	sign_2	return_3	momentum_3	...	sign_5	return_6	momentu
528	2020-06-01	-8.52920	8.059161	-7.58770	1	1.287759	-4.80101	1	137.934680	-8.46781	...	1	17.967687	-8.0
529	2020-07-01	-5.22077	-0.387895	3.30843	-1	4.545162	-4.27927	1	0.400350	-1.49258	...	1	9.255102	-4.7
530	2020-08-01	-6.71652	0.286500	-1.49575	1	-0.212526	1.81268	-1	6.133850	-5.77502	...	1	9.647453	-6.0
531	2020-09-01	1.82300	-1.271420	8.53952	-1	-1.349182	7.04377	-1	-1.213736	10.35220	...	-1	-30.695390	1.8
532	2020-10-01	3.00375	0.647696	1.18075	1	-1.447218	9.72027	-1	-1.575346	8.22452	...	-1	-1.805686	6.7
533	2020-11-01	2.20800	-0.264919	-0.79575	-1	0.211190	0.38500	1	-1.328742	8.92452	...	-1	-3.345194	3.1
534	2020-12-01	1.38178	-0.374194	-0.82622	-1	-0.539982	-1.62197	-1	-0.242030	-0.44122	...	-1	-1.162006	9.9
535	2021-01-01	0.80070	-0.420530	-0.58108	-1	-0.637364	-1.40730	-1	-0.733433	-2.20305	...	-1	-1.153368	6.0
536	2021-02-01	0.55475	-0.307169	-0.24595	-1	-0.598525	-0.82703	-1	-0.748755	-1.65325	...	-1	-1.082595	7.2
537	2021-03-01	1.28082	1.308824	0.72607	1	0.599625	0.48012	1	-0.073065	-0.10096	...	-1	-0.297411	-0.5

10 rows × 26 columns

GROUP B - a quick plot of Midwest Economy Index and its 6 periods Moving Average line

In [8]:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

go.Figure(data=[
    go.Scatter(x=group_b['date'], y=group_b['value'], line=dict(color='orange', width=1), name='Midwest Economy Inde
    go.Scatter(x=group_b['date'], y=group_b['ma_6'], line=dict(color='blue', width=1), name='6 Period Moving Average
]).show()
```

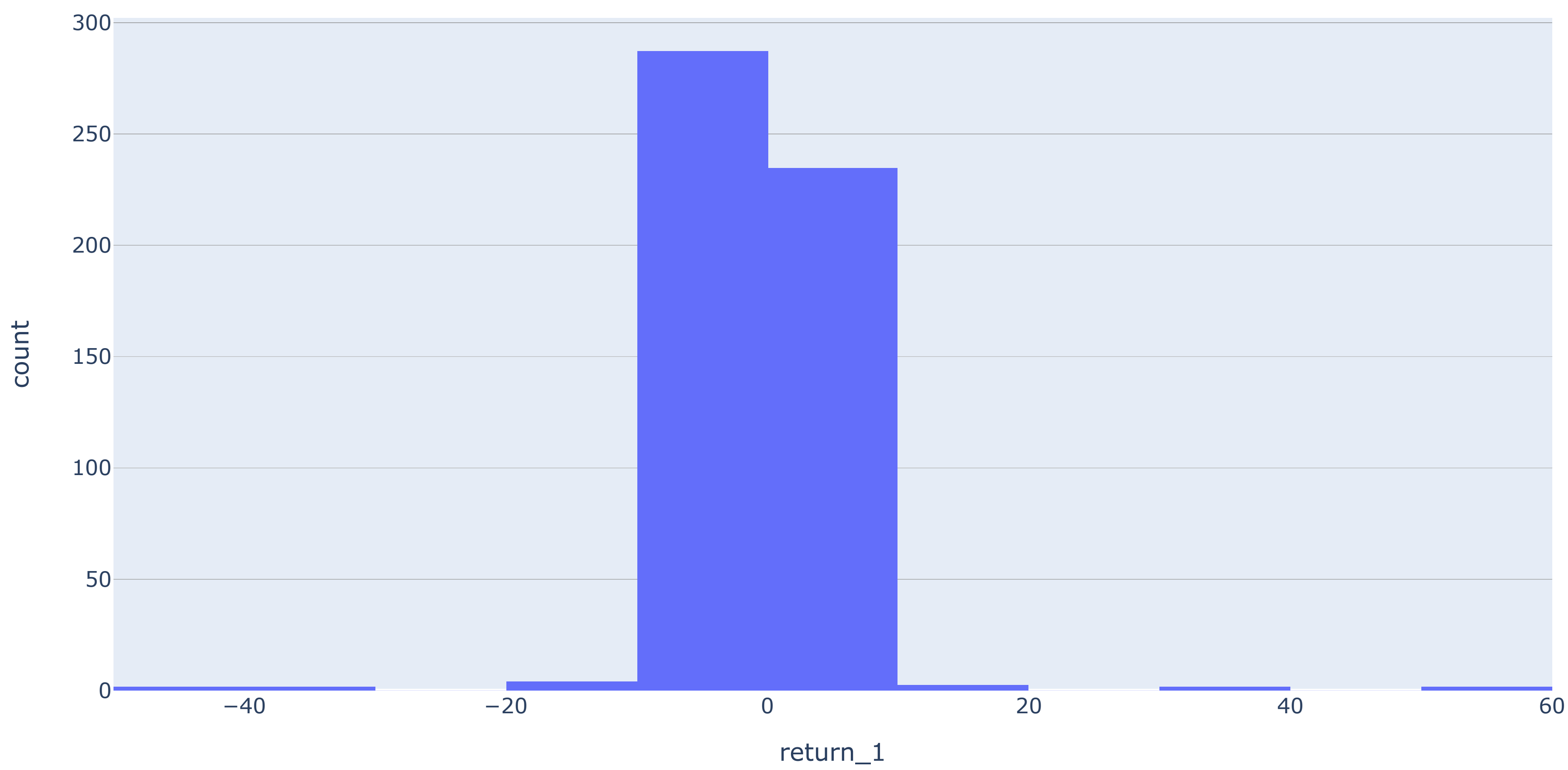


GROUP B - Histogram of 1 Period Return

reminder we are gonna have 10 bins based on quantile

In [9]:

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(group_b, x="return_1",nbins=20)
fig.show()
```



Section 1 - Classifier and Hyperparameters

1.1 Briefly present the maths of logistic classifier, use β for coefficients not θ, by writing down:

1.1.(a) the complete Loss Function (cost function) and relate it to MLE total log-likelihood function;

(Only briefly list down the math formula without a step by step induction)

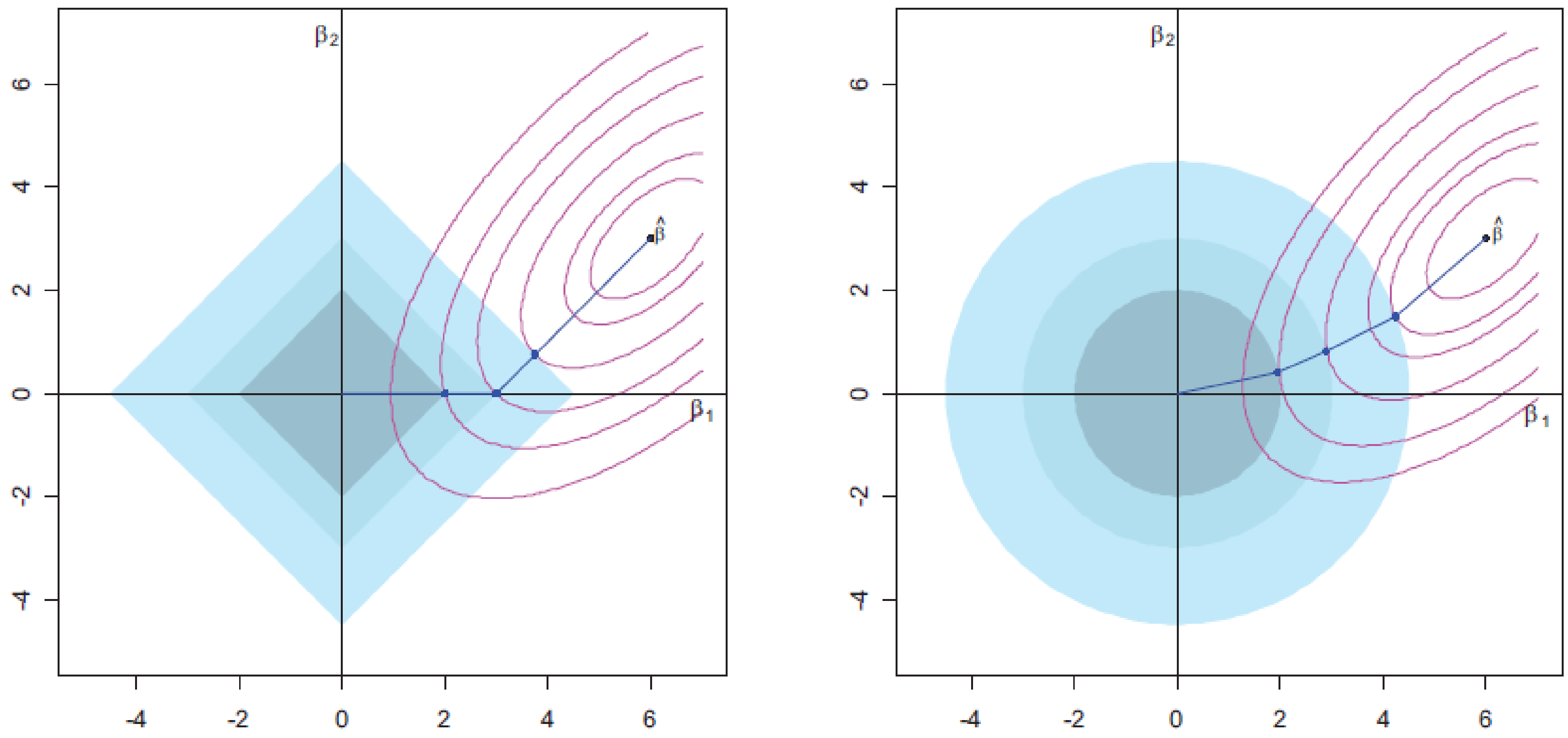
$J(\beta) = \frac{1}{m}\sum_i^M y^i log(h_\beta(x^i)) + (1-y^i)log(1-h_\beta(x^i))$, where $h_\beta(x) = \frac{1}{1+e^{-(\beta \cdot X)}}$,

so Log likeliness MLE $lik(\beta) = \prod f(x|\beta) = \sum_i^M y^i log(h_\beta(x^i)) + (1-y^i)log(1-h_\beta(x^i))$

1.1.(b) the penalty expressions for Lasso and Ridge regressions (mathematical and graphical);

$J_{LASSO}(\beta) = J(\beta) + \lambda||\beta|| = J(\beta) + \lambda\sum^k|\beta_k|$

$J_{RIDGE}(\beta) = J(\beta) + \lambda||\beta||^2 = J(\beta) + \lambda\sum^k|\beta_k|^2$



1.1.(c) RSS and MSE expressions, specific to logistic regression.

$RSS = \sum_i^M (h_\beta(x^i) - y^i)^2$

$MSE = \frac{1}{n}\sum_i^N (h_\beta(x^i) - y^i)^2$

In [ ]:
```
1
```

1.2. Implement Ridge and Lasso logistic regressions using the suitable full set of features in each prediction task Group A, Group B

1.2.(a) produce a table comparing L1 and L2 type of penalisation: the impact made on regression coefficients;

Group A: HSI data

for group A data, setting daily return's sign(sign_1 column in group_a from the code below) (0 if return is negative and 1 otherwise) of daily return of dependent variable and the rest as independent variable, below is python code to do regression with L1 and L2 penlization and table comparing L1 and L2 penalization.

In conclusion, it looks like from the table and graph (each dot on the line represents a coefficient) below L2 boost L1's coefficients.

In [14]:

```
from sklearn.linear_model import LogisticRegression

y = group_a['sign_1'] # sign of daily return
X = group_a[set(group_a.columns) - set(['date','sign_1'])]
assert 'sign_1' not in X.columns
# L1 model
l1_model = LogisticRegression(penalty='l1', solver='saga')
l1_model.fit(X, y)
l1_prediction = l1_model.predict(X) # predicting

# L2 model
l2_model = LogisticRegression(penalty='l2')
l2_model.fit(X, y)
l2_prediction = l2_model.predict(X) # predicting

l1_model_coefs = pd.DataFrame(l1_model.coef_, columns = X.columns)
l2_model_coefs = pd.DataFrame(l2_model.coef_, columns = X.columns)
table = pd.concat([l1_model_coefs, l2_model_coefs])
table.index=['L1','L2']
table
```

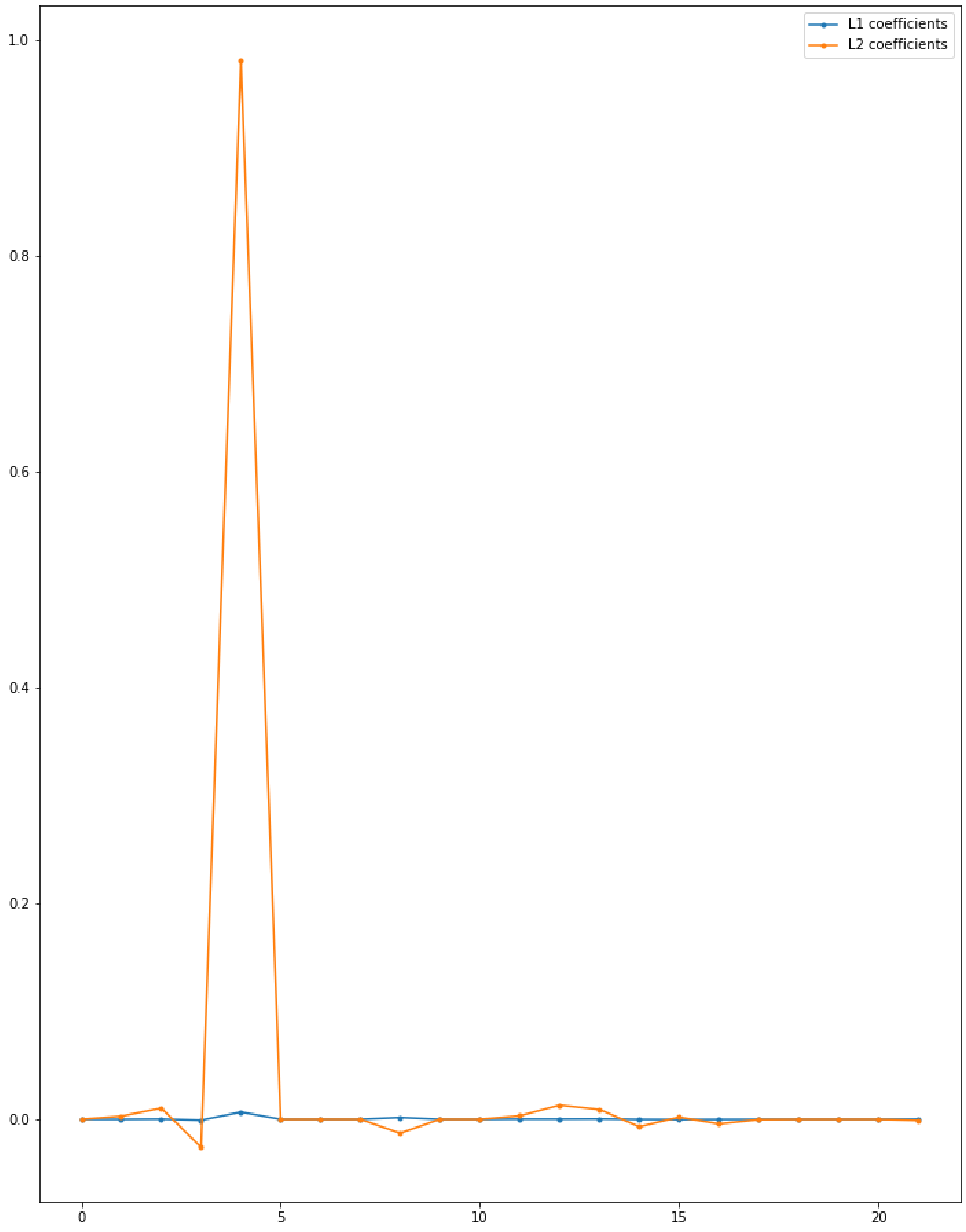
Out[14]:

	return_4	O-C	close	ewma	momentum_1	sign_5	return_1	sign_4	momentum_2	sign_3	...	open	momentum_3
L1	0.000000	0.000013	0.000222	-0.000826	0.006719	1.001688e-07	3.228137e-07	6.636860e-07	0.001691	0.000001	...	0.000235	0.000368
L2	0.000004	0.002875	0.010372	-0.025796	0.981807	-3.660943e-05	5.167622e-05	6.667411e-06	-0.012800	0.000052	...	0.013247	0.009137

2 rows × 22 columns

In [15]:

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12,16))
ax = plt.axes()
ax.plot(l1_model.coef_[0], marker ='.')
ax.plot(l2_model.coef_[0], marker ='.')
ax.legend(['L1 coefficients', 'L2 coefficients'])
fig.show()
```



Group B: Chicago Fed Relative Midwest Economy Index

before training a model, I'd first bucket the target variable with 10 bins based on quantile of the periodic(monthly) return of data. Then run L1 and L2 logistic regression to predict the bucketed periodic return, table below this cell shows a comparison between L1 and L2 coeffcients (index of table show if the row is L1 or L2), L1 and L2 model each has 10 rows, each of which shows coefficients to predict a specific bucket level of the periodic return.

In [16]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import KBinsDiscretizer

####### bucketing dependent variable ########
y = group_b['return_1'] # sign of
X = group_b[set(group_b.columns) - set(['date','return_1'])]

est = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='quantile')
y_bucket = est.fit_transform([[i] for i in y.to_list()]) # arrange it as a 2-D array
y_bucket = [i[0] for i in y_bucket]  # flats it back to 1D
####### bucketing dependent variable ########

# L1 model
l1_model = LogisticRegression(multi_class='multinomial',solver='saga', penalty='l1')
l1_model.fit(X, y_bucket)
l1_model_coefs = pd.DataFrame(l1_model.coef_, columns = X.columns)
l1_prediction = l1_model.predict(X) # predicting
l1_model_coefs.index = ['L1']*len(l1_model_coefs)

# L2 model
l2_model = LogisticRegression(multi_class='multinomial',solver='lbfgs', penalty='l2')
l2_model.fit(X, y_bucket)
l2_model_coefs = pd.DataFrame(l2_model.coef_, columns = X.columns)
l2_model_coefs.index = ['L2']*len(l2_model_coefs)
l2_prediction = l2_model.predict(X) # predicting

table = pd.concat([l1_model_coefs, l2_model_coefs])
table
```

Out[16]:

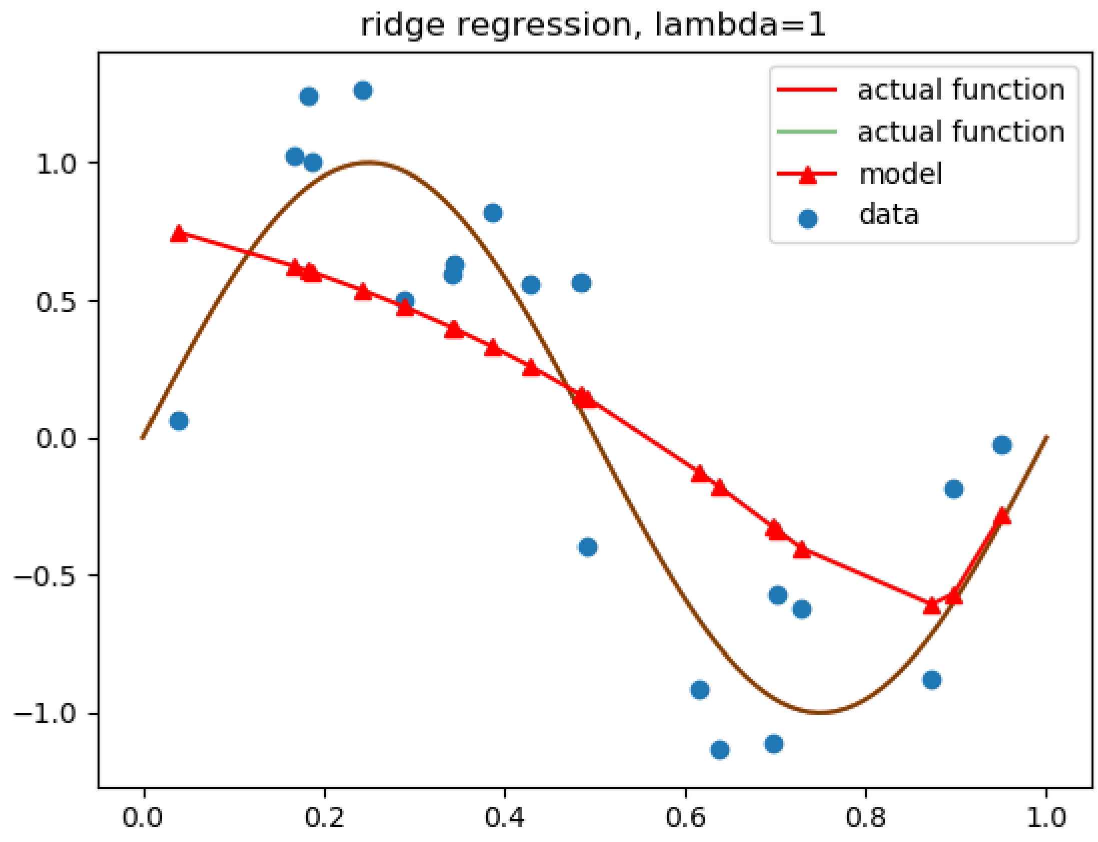
	ma_4	value	ma_2	return_4	sign_1	ma_5	momentum_1	sign_5	sign_4	momentum_2	...	momentum_3	momentun
L1	0.001745	-0.011476	-0.001104	-0.031650	-0.208707	0.002427	-0.015166	-0.052909	-0.023051	-0.029793	...	-0.035706	-0.022
L1	-0.014424	0.003330	-0.011308	0.012162	-0.201096	-0.008873	0.045649	-0.102978	-0.081923	0.015801	...	0.037196	-0.000
L1	-0.000710	-0.008362	-0.018033	0.031490	-0.209872	-0.003902	0.013985	-0.041588	-0.050190	-0.019554	...	-0.020068	0.003
L1	0.007702	0.004366	0.013293	0.000384	-0.216683	0.004486	-0.012278	0.006485	-0.016479	-0.014940	...	0.019475	0.007
L1	0.019868	0.012840	0.016119	-0.027327	-0.230787	0.017501	-0.001061	0.034434	0.003097	-0.012717	...	0.000000	0.000
L1	0.000000	0.000381	0.000006	0.000381	-0.001013	0.000097	0.000000	0.062873	0.060278	0.000806	...	0.002873	-0.015
L1	-0.020249	-0.015550	-0.017196	-0.010458	0.265452	-0.021952	0.000000	0.069701	0.056249	-0.000142	...	0.014842	0.008
L1	0.004594	0.004509	0.003623	0.006815	0.254427	0.005948	0.000000	0.059651	0.084326	0.017475	...	-0.019797	-0.001
L1	0.000000	0.008490	0.013625	0.036798	0.265885	-0.002221	-0.004977	-0.004659	0.018380	0.043379	...	0.014577	0.043
L1	0.000000	-0.007892	0.000000	-0.023775	0.284843	0.000000	-0.017751	-0.034147	-0.050284	0.000574	...	-0.015026	-0.029
L2	0.077699	-0.125609	0.013171	-0.039504	-2.072282	-0.023634	-0.277562	-0.158103	0.146634	-0.178627	...	-0.357046	0.303
L2	0.032220	0.147961	-0.153822	0.043905	-1.536727	0.084102	0.603565	-0.536774	-0.130749	-0.248710	...	0.108109	-0.143
L2	0.061495	0.008949	-0.218703	0.030546	-1.818749	0.067040	0.455305	0.242778	0.035671	-0.232516	...	-0.432973	-0.080
L2	-0.056827	-0.045722	0.090191	-0.011895	-1.884748	0.031800	-0.271826	0.159532	-0.007216	-0.251420	...	0.567668	-0.432
L2	0.067691	0.023612	-0.013568	-0.041750	-2.057006	0.015705	0.074360	-0.103710	-0.013803	-0.195486	...	-0.055191	0.215
L2	-0.071845	0.028571	0.168442	-0.009101	0.017591	0.003061	-0.279742	0.043012	-0.019694	0.349052	...	0.332352	-0.274
L2	-0.071472	-0.036979	-0.080523	-0.014068	2.339189	-0.077610	0.087088	0.316466	-0.052581	-0.466922	...	0.517808	0.065
L2	0.061315	0.049942	0.005899	0.002455	2.213265	0.054591	0.088086	0.013855	0.028205	0.672703	...	-0.806283	0.022
L2	0.016829	-0.003814	0.084988	0.056064	2.330196	-0.122895	-0.177604	-0.076858	-0.108366	0.079628	...	0.015404	0.677
L2	-0.117105	-0.046910	0.103925	-0.016652	2.469269	-0.032160	-0.301671	0.099804	0.121899	0.472299	...	0.110153	-0.354

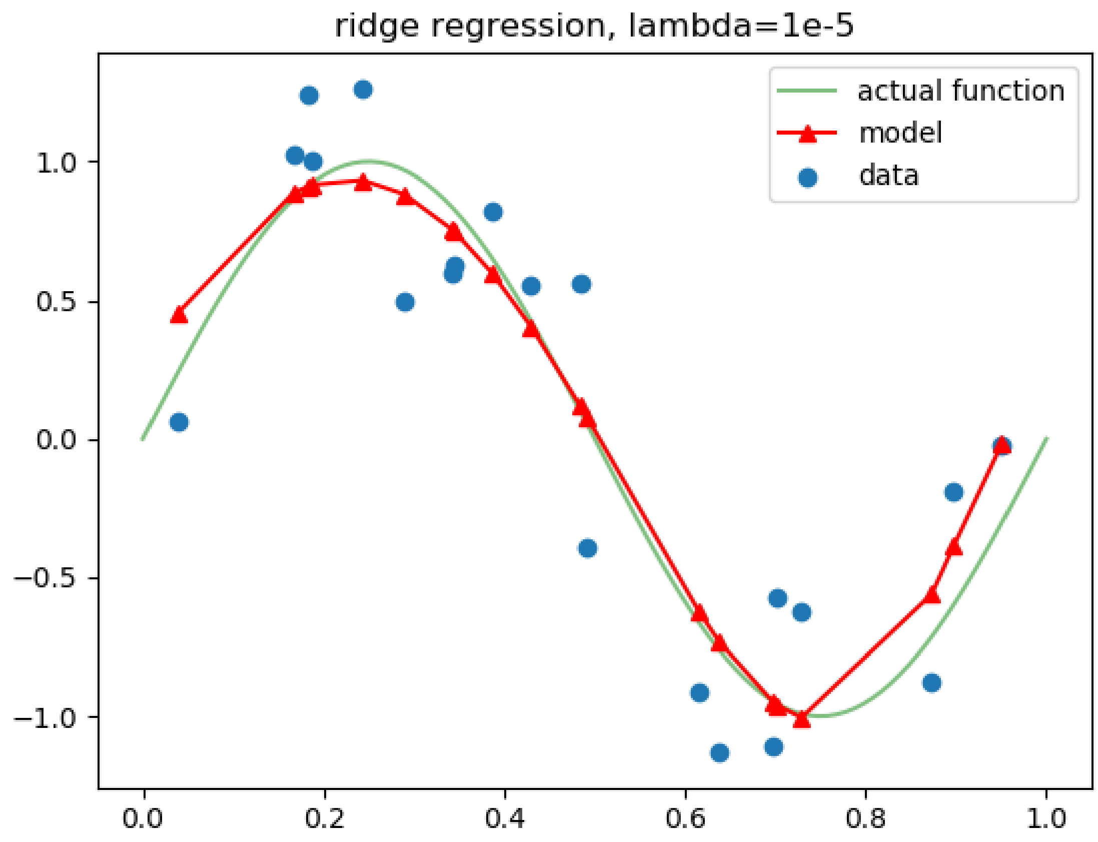
20 rows × 24 columns

1.2.(b) clearly explain whether L1 or L2 regression likely to have a high bias and low variance.

looks like model from group a has an accuracy score of 99.8% _, and model grom group b has an accurary score of 10% (almost like randomly guess from a set of 10 targets, since it has 10 buckets), no matter what regularizarion factor I use, apparently both need to be tweaked (preprocessing, scaling, things like that) to show correctly the L1/L2's effect on bias/variance. So for now I'm going to borrow pictures from the internet to answer this question:

It is likely if we use an improper λ value of lasso/ridge regression. as the following images show (high λ versus low λ), apparently 1 is a overfit and the other being a underfit by tweaking λ





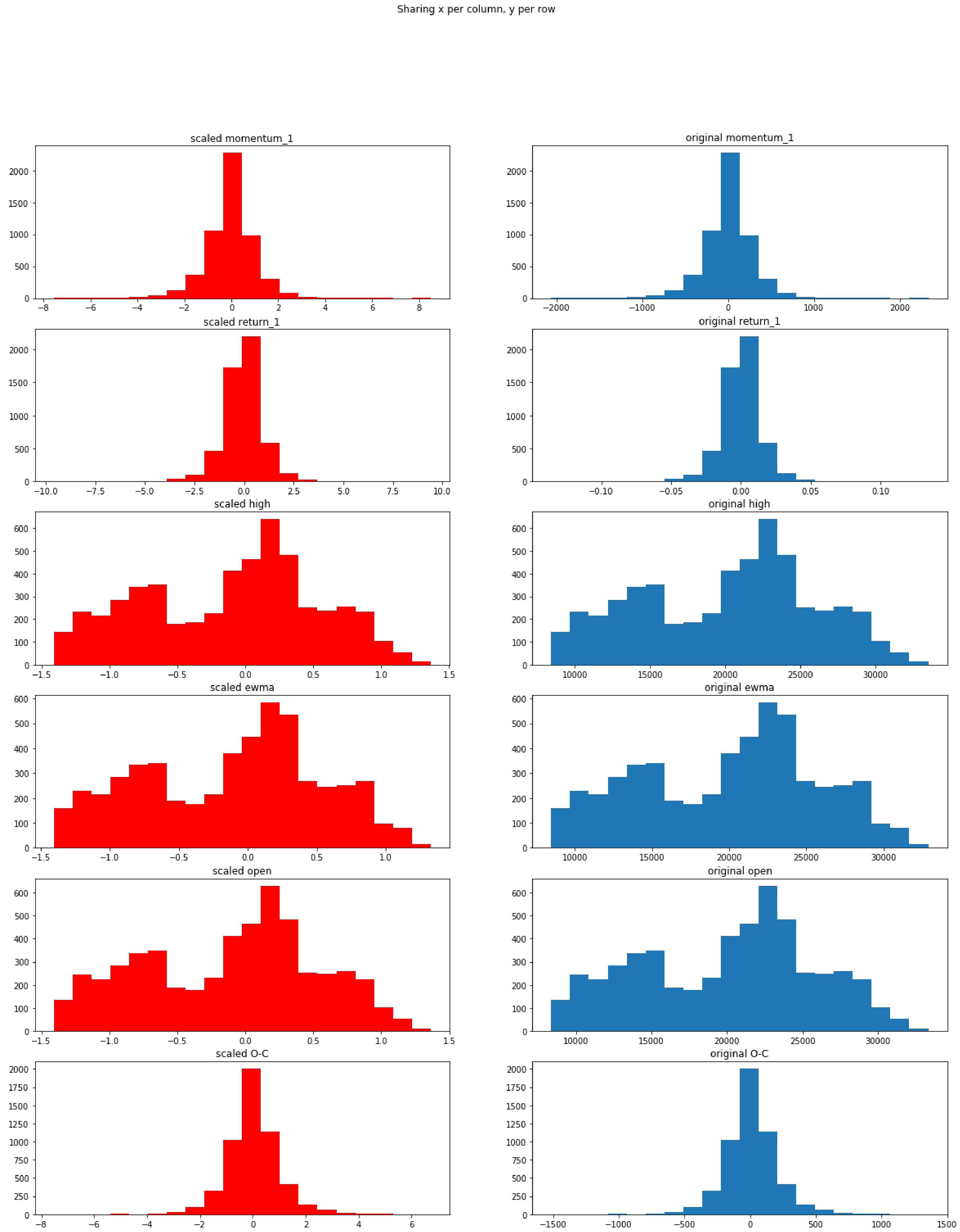
Section 2 - Model Selection

21. Return to preprocessing and fit one model using 2 types of scaling on features data. Produce histograms of changed data columns. Report on changes to coefficients.

2.1.(a) min-max scaling or robust quantiles-based scaling;

below is a trial of scaling on group A with RobustScaler, to best show the histograms, here I'm going to only plot a subset ('momentum_1', 'return_1', 'high', 'ewma', 'open', 'O-C') of all scaled columns rather than all (signed columns are not changed since they are bool). left column of histograms (in red) are scaled values, and right ones are original. One thing to note is shapes from left to right are not curved out (not changed), only the units are different.

In [17]:
```
import numpy as np
from sklearn import preprocessing

# X and y
y = group_a['sign_1'] # sign of daily return
X = group_a[set(group_a.columns) - set(['date','sign_1'])]
# Create scaler
minmax_scale = preprocessing.RobustScaler()
# Scale or Transform the feature
scaled_X = minmax_scale.fit_transform(X)
scaled_X = pd.DataFrame(scaled_X, columns =X.columns)

## subplots scaled X
plot_cols = ['momentum_1', 'return_1', 'high', 'ewma', 'open', 'O-C']
fig, ((ax1,ax2),(ax3,ax4),(ax5,ax6),(ax7,ax8),(ax9,ax10),(ax11,ax12)) = plt.subplots(6,2, figsize=(20,24))
fig.suptitle('Sharing x per column, y per row')

ax1.hist(scaled_X['momentum_1'], bins=20, color="red")
ax1.set_title("scaled momentum_1")
ax2.hist(X['momentum_1'], bins=20)
ax2.set_title("original momentum_1")

ax3.hist(scaled_X['return_1'], bins=20, color="red")
ax3.set_title("scaled return_1")
ax4.hist(X['return_1'], bins=20)
ax4.set_title("original return_1")

ax5.hist(scaled_X['high'], bins=20, color="red")
ax5.set_title("scaled high")
ax6.hist(X['high'], bins=20)
ax6.set_title("original high")

ax7.hist(scaled_X['ewma'], bins=20, color="red")
ax7.set_title("scaled ewma")
ax8.hist(X['ewma'], bins=20)
ax8.set_title("original ewma")

ax9.hist(scaled_X['open'], bins=20, color="red")
ax9.set_title("scaled open")
ax10.hist(X['open'], bins=20)
ax10.set_title("original open")

ax11.hist(scaled_X['O-C'], bins=20, color="red")
ax11.set_title("scaled O-C")
ax12.hist(X['O-C'], bins=20)
ax12.set_title("original O-C")

fig.show()
```

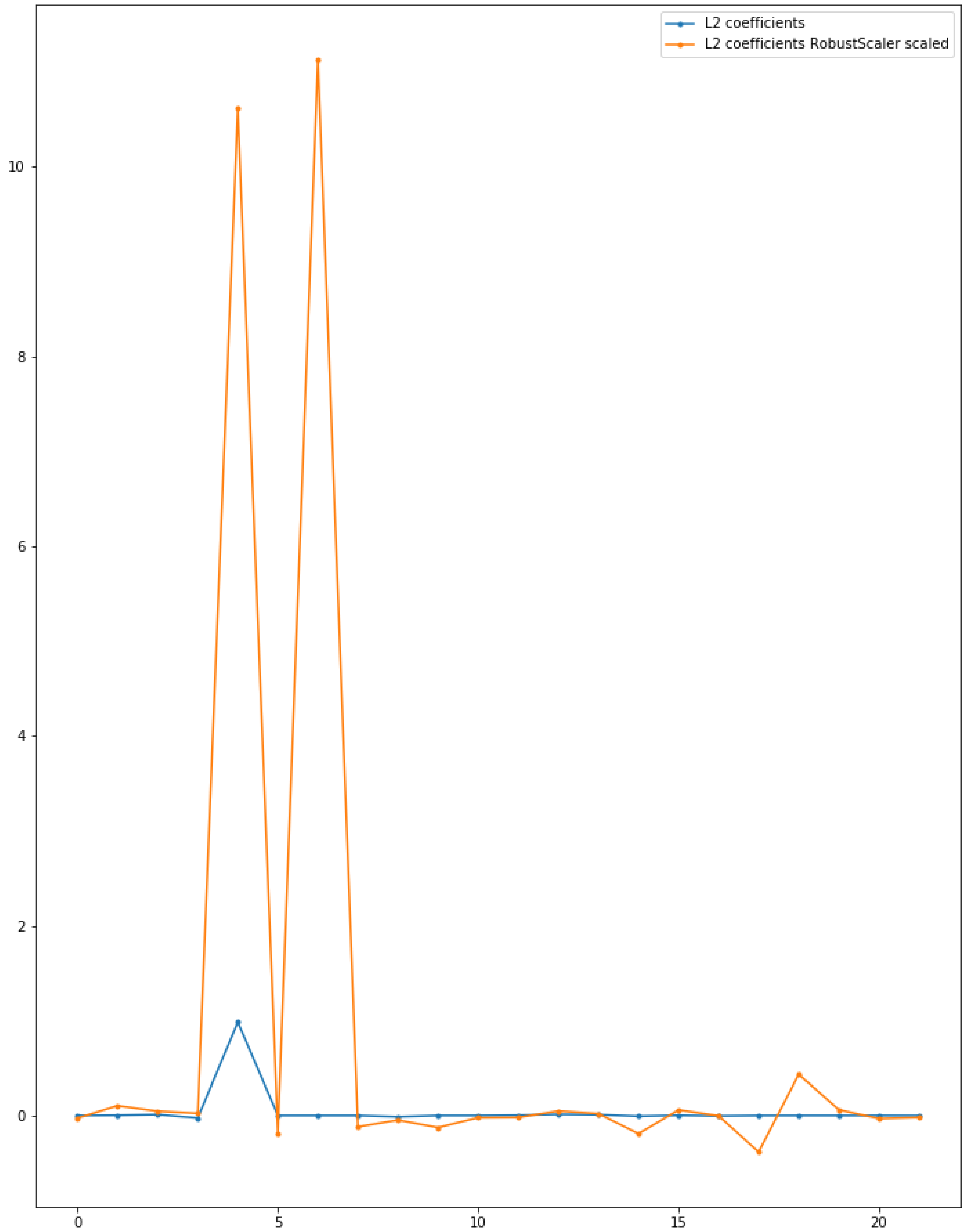


changes to cofficients of robust scaler:

below we gonna repeat L2 logistic regression modelling from section 1.2.1, and looks like from the plotting(each dot on both lines represents a coefficient value) below scaling the feature columns would make the coefficients boosted,

In [18]:

```
from sklearn.linear_model import LogisticRegression

y = group_a['sign_1'] # sign of daily return
X = group_a[set(group_a.columns) - set(['date','sign_1'])]
assert 'sign_1' not in X.columns
# L2 model
l2_model = LogisticRegression(penalty='l2')
l2_model.fit(X, y)
l2_prediction = l2_model.predict(X) # predicting

# L2 model
l2_model_scaled = LogisticRegression(penalty='l2')
l2_model_scaled.fit(scaled_X, y)
l2_prediction_scaled = l2_model_scaled.predict(scaled_X) # predicting

fig = plt.figure(figsize=(12,16))
ax = plt.axes()
ax.plot(l2_model.coef_[0], marker ='.')
ax.plot(l2_model_scaled.coef_[0], marker ='.')
ax.legend(['L2 coefficients', 'L2 coefficients RobustScaler scaled'])
fig.show()
```



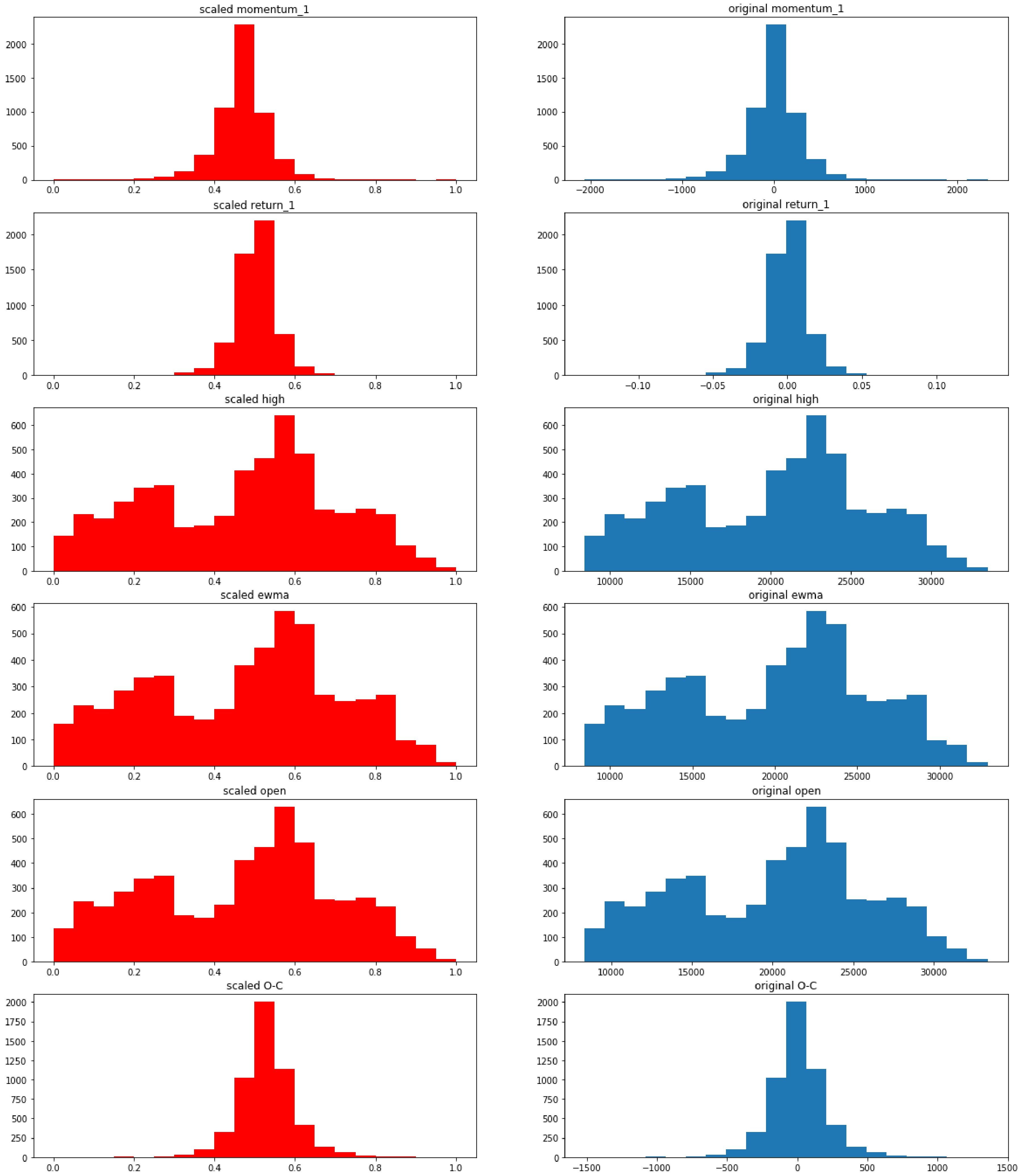
2.1.(b) scaling to uniform [0, 1] range:

below is a trial of scaling on group A with `MinMaxScaler` , to best show the histograms, here I'm going to only plot a subset ('momentum_1', 'return_1', 'high', 'ewma', 'open', 'O-C') of all scaled columns rather than all (signed columns are not changed since they are bool). left column of histograms (in red) are scaled values, and right ones are original. One thing to note is all scaled columns are now ranging from 0 - 1. Also, coefficients are boosted as well compared to L2 model with original unscaled features data

In [19]:

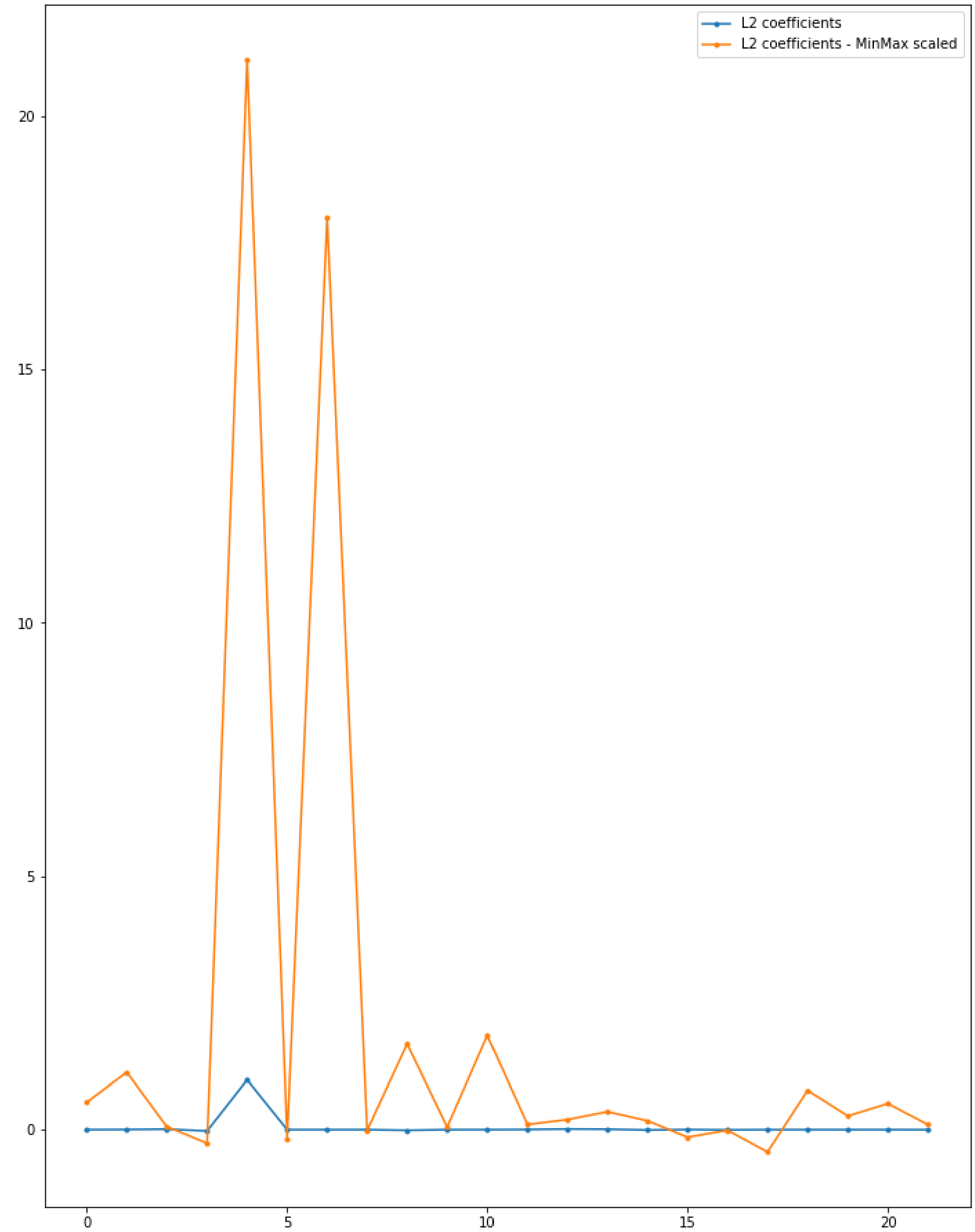
```
import numpy as np
from sklearn import preprocessing

# X and y
y = group_a['sign_1'] # sign of daily return
X = group_a[set(group_a.columns) - set(['date','sign_1'])]
# Create scaler
minmax_scale = preprocessing.MinMaxScaler()
# Scale or Transform the feature
scaled_X = minmax_scale.fit_transform(X)
scaled_X = pd.DataFrame(scaled_X, columns =X.columns)

## subplots scaled X
plot_cols = ['momentum_1', 'return_1', 'high', 'ewma', 'open', 'O-C']
fig, ((ax1,ax2),(ax3,ax4),(ax5,ax6),(ax7,ax8),(ax9,ax10),(ax11,ax12)) = plt.subplots(6,2, figsize=(20,24))
fig.suptitle('Sharing x per column, y per row')

ax1.hist(scaled_X['momentum_1'], bins=20, color="red")
ax1.set_title("scaled momentum_1")
ax2.hist(X['momentum_1'], bins=20)
ax2.set_title("original momentum_1")

ax3.hist(scaled_X['return_1'], bins=20, color="red")
ax3.set_title("scaled return_1")
ax4.hist(X['return_1'], bins=20)
ax4.set_title("original return_1")

ax5.hist(scaled_X['high'], bins=20, color="red")
ax5.set_title("scaled high")
ax6.hist(X['high'], bins=20)
ax6.set_title("original high")

ax7.hist(scaled_X['ewma'], bins=20, color="red")
ax7.set_title("scaled ewma")
ax8.hist(X['ewma'], bins=20)
ax8.set_title("original ewma")

ax9.hist(scaled_X['open'], bins=20, color="red")
ax9.set_title("scaled open")
ax10.hist(X['open'], bins=20)
ax10.set_title("original open")

ax11.hist(scaled_X['O-C'], bins=20, color="red")
ax11.set_title("scaled O-C")
ax12.hist(X['O-C'], bins=20)
ax12.set_title("original O-C")

fig.show()
```

Sharing x per column, y per row



In [20]:

```
from sklearn.linear_model import LogisticRegression

y = group_a['sign_1'] # sign of daily return
X = group_a[set(group_a.columns) - set(['date','sign_1'])]
assert 'sign_1' not in X.columns
# L2 model
l2_model = LogisticRegression(penalty='l2')
l2_model.fit(X, y)
l2_prediction = l2_model.predict(X) # predicting

# L2 model
l2_model_scaled = LogisticRegression(penalty='l2')
l2_model_scaled.fit(scaled_X, y)
l2_prediction_scaled = l2_model_scaled.predict(scaled_X) # predicting

fig = plt.figure(figsize=(12,16))
ax = plt.axes()
ax.plot(l2_model.coef_[0], marker ='.')
ax.plot(l2_model_scaled.coef_[0], marker ='.')
ax.legend(['L2 coefficients', 'L2 coefficients - MinMax scaled'])
fig.show()
```



2.2. Choose a restricted set of features, use penalisation and other (below), and produce a model that makes predictions with reduced variance (MSE). Accuracy score is not estimator variance

2.2(a) formally explain your approach to feature selection or elimination. For example, VIF for elimination of interdependent (colinear) features, or selection according to a score from the F-test.

Gonna use group b data (St Louis Fed Relative Midwest Economy Index) and VIF as an approach to select best limited set of features, and only keep features with VIF value between 5 and 0. The features being kept are `['return_2', 'return_5', 'sign_2', 'sign_3', 'return_3', 'return_6', 'sign_4', 'sign_6', 'sign_1', 'sign_5', 'return_4']` which are sign of T-1 thru T-6 returns and sign of them

In [21]:

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler

def vif(X):
    # perform feature scaling
    scaler = StandardScaler()
    xs = scaler.fit_transform(X)
    # subsume into a dataframe
    vif = pd.DataFrame()
    vif["Features"] = X.columns
    vif["VIF Factor"] = [variance_inflation_factor(xs, i) for i in range(xs.shape[1])]
    return vif

y = group_b['return_1'] # sign of
X = group_b[set(group_b.columns) - set(['date','return_1'])]

result = vif(X)
select_features = result[(result['VIF Factor'] <= 5) &(result['VIF Factor'] > 0)]['Features'].tolist()
print(select_features)
```

```
['return_4', 'sign_1', 'sign_5', 'sign_4', 'return_6', 'sign_3', 'return_2', 'sign_2', 'return_3', 'sign_6', 'return_
5']
```

below is an experiment using group b data, L2 model(C=1) with dependent variable (periodic return) bucketed, 10 bins based on quantile, between full set of features and a subset of features output by VIF metrics. we use 400/532 observations for training, 132/532 as out of sample test. As the result shows, MSE metrics actually decreases a bit if we were to use a subset of features by (3.45 - 2.96)/3.45 = 14%.

In [22]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.metrics import mean_squared_error

# create training and test dataset with full set of features
y = group_b['return_1'] # sign of
X = group_b[set(group_b.columns) - set(['date','return_1'])]
####### bucketing dependent variable ########
est = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='quantile')
y_bucket = est.fit_transform([[i] for i in y.to_list()]) # arrange it as a 2-D array
y_bucket = [i[0] for i in y_bucket]  # flats it back to 1D

y_train, y_test = y_bucket[:400], y_bucket[400:]
X_train_all_features, X_test_features = X[:400], X[400:]

# create training and test dataset with subset of features from VIF experiment
X_subset = group_b[select_features]
X_train_subset_features, X_test_subset_features = X_subset[:400], X_subset[400:]

####### bucketing dependent variable ########
def calc_mse(X_train, y_train, X_test, y_test):
    # L2 model
    l2_model = LogisticRegression(multi_class='multinomial',solver='lbfgs', penalty='l2', C=1)
    #l2_model = LogisticRegression(penalty='l2')
    l2_model.fit(X_train, y_train)
    l2_prediction = l2_model.predict(X_test) # predicting
    return mean_squared_error(y_test, l2_prediction)

print(f"MSE when using full set of features: {calc_mse(X_train_all_features, y_train, X_test_features, y_test)}")
print(f"MSE when using features selected by VIF: {calc_mse(X_train_subset_features, y_train, X_test_subset_features,
```

```
MSE when using full set of features: 3.4545454545454546
MSE when using features selected by VIF: 2.962121212121212
```

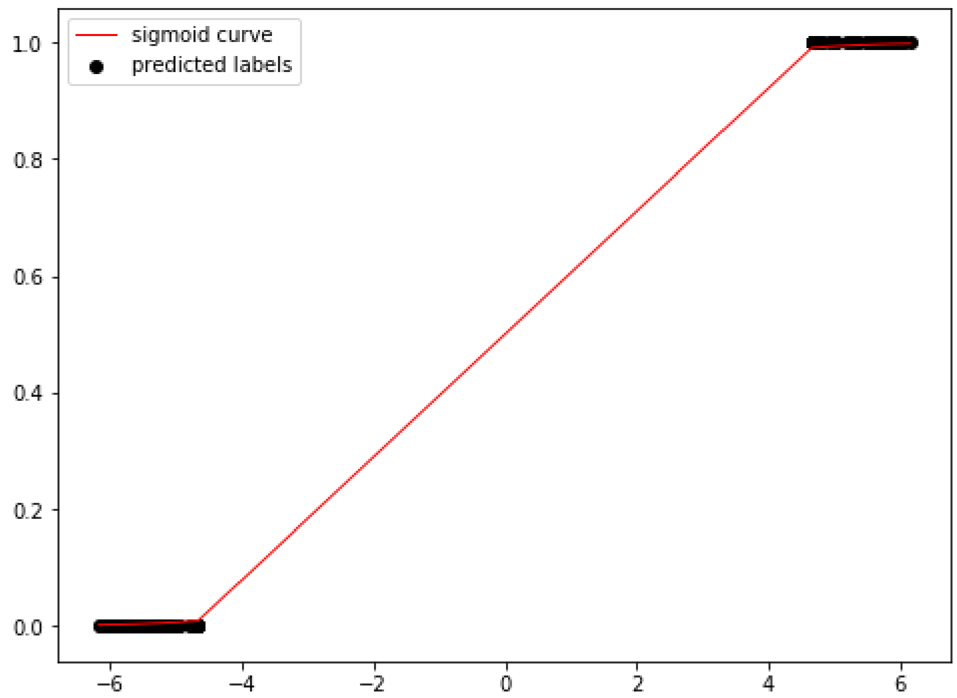
2.2.(b) if prediction task allows (Group A, restricted model) graphically plot a logistic sigmoid for each feature. If prediction is not binomial (Group B), sigmoid only possible between two classes.

not so understand what's `plot a logistic sigmoid for each feature` mean, but here I'm gonna use group a data to train a L2 logistic regression model with features coming from VIF selection, and plot a sigmoid function of subset of features with scatter plotting the predicted labels.

In [23]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from scipy.special import expit

### select subset of features by VIF ###
y = group_a['sign_1'] # sign of daily return
X = group_a[set(group_a.columns) - set(['date','return_1'])]
result = vif(X)
select_features = result[(result['VIF Factor'] <= 5) &(result['VIF Factor'] > 0)]['Features'].tolist()
X = group_a[select_features]

# Scale or Transform the feature
minmax_scale = preprocessing.MinMaxScaler()
scaled_X = minmax_scale.fit_transform(X)
scaled_X = pd.DataFrame(scaled_X, columns =X.columns)

X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size=0.05, random_state=42)

# L2 model
l2_model = LogisticRegression(penalty='l2')
l2_model.fit(X_train, y_train)
l2_prediction = l2_model.predict(X_test) # predicting

# plot sigmod curve
plt.figure(1, figsize=(8, 6))
plt.clf()
# sort data
X_test['row_sum'] = ((X_test * l2_model.coef_).sum(axis=1) + l2_model.intercept_)
X_test.sort_values('row_sum',inplace=True)
row_sum = X_test['row_sum']
X_test.drop('row_sum', axis=1, inplace=True)

plt.scatter(row_sum, l2_model.predict(X_test), color='black', zorder=1)

# sigmoid curve somehow it looks like a straightline below -_-
loss = expit(row_sum).ravel()

plt.plot(row_sum, loss, color='red', linewidth=1)
plt.legend(["sigmoid curve", 'predicted labels'])
```

Out[23]: <matplotlib.legend.Legend at 0x216e8940508>



2.3. For the winning restricted model produce a suitable set of Evaluation metrics, such as area under ROC curve (each class) and confusion matrix. Give expressions for precision/recall.

Using group a data for this question as a demo, below is code to do a grid search with cross validation to find the best model to predict sign of T-1 daily return of HSI data. And cells following it would be some evaluation metrics to gauge the best model.

The best model is:

best params: {'logistic__C': 0.01, 'logistic__penalty': 'l2'}

best score: 0.7835389627765361

In [31]:

```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import balanced_accuracy_score, mean_squared_error, roc_curve
from sklearn import metrics

### select subset of features by VIF ###
y = group_a['sign_1'] # sign of daily return
est = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='quantile')
#y_bucket = est.fit_transform([[i] for i in y.to_list()]) # arrange it as a 2-D array
#y_bucket = [i[0] for i in y_bucket]  # flats it back to 1D

X = group_a[set(group_a.columns) - set(['date','sign_1'])]
result = vif(X)
select_features = result[(result['VIF Factor'] <= 5) &(result['VIF Factor'] > 0)]['Features'].tolist()
X = group_a[select_features]

pipe = Pipeline([("scaler", StandardScaler()), ("logistic", LogisticRegression(solver='liblinear'))])
pipe.fit(X, y)

# set up a
penalty = ['l1', 'l2']
C = np.linspace(0.01,10,10)
param_grid = dict(logistic__C=C, logistic__penalty=penalty)

scoring = {'AUC': 'roc_auc', 'Accuracy': 'accuracy'}
gridsearch = GridSearchCV(pipe, param_grid, n_jobs=1, cv=5, verbose=1,
                          scoring =scoring, refit='AUC')
# fit grid search
best_model = gridsearch.fit(X, y)
print("best params: ", best_model.best_params_)
print("best score: ", best_model.best_score_)
```

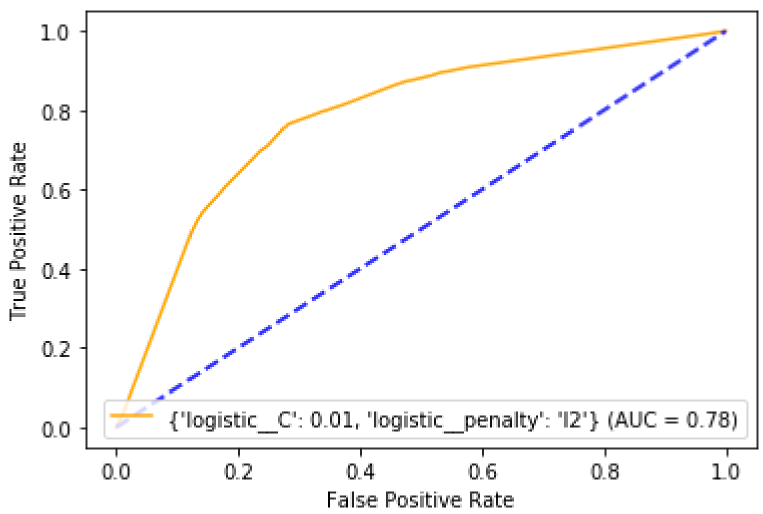
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

best params:  {'logistic__C': 0.01, 'logistic__penalty': 'l2'}
best score:  0.7835389627765361

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.3s finished
```

Evaluation metrics - ROC

Below is ROC Curve metrics (in sample), the orange curve is above the benchmark line (blue dotted), and area under curve (AUC) is 0.78, so the model is not too bad.

In [34]:

```python
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, model_selection, svm

metrics.plot_roc_curve(pipe, X, y, color='orange', name =best_model.best_params_)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='b',
        label='Chance', alpha=.8)

plt.show()
```

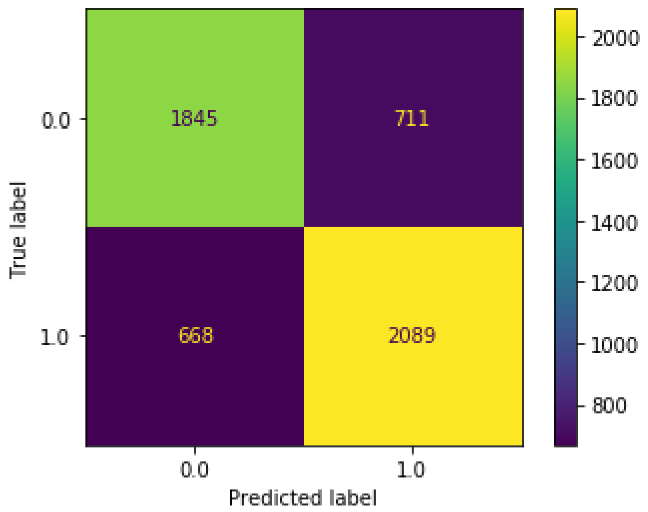


Evaluation metrics - confusion matrix

Below is confusion matrix metrics (in sample), next cell would be a confusion matrix heat map:

In [46]:

```
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

result = confusion_matrix(y, best_model.predict(X))
metrics.plot_confusion_matrix(pipe, X, y)
plt.show()

true_negative, false_positive, false_negative ,true_positive = result[0][0], result[0][1], result[1][0], result[1][1
print(f"True Negative: {true_negative}")
print(f"False Positive: {false_positive}")
print(f"False Negative: {false_negative}")
print(f"True Positive: {true_positive}")
```




```
True Negative: 1845
False Positive: 711
False Negative: 668
True Positive: 2089
```

and the expression for precision/recall would be :

$$precision = \frac{truepositive}{truepositive + falsepositive} = 2089/(2089 + 711) = 0.746$$

$$recall = \frac{truepositive}{truepositive + falsenegative} = 2089/(2089 + 668) = 0.7577$$

$$accuracy = \frac{truepositive + truenegative}{total} = (2089 + 1845)/(2089 + 1845 + 711 + 668) = 0.74$$

Section 3 - Mathematical bases of supervised learning

$$MSE(\hat{\beta}) = E[(\hat{\beta} - \beta)^2] = Var[\hat{\beta}] + (E[\hat{\beta}] - \beta)^2$$

3.1.(a) can there exist an estimator with the smaller MSE than minimal least squares?

Yes and No, "No" as a first impression of this question, of course not! since it gives the minial squared error according to its definition given model type and number of training data are fixed. BUT "YES" at risk of being over meticulous, estimator of a Lass/ridge regression has an MLS larger than MLS of ordinary linear regression. For example,

Lasso's cost function $\beta = (X^t X + \mu * I) - 1X^t Y$

ordinary linear regression's cost function $\beta = (X^t X) - 1X^t Y$

so of course Lasso's cost function is minimal when $\mu = 0$, which is ordinary Linear regression's cost function

3.1(b) for a prediction, does the MSE measure an irreducible error or model error?

Yes. MSE could be described as the addition of model variance, model bias, and irreducible uncertainty, which comes from noise in our data set.

In [ ]: