In [1]:
```python
# Basic libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# For read file from url
import io
import requests

# Set font sizes in plots
sns.set(font_scale = 1.)
# Display all columns
pd.set_option('display.max_columns', None)
```

In [2]:
```python
# Read in NYSE data from url
url = "https://raw.githubusercontent.com/ucla-econ-425t/2023winter/master/sl
s = requests.get(url).content.decode('utf-8')
NYSE = pd.read_csv(io.StringIO(s), index_col = 0)
NYSE
```

Out[2]:

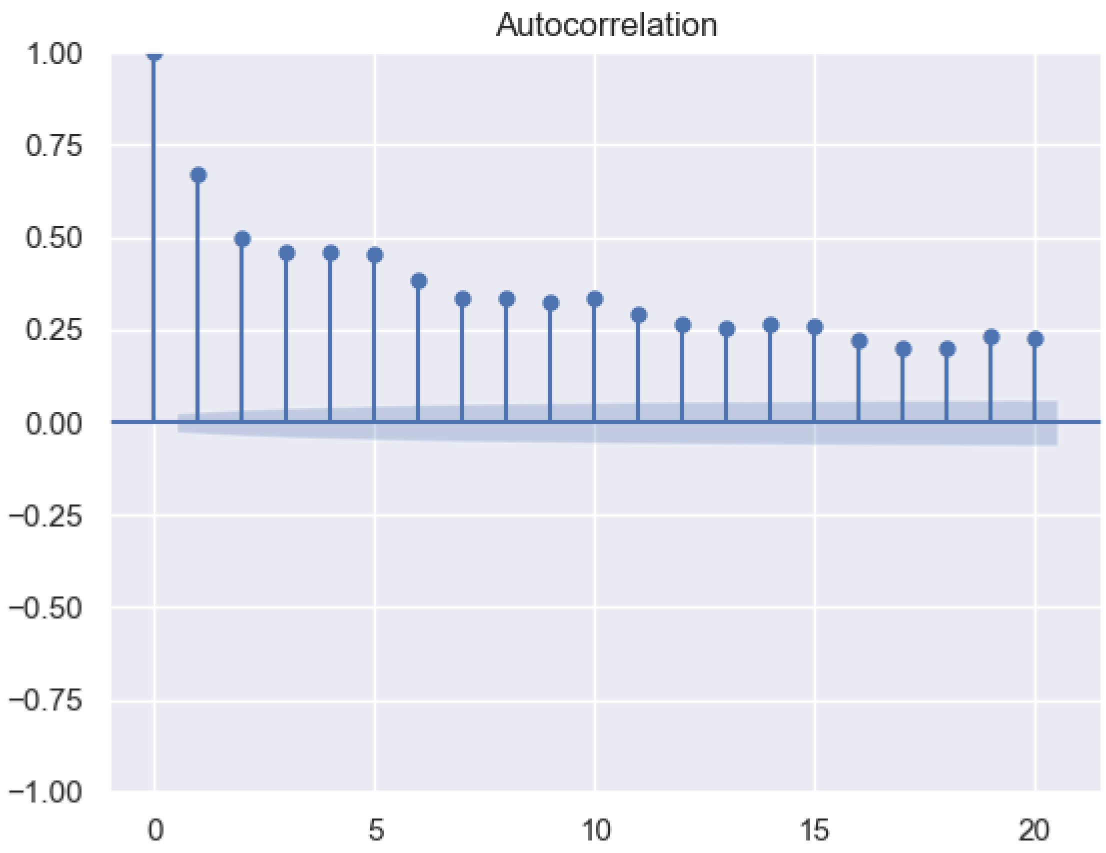
	day_of_week	DJ_return	log_volume	log_volatility	train
date					
1962-12-03	mon	-0.004461	0.032573	-13.127403	True
1962-12-04	tues	0.007813	0.346202	-11.749305	True
1962-12-05	wed	0.003845	0.525306	-11.665609	True
1962-12-06	thur	-0.003462	0.210182	-11.626772	True
1962-12-07	fri	0.000568	0.044187	-11.728130	True
...	...	...	...	...	...
1986-12-24	wed	0.006514	-0.236104	-9.807366	False
1986-12-26	fri	0.001825	-1.322425	-9.906025	False
1986-12-29	mon	-0.009515	-0.371237	-9.827660	False
1986-12-30	tues	-0.001837	-0.385638	-9.926091	False
1986-12-31	wed	-0.006655	-0.264986	-9.935527	False

6051 rows × 5 columns

In [3]:
```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plt.figure()
plot_acf(NYSE['log_volume'], lags = 20)
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



Do a similar plot for (1) the correlation between v_t and lag ℓ Dow Jones return $r_{t-\ell}$ and (2) correlation between v_t and lag ℓ Log volatility $z_{t-\ell}$.

In [4]:
```
L = 5

# 一次性生成每组五个的滞后项
for s in range(1, L+1):
    NYSE[f'DJ_return_lag{s}'] = NYSE['DJ_return'].shift(s)
    NYSE[f'log_volume_lag{s}'] = NYSE['log_volume'].shift(s)
    NYSE[f'log_volatility_lag{s}'] = NYSE['log_volatility'].shift(s)

#按名称将滞后项排列在原项后
NYSE = NYSE.reindex(sorted(NYSE.columns), axis = 1)
```

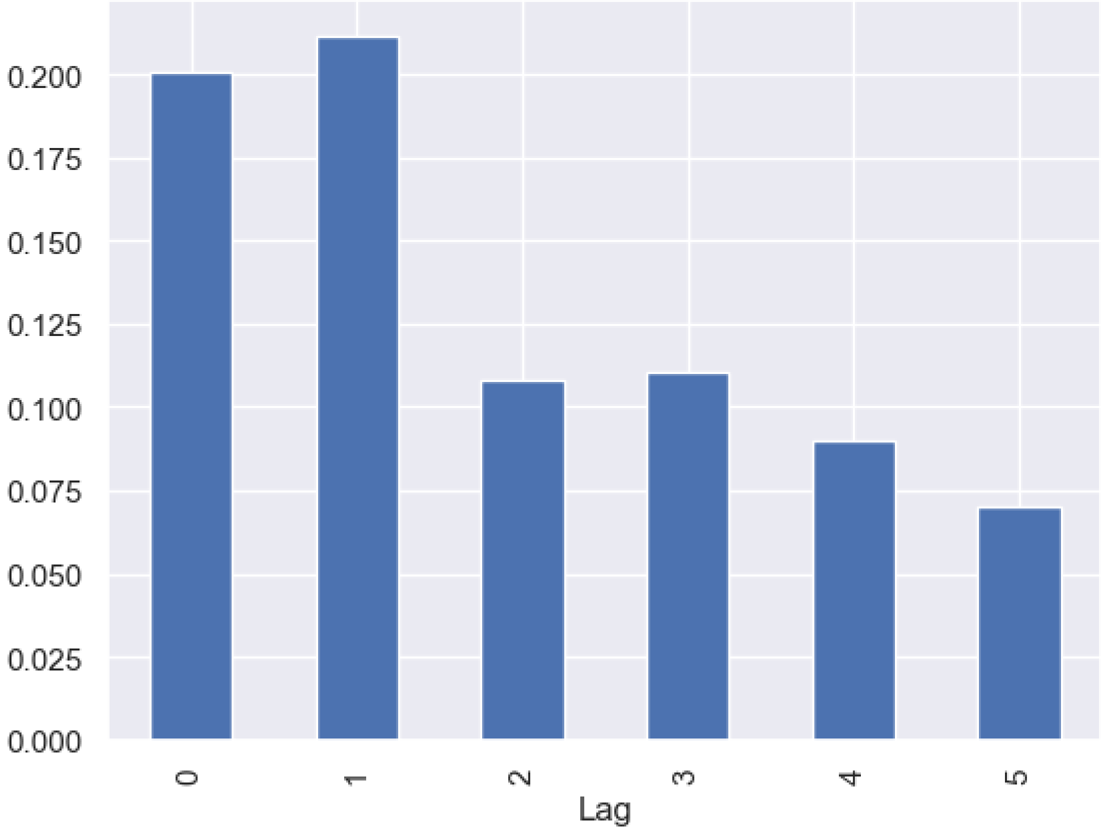
In [5]:
```
#将所有以原项名起头的列（抓取所有滞后）做相关系数
corr = NYSE.filter(regex = "log_volume*|DJ_return*|log_volatility*").corr()
#画相关系数图，背景色为红蓝
corr.style.background_gradient(cmap = "coolwarm")
```

Out[5]:

	DJ_return	DJ_return_lag1	DJ_return_lag2	DJ_return_lag3	DJ_return_la
DJ_return	1.000000	0.143388	-0.003597	-0.005304	-0.0055
DJ_return_lag1	0.143388	1.000000	0.143365	-0.003752	-0.0052
DJ_return_lag2	-0.003597	0.143365	1.000000	0.143336	-0.0037
DJ_return_lag3	-0.005304	-0.003752	0.143336	1.000000	0.1433
DJ_return_lag4	-0.005528	-0.005278	-0.003744	0.143389	1.0000
DJ_return_lag5	-0.014239	-0.005428	-0.005249	-0.003602	0.1433
log_volatility	0.026793	0.021996	0.017013	0.011547	0.0003
log_volatility_lag1	0.014177	0.026778	0.021991	0.016992	0.011
log_volatility_lag2	0.013557	0.014163	0.026773	0.021972	0.0169
log_volatility_lag3	0.010581	0.013561	0.014164	0.026780	0.0219
log_volatility_lag4	0.008297	0.010570	0.013557	0.014149	0.0267
log_volatility_lag5	0.009971	0.008303	0.010572	0.013568	0.014
log_volume	0.200892	0.211669	0.108032	0.110325	0.090
log_volume_lag1	0.047600	0.200776	0.211648	0.107846	0.110
log_volume_lag2	0.015934	0.047396	0.200756	0.211410	0.107
log_volume_lag3	0.008729	0.015730	0.047345	0.200523	0.211
log_volume_lag4	-0.004202	0.007997	0.015549	0.046402	0.2012
log_volume_lag5	0.002981	-0.004333	0.007959	0.015366	0.0464

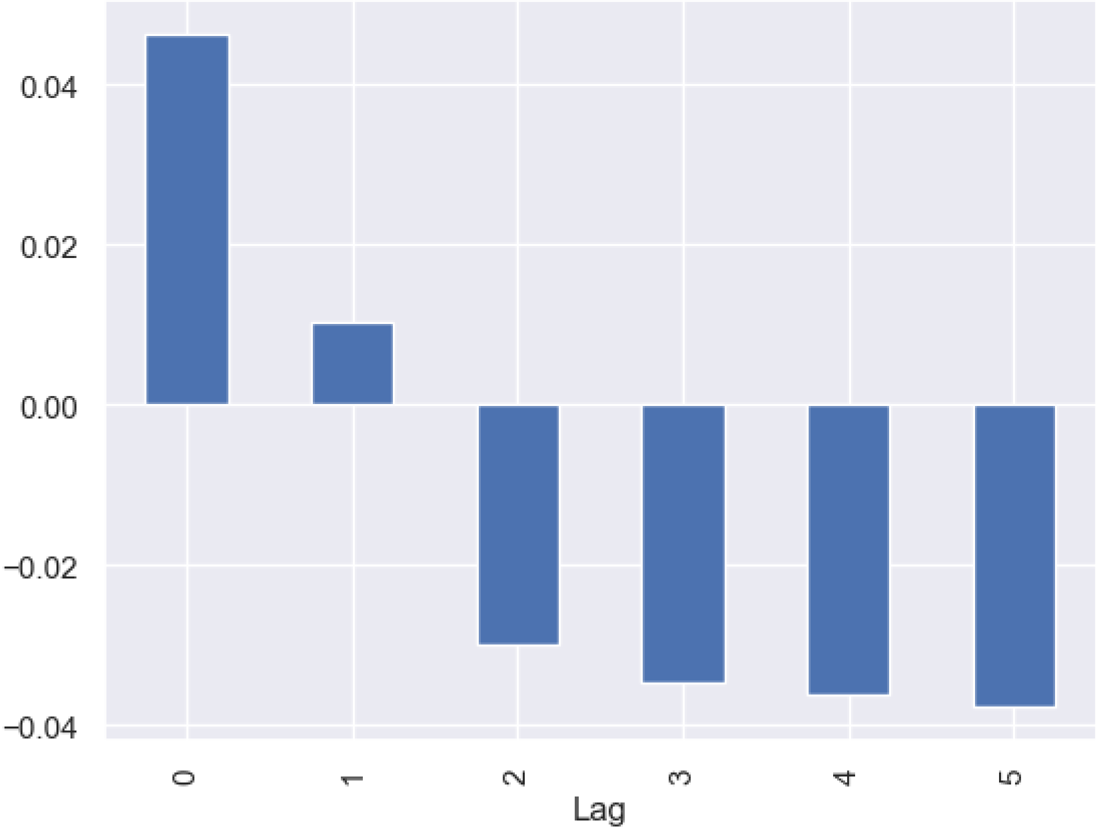
In [143…

```
plt.figure()
#抓取log_volume和所有以DJ_return开头的原项和滞后项相关系数作图
corr['log_volume'].filter(regex = 'DJ_return*').plot(
    kind = 'bar',
#将x轴坐标由原列名变成0-5
    x = range(1, L+1),
    use_index = False
    ).set_xlabel('Lag')
plt.show()
```



In [144…

```
plt.figure()
#抓取log_volume和所有以log_volatility开头的原项和滞后项相关系数作图
corr['log_volume'].filter(regex = 'log_volatility*').plot(
   kind = 'bar',
#将x轴坐标由原列名变成0-5
   x = range(1, L+1),
   use_index = False
   ).set_xlabel('Lag')
plt.show()
```



Project goal: use the previous five trading days' data to forecast today's log trading volume. Use the R^2 between forecast and actual values as the cross validation and test evaluation criterion.

In [35]:
```
# In order to track time
import time

# Scikit-Learn
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import ElasticNet
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Tensorflow
import tensorflow as tf
from tensorflow import keras

# XGBoost
import xgboost as xgb
```

In [36]:
```
NYSE_train = NYSE[NYSE['train']==True].dropna()
print(NYSE_train.shape)
NYSE_test = NYSE[NYSE['train']==False].dropna()
NYSE_test.shape
```

```
(4276, 20)
```

Out[36]: (1770, 20)

In [37]:
```
X_train = NYSE_train.drop(['train','DJ_return','log_volume','log_volatility'
y_train = NYSE_train['log_volume']

X_test = NYSE_test.drop(['train','DJ_return','log_volume','log_volatility'],
y_test = NYSE_test['log_volume']
```

In [38]:
```
df = pd.DataFrame(columns=['Method', 'In sample R^2', 'Out of sample R^2'])
```

1.baseline method: use yesterday's value of log trading volume to predict that of today

In [39]:
```
r2_baseline_train = r2_score(y_train, X_train['log_volume_lag1'])
r2_baseline_train
```

Out[39]: 0.4199386914132621

In [40]:
```
r2_baseline_test = r2_score(y_test, X_test['log_volume_lag1'])
r2_baseline_test
```

Out[40]: 0.18026287838158628

In [41]:
```
new_row = {'Method': 'Straw Man', 'In sample R^2': r2_baseline_train, 'Out c
df = df.append(new_row, ignore_index = True)
df
```

```
/var/folders/g8/bnlxdn656x5f3c7v8y4v67lr0000gn/T/ipykernel_706/2469903245.p
y:2: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index = True)
```

Out[41]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263

2. ElasticNet: Tune AR(5) with elastic net (lasso + ridge) regularization using all 3 features on the training data

In [11]:
```
# 自动挑出非数值行
cat_features = X_train.select_dtypes(exclude = 'float64').columns
# 自动挑出数值行
num_features = X_train.select_dtypes('float64').columns

cat_tf = Pipeline(steps = [
    ("encoder", OneHotEncoder(drop = 'first')),
    ("std", StandardScaler(with_mean = False))
])

num_tf = Pipeline(steps = [
    ("std", StandardScaler())
])

# Column Transformer
enet_col_tf = ColumnTransformer(transformers = [
    ('num', num_tf, num_features),
    ('cat', cat_tf, cat_features)
])
```

In [16]:
```
enet_mod = ElasticNet(
alpha = 1.0,
l1_ratio = 0.5,
max_iter = 100000,
warm_start = True,
random_state = 425,
#selection = 'random'
)
```

In [17]:
```
enet_pipe = Pipeline(steps = [
    ("col_tf", enet_col_tf),
    ("model", enet_mod)
])
```

In [18]:
```
alpha_grid = np.logspace(start = -12, stop = 2, num = 10)
l1_ratio_grid = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
enet_tuned_parameters = {
    "model__alpha": alpha_grid,
    "model__l1_ratio": l1_ratio_grid
}
```

In [19]:
```
enet_search = GridSearchCV(
  enet_pipe,
  enet_tuned_parameters,
  cv = TimeSeriesSplit(5),
  scoring = 'r2',
  refit = True
)
```

In [20]:
```
tic = time.time()
enet_search.fit(X_train, y_train)
toc = time.time()
print('Execution time: ', toc-tic, "seconds")
```

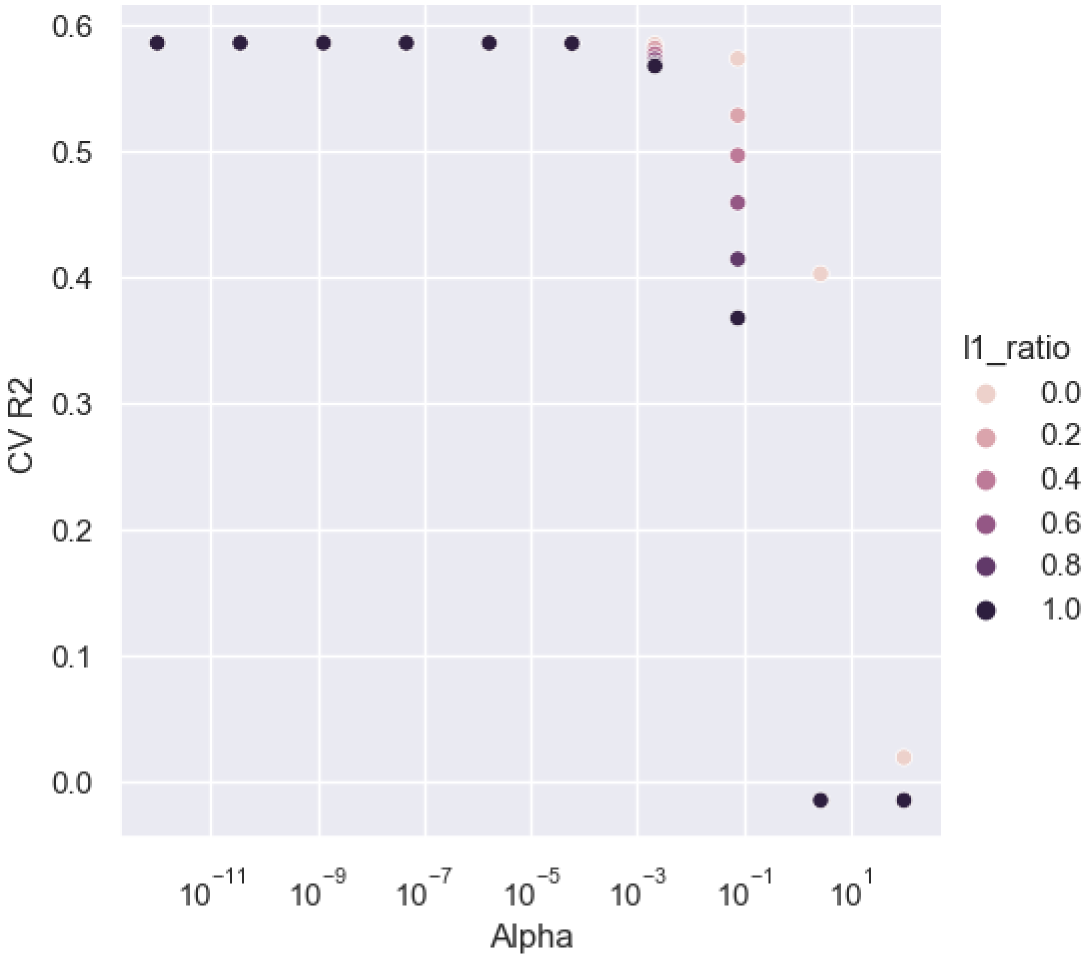
```
/Users/apple/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/
_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations, check the scale of the
features or consider increasing regularisation. Duality gap: 6.779e+01, tole
rance: 1.379e-02 Linear regression models with null weight for the l1 regula
rization term are more efficiently fitted using one of the solvers implement
ed in sklearn.linear_model.Ridge/RidgeCV instead.
  model = cd_fast.enet_coordinate_descent(
/Users/apple/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/
_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations, check the scale of the
features or consider increasing regularisation. Duality gap: 9.369e+01, tole
rance: 1.907e-02 Linear regression models with null weight for the l1 regula
rization term are more efficiently fitted using one of the solvers implement
ed in sklearn.linear_model.Ridge/RidgeCV instead.
  model = cd_fast.enet_coordinate_descent(
Execution time:  160.70518708229065 seconds
/Users/apple/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/
_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge.
You might want to increase the number of iterations, check the scale of the
features or consider increasing regularisation. Duality gap: 4.560e+01, tole
rance: 2.278e-02 Linear regression models with null weight for the l1 regula
rization term are more efficiently fitted using one of the solvers implement
ed in sklearn.linear_model.Ridge/RidgeCV instead.
  model = cd_fast.enet_coordinate_descent(
```

In [21]:

```
cv_res = pd.DataFrame({
  "alpha": np.array(enet_search.cv_results_["param_model__alpha"]),
  "r2": enet_search.cv_results_["mean_test_score"],
  "l1_ratio": enet_search.cv_results_["param_model__l1_ratio"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "alpha",
  y = "r2",
  hue = "l1_ratio"
  ).set(
    xscale = "log",
    xlabel = "Alpha",
    ylabel = "CV R2"
    )
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



In [22]:
```
enet_search.best_estimator_
```

Out[22]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('std',
                                                                   StandardS
caler())]),
                                                  Index(['DJ_return_lag1',
'DJ_return_lag2', 'DJ_return_lag3', 'DJ_return_lag4',
       'DJ_return_lag5', 'log_volatility_lag1', 'log_volatility_lag2',
       'log_volatility_lag3', 'log_volatility_lag4', 'log_volatility_lag5',
       'log_volume_lag1', 'log_volume_lag2', 'log_volume_lag3',
       'log_volume_lag4', 'log_volume_lag5'],
      dtype='object')),
                                                 ('cat',
                                                  Pipeline(steps=[('encode
r',
                                                                   OneHotEnc
oder(drop='first')),
                                                                  ('std',
                                                                   StandardS
caler(with_mean=False))]),
                                                  Index(['day_of_week'], dty
pe='object'))])),
                ('model',
                 ElasticNet(alpha=1e-12, l1_ratio=0.0, max_iter=100000,
                            random_state=425, warm_start=True))])
```

In [23]:
```
r2_train_enet = r2_score(
  y_train,
  enet_search.best_estimator_.predict(X_train)
)
r2_train_enet
```

Out[23]: 0.599718664285801

In [24]:
```
r2_test_enet = r2_score(
  y_test,
  enet_search.best_estimator_.predict(X_test)
)
r2_test_enet
```

Out[24]: 0.4595563133053302

In [233…
```
new_row = {'Method': 'ElasticNet', 'In sample R^2': r2_train_enet, 'Out of s
df = df.append(new_row, ignore_index = True)
df
```

```
/var/folders/g8/bnlxdn656x5f3c7v8y4v67lr0000gn/T/ipykernel_1883/951802392.p
y:2: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index = True)
```

Out[233]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263
1	ElasticNet	0.599719	0.459556

3. MLP: Tune Multiple layer Perceptron using all 3 features on the training data (using scikit-learn)

In [26]:
```
# Column Transformer
mlp_col_tf = ColumnTransformer(transformers = [
    ('num', num_tf, num_features),
    ('cat', cat_tf, cat_features)
])
```

In [27]:
```
mlp_mod = MLPRegressor(
  hidden_layer_sizes = (8, 4),
  activation = 'relu',
  solver = 'adam',
  batch_size = 16,
  random_state = 425
  )
```

In [28]:
```
mlp_pipe = Pipeline(steps = [
    ("col_tf", mlp_col_tf),
    ("model", mlp_mod)
])
```

In [29]:
```
hls_grid = [(4), (8), (12), (4, 2), (8, 4), (12, 6)] # hidden layer size
bs_grid = [40, 120, 200, 280] # batch sizes
mlp_tuned_parameters = {
  "model__hidden_layer_sizes": hls_grid,
  "model__batch_size": bs_grid
  }
```

In [30]:
```
mlp_search = GridSearchCV(
  mlp_pipe,
  mlp_tuned_parameters,
  cv = TimeSeriesSplit(5),
  scoring = 'r2',
```

```
  refit = True
)
```

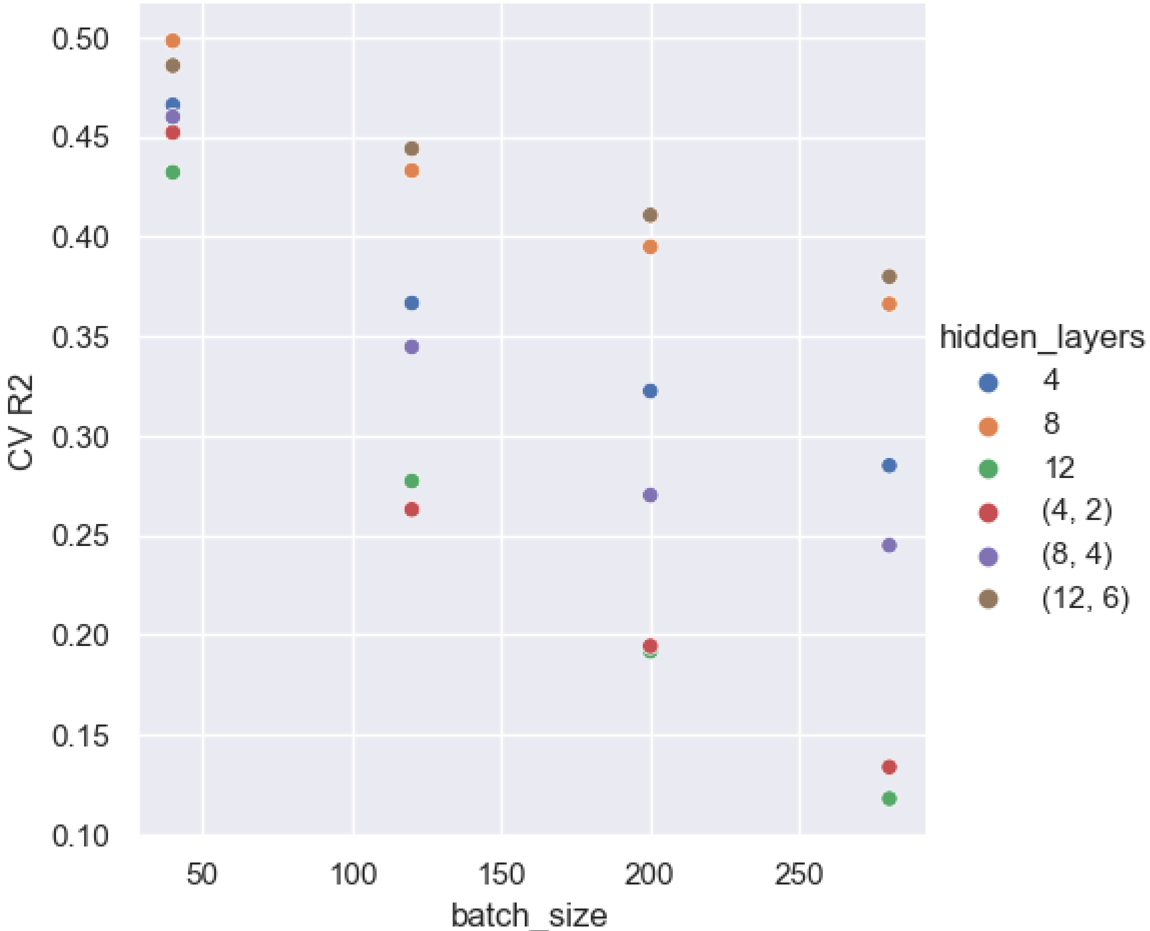
In [31]:
```
tic = time.time()
mlp_search.fit(X_train, y_train)
toc = time.time()
print('Execution time: ', toc-tic, "seconds")
```

```
Execution time:  52.096120834350586 seconds
```

In [32]:
```
cv_res = pd.DataFrame({
  "hidden_layers": np.array(mlp_search.cv_results_["param_model__hidden_laye
  "r2": mlp_search.cv_results_["mean_test_score"],
  "batch_size": mlp_search.cv_results_["param_model__batch_size"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "batch_size",
  y = "r2",
  hue = "hidden_layers"
  ).set(
    xlabel = "batch_size",
    ylabel = "CV R2"
    )
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



In [33]:
```
mlp_search.best_estimator_
```

```
Out[33]: Pipeline(steps=[('col_tf',
                  ColumnTransformer(transformers=[('num',
                                                   Pipeline(steps=[('std',
                                                                    StandardS
caler())]),
                                                   Index(['DJ_return_lag1',
'DJ_return_lag2', 'DJ_return_lag3', 'DJ_return_lag4',
       'DJ_return_lag5', 'log_volatility_lag1', 'log_volatility_lag2',
       'log_volatility_lag3', 'log_volatility_lag4', 'log_volatility_lag5',
       'log_volume_lag1', 'log_volume_lag2', 'log_volume_lag3',
       'log_volume_lag4', 'log_volume_lag5'],
      dtype='object')),
                                                  ('cat',
                                                   Pipeline(steps=[('encode
r',
                                                                    OneHotEnc
oder(drop='first')),
                                                                   ('std',
                                                                    StandardS
caler(with_mean=False))]),
                                                   Index(['day_of_week'], dty
pe='object'))])),
                 ('model',
                  MLPRegressor(batch_size=40, hidden_layer_sizes=8,
                               random_state=425))])
```

```
In [34]: r2_train_mlp = r2_score(
           y_train,
           mlp_search.best_estimator_.predict(X_train)
         )
         r2_train_mlp
```

```
Out[34]: 0.6039243090552505
```

```
In [35]: r2_test_mlp = r2_score(
           y_test,
           mlp_search.best_estimator_.predict(X_test)
         )
         r2_test_mlp
```

```
Out[35]: 0.42431859461874455
```

```
In [43]: new_row = {'Method': 'MLP', 'In sample R^2': r2_train_mlp, 'Out of sample R^
         df = df.append(new_row, ignore_index = True)
         df
```

```
/var/folders/g8/bnlxdn656x5f3c7v8y4v67lr0000gn/T/ipykernel_706/1369428992.p
y:2: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index = True)
```

Out[43]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263
1	ElasticNet	0.599719	0.459556
2	MLP	0.603924	0.424319

4. LSTM: Long Short-Term Memory networks are a special kind of RNN capable of learning long-term dependencies

In [72]:
```python
train_val_split_fraction = 0.8
train_split = int(train_val_split_fraction * int(X_train.shape[0]))

predictors = ['log_volatility', 'DJ_return', 'log_volume']

batch_size = 4
learning_rate = 0.001
epochs = 20
sequence_length = 5
```

In [73]:
```python
train_data = NYSE[predictors].iloc[0 : train_split - L -1]
X_train2 = train_data[[i for i in predictors]].values
y_train = NYSE['log_volume'].iloc[L:train_split]
dataset_train = keras.preprocessing.timeseries_dataset_from_array(
   X_train2,
   y_train,
   sequence_length = sequence_length,
   sampling_rate = 1,
   batch_size = batch_size,
   shuffle = False
)
```

In [74]:
```python
# Sanity Check
for batch in dataset_train.take(1):
    inputs, targets = batch
print("Input shape: ", inputs.numpy().shape)
print("Target shape: ", targets.numpy().shape)
#inputs.numpy()
```

```
Input shape:  (4, 5, 3)
Target shape:  (4,)
```

In [75]:
```python
val_data = NYSE[predictors].iloc[(train_split - L):(X_train.shape[0] - 2)]
X_val = val_data[[i for i in predictors]].values
y_val = NYSE['log_volume'].iloc[train_split:(X_train.shape[0] + L -2)]
dataset_val = keras.preprocessing.timeseries_dataset_from_array(
   X_val,
   y_val,
   sequence_length = sequence_length,
   sampling_rate = 1,
   batch_size = batch_size
)
```

In [76]:
```python
inputs = keras.layers.Input(
   shape = (inputs.shape[1], inputs.shape[2])
)
lstm_out = keras.layers.LSTM(12)(inputs)
outputs = keras.layers.Dense(1)(lstm_out)
```

In [77]:
```python
model = keras.Model(
   inputs = inputs,
   outputs = outputs
)
model.compile(
   optimizer = keras.optimizers.Adam(learning_rate = learning_rate),
   loss = 'mse',
   metrics = [tf.keras.metrics.CosineSimilarity(axis = 1)]
)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 5, 3)]            0         
                                                                 
 lstm_1 (LSTM)               (None, 12)                768       
                                                                 
 dense_1 (Dense)             (None, 1)                 13        
                                                                 
=================================================================
Total params: 781
Trainable params: 781
Non-trainable params: 0
_________________________________________________________________
```

In [78]:

```
history = model.fit(
    dataset_train,
    epochs = epochs,
    validation_data = dataset_val,
    verbose = 2
)
```

```
Epoch 1/20
854/854 - 5s - loss: 0.0521 - cosine_similarity: 0.1670 - val_loss: 0.0433 -
val_cosine_similarity: 0.1612 - 5s/epoch - 6ms/step
Epoch 2/20
854/854 - 3s - loss: 0.0289 - cosine_similarity: 0.5395 - val_loss: 0.0351 -
val_cosine_similarity: 0.3879 - 3s/epoch - 3ms/step
Epoch 3/20
854/854 - 2s - loss: 0.0262 - cosine_similarity: 0.5694 - val_loss: 0.0338 -
val_cosine_similarity: 0.4182 - 2s/epoch - 3ms/step
Epoch 4/20
854/854 - 2s - loss: 0.0259 - cosine_similarity: 0.5735 - val_loss: 0.0335 -
val_cosine_similarity: 0.4182 - 2s/epoch - 3ms/step
Epoch 5/20
854/854 - 2s - loss: 0.0258 - cosine_similarity: 0.5776 - val_loss: 0.0328 -
val_cosine_similarity: 0.4252 - 2s/epoch - 3ms/step
Epoch 6/20
854/854 - 2s - loss: 0.0257 - cosine_similarity: 0.5811 - val_loss: 0.0320 -
val_cosine_similarity: 0.4439 - 2s/epoch - 3ms/step
Epoch 7/20
854/854 - 2s - loss: 0.0255 - cosine_similarity: 0.5806 - val_loss: 0.0311 -
val_cosine_similarity: 0.4533 - 2s/epoch - 3ms/step
Epoch 8/20
854/854 - 3s - loss: 0.0254 - cosine_similarity: 0.5794 - val_loss: 0.0303 -
val_cosine_similarity: 0.4813 - 3s/epoch - 3ms/step
Epoch 9/20
854/854 - 2s - loss: 0.0253 - cosine_similarity: 0.5753 - val_loss: 0.0296 -
val_cosine_similarity: 0.4883 - 2s/epoch - 3ms/step
Epoch 10/20
854/854 - 2s - loss: 0.0252 - cosine_similarity: 0.5747 - val_loss: 0.0291 -
val_cosine_similarity: 0.4883 - 2s/epoch - 3ms/step
Epoch 11/20
854/854 - 2s - loss: 0.0252 - cosine_similarity: 0.5747 - val_loss: 0.0286 -
val_cosine_similarity: 0.5000 - 2s/epoch - 3ms/step
Epoch 12/20
854/854 - 2s - loss: 0.0251 - cosine_similarity: 0.5764 - val_loss: 0.0283 -
val_cosine_similarity: 0.4977 - 2s/epoch - 3ms/step
Epoch 13/20
854/854 - 2s - loss: 0.0251 - cosine_similarity: 0.5764 - val_loss: 0.0280 -
val_cosine_similarity: 0.5000 - 2s/epoch - 3ms/step
Epoch 14/20
854/854 - 2s - loss: 0.0250 - cosine_similarity: 0.5794 - val_loss: 0.0278 -
val_cosine_similarity: 0.4930 - 2s/epoch - 3ms/step
Epoch 15/20
854/854 - 2s - loss: 0.0250 - cosine_similarity: 0.5794 - val_loss: 0.0276 -
val_cosine_similarity: 0.4836 - 2s/epoch - 3ms/step
Epoch 16/20
854/854 - 3s - loss: 0.0249 - cosine_similarity: 0.5776 - val_loss: 0.0274 -
val_cosine_similarity: 0.4790 - 3s/epoch - 3ms/step
Epoch 17/20
854/854 - 3s - loss: 0.0249 - cosine_similarity: 0.5788 - val_loss: 0.0272 -
val_cosine_similarity: 0.4883 - 3s/epoch - 3ms/step
Epoch 18/20
854/854 - 2s - loss: 0.0248 - cosine_similarity: 0.5776 - val_loss: 0.0271 -
val_cosine_similarity: 0.4860 - 2s/epoch - 3ms/step
Epoch 19/20
854/854 - 2s - loss: 0.0248 - cosine_similarity: 0.5788 - val_loss: 0.0270 -
val_cosine_similarity: 0.4930 - 2s/epoch - 3ms/step
Epoch 20/20
854/854 - 2s - loss: 0.0248 - cosine_similarity: 0.5776 - val_loss: 0.0268 -
val_cosine_similarity: 0.4930 - 2s/epoch - 3ms/step
```

In [79]:

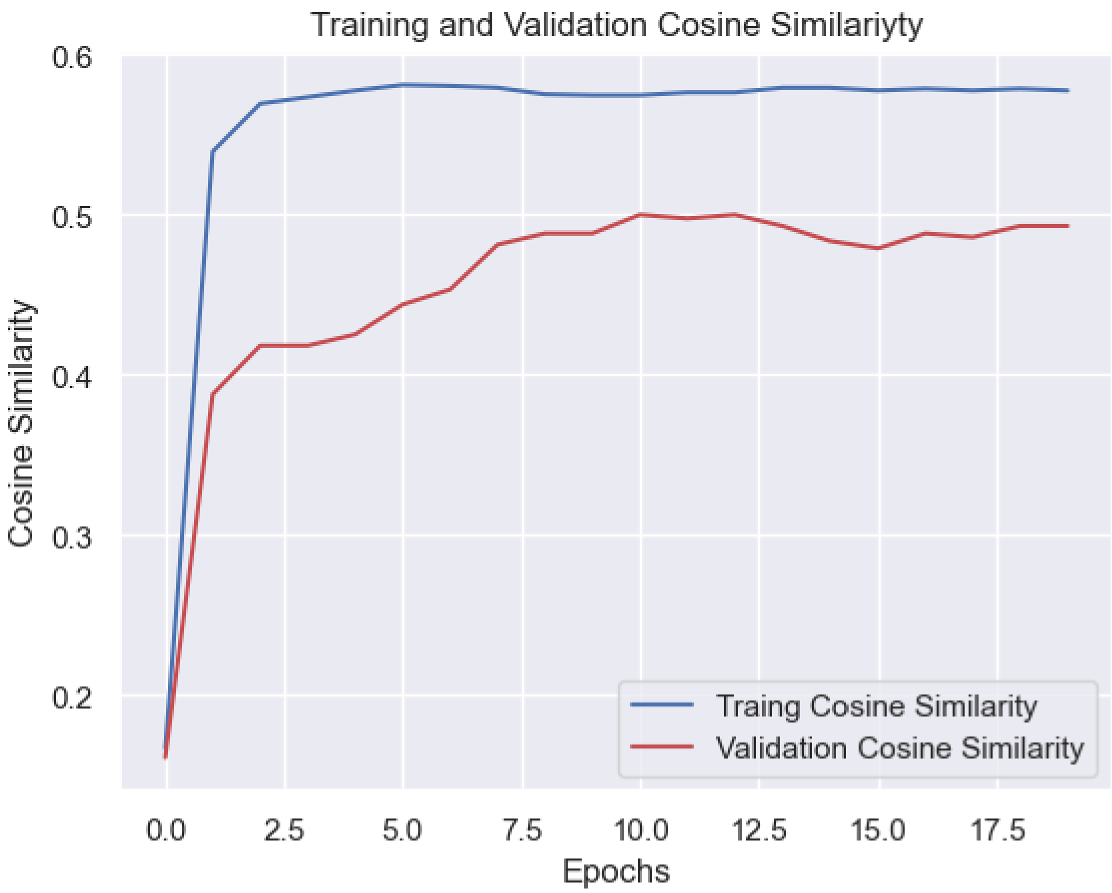
```
def visualize_loss(history, title):
    loss = history.history['loss']
    val_loss = history.history['val_loss']
```

```
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, 'b', label = 'Training loss')
    plt.plot(epochs, val_loss, 'r', label = 'Validation loss')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

visualize_loss(history, "Training and Validation Loss")
```

In [80]:
```
def visulize_cossim(history, title):
    cossim = history.history["cosine_similarity"]
    val_cossim = history.history["val_cosine_similarity"]
    epochs = range(len(cossim))
    plt.figure()
    plt.plot(epochs, cossim, 'b', label = "Traing Cosine Similarity")
    plt.plot(epochs, val_cossim, 'r', label = "Validation Cosine Similarity"
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel("Cosine Similarity")
    plt.legend()
    plt.show()

visulize_cossim(history, "Training and Validation Cosine Similariyty")
```



In [122…

```
train_data = NYSE[predictors].iloc[0 : X_train.shape[0] + 4]
dataset_train = keras.preprocessing.timeseries_dataset_from_array(
   train_data[[i for i in predictors]].values,
   y_train,
   sequence_length = sequence_length,
   sampling_rate = 1,
   batch_size = batch_size,
)
```

In [123…

```
r2_train_lstm = r2_score(
   y_train,
   np.c_[model.predict(
      dataset_train,
      batch_size = batch_size,
      verbose = 2
   )].flatten()
)
r2_train_lstm
```

```
1071/1071 - 2s - 2s/epoch - 2ms/step
```

Out[123]:

```
0.4065061429235378
```

In [107…

```
test_data = NYSE[predictors].iloc[NYSE_train.shape[0] - 5:]
dataset_test = keras.preprocessing.timeseries_dataset_from_array(
   test_data[[i for i in predictors]].values,
   y_test,
   sequence_length = sequence_length,
   sampling_rate = 1,
   batch_size = batch_size,
)
```

In [117…

```
# Sanity Check
for batch in dataset_test.take(1):
```

```
    inputs, targets = batch
print("Input shape: ", inputs.numpy().shape)
print("Target shape: ", targets.numpy().shape)
#inputs.numpy()
#targets.numpy()
#NYSE[predictors].iloc[NYSE_train.shape[0]:(NYSE_train.shape[0] + 7)]
```

```
Input shape:  (4, 5, 3)
Target shape:  (4,)
```

In [110…

```
score, cossim = model.evaluate(
   dataset_test,
   batch_size = batch_size,
   verbose = 2
)
print('Test score: ', score)
print('Test metric: ', cossim)
```

```
443/443 - 1s - loss: 0.0377 - cosine_similarity: 0.4531 - 974ms/epoch - 2ms/
step
Test score:  0.037653662264347076
Test metric:  0.4531073570251465
```

In [111…

```
r2_test_lstm = r2_score(
   y_test,
   np.c_[model.predict(
      dataset_test,
      batch_size = batch_size,
      verbose = 2
   )].flatten()
)
r2_test_lstm
```

```
443/443 - 1s - 860ms/epoch - 2ms/step
```

Out[111]:

```
0.34550259028577346
```

In [235…

```
new_row = {'Method': 'LSTM', 'In sample R^2': r2_train_lstm, 'Out of sample
df = df.append(new_row, ignore_index = True)
df
```

```
/var/folders/g8/bnlxdn656x5f3c7v8y4v67lr0000gn/T/ipykernel_1883/1975925947.p
y:2: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index = True)
```

Out[235]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263
1	ElasticNet	0.599719	0.459556
2	MLP	0.603924	0.424319
3	LSTM	0.406506	0.345503

5. Random Forest: Use the same features as in ElasticNet for the random forest

In [12]:

```
# Column Transformer
rf_col_tf = ColumnTransformer(transformers = [
    ('num', num_tf, num_features),
    ('cat', cat_tf, cat_features)
])
```

In [13]:
```
rf_mod = RandomForestRegressor(
  # Number of trees
  n_estimators = 100,
  criterion = 'squared_error',
  # Number of features to use in each split
  max_features = 'sqrt',
  oob_score = True,
  random_state = 425
  )
```

In [14]:
```
rf_pipe = Pipeline(steps = [
    ("col_tf", rf_col_tf),
    ("model", rf_mod)
])
```

In [15]:
```
# Tune hyper-parameter(s)
B_grid = [800, 1000, 1200]
m_grid = ['sqrt', 1.0] # max_features = 1.0 uses all features
rf_tuned_parameters = {
  "model__n_estimators": B_grid,
  "model__max_features": m_grid
  }
```

In [16]:
```
rf_search = GridSearchCV(
  rf_pipe,
  rf_tuned_parameters,
  cv = TimeSeriesSplit(5),
  scoring = 'r2',
  refit = True
)
```

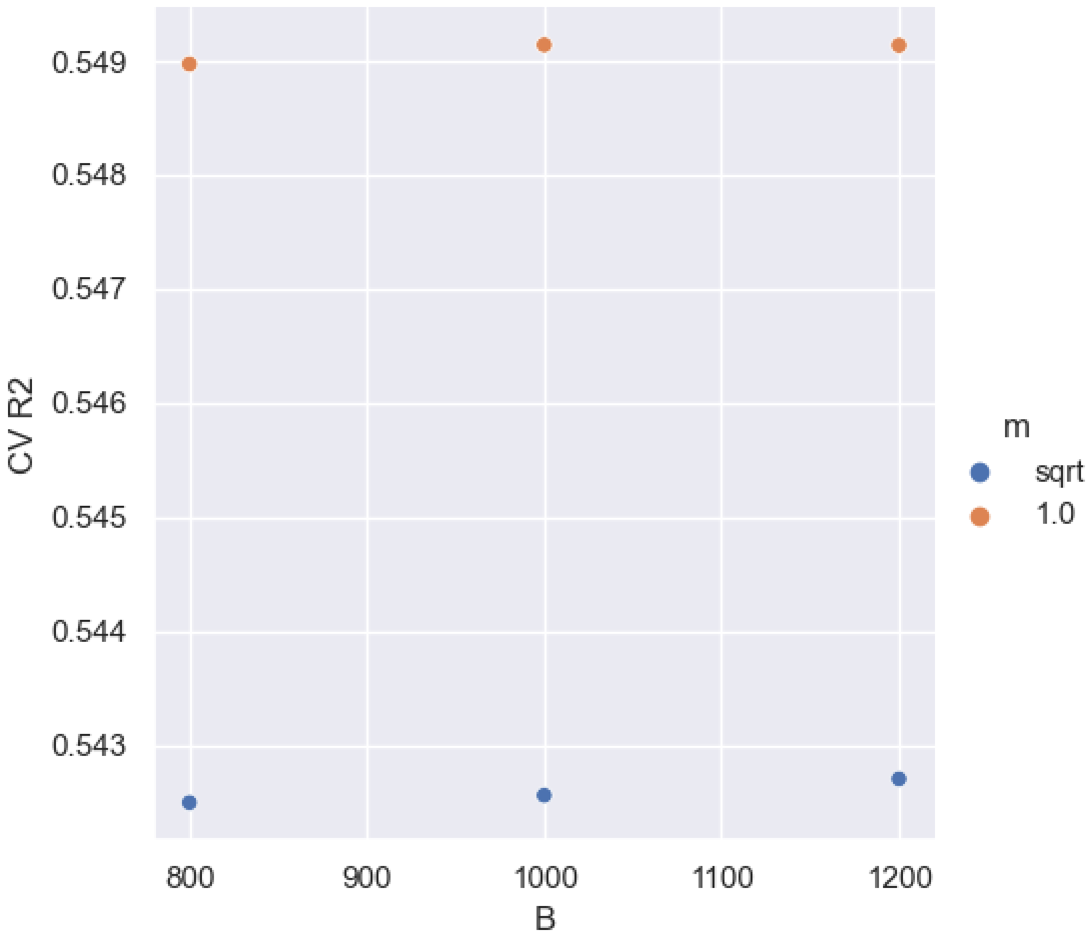
In [17]:
```
tic = time.time()
rf_search.fit(X_train, y_train)
toc = time.time()
print('Execution time: ', toc-tic, "seconds")
```

```
Execution time:  450.868732213974 seconds
```

In [18]:
```
cv_res = pd.DataFrame({
  "B": np.array(rf_search.cv_results_["param_model__n_estimators"]),
  "r2": rf_search.cv_results_["mean_test_score"],
  "m": rf_search.cv_results_["param_model__max_features"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "B",
  y = "r2",
  hue = "m",
  ).set(
    xlabel = "B",
    ylabel = "CV R2"
);
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



In [19]:

```
rf_search.best_estimator_
```

Out[19]:

```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('std',
                                                                   StandardS
caler())]),
                                                  Index(['DJ_return_lag1',
'DJ_return_lag2', 'DJ_return_lag3', 'DJ_return_lag4',
       'DJ_return_lag5', 'log_volatility_lag1', 'log_volatility_lag2',
       'log_volatility_lag3', 'log_volatility_lag4', 'log_volatility_lag5',
       'log_volume_lag1', 'log_volume_lag2', 'log_volume_lag3',
       'log_volume_lag4', 'log_volume_lag5'],
      dtype='object')),
                                                 ('cat',
                                                  Pipeline(steps=[('encode
r',
                                                                   OneHotEnc
oder(drop='first')),
                                                                  ('std',
                                                                   StandardS
caler(with_mean=False))]),
                                                  Index(['day_of_week'], dty
pe='object'))])),
                ('model',
                 RandomForestRegressor(max_features=1.0, n_estimators=1000,
                                       oob_score=True, random_state=425))])
```

In [20]:

```
r2_train_rf = r2_score(
  y_train,
  rf_search.best_estimator_.predict(X_train)
)
r2_train_rf
```

Out[20]: 0.9444562218455823

In [21]:
```
r2_test_rf = r2_score(
  y_test,
  rf_search.best_estimator_.predict(X_test)
)
r2_test_rf
```

Out[21]: 0.42534024466197673

In [45]:
```
new_row = {'Method': 'Random Forest', 'In sample R^2': r2_train_rf, 'Out of
df = df.append(new_row, ignore_index = True)
df
```

```
/var/folders/g8/bnlxdn656x5f3c7v8y4v67lr0000gn/T/ipykernel_706/1244617294.p
y:2: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index = True)
```

Out[45]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263
1	ElasticNet	0.599719	0.459556
2	MLP	0.603924	0.424319
3	LSTM	0.406506	0.345503
4	Random Forest	0.944456	0.425340

6. Boosting: Use the same features as in ElasticNet for boosting

In [23]:
```
# Column Transformer
bst_col_tf = ColumnTransformer(transformers = [
    ('num', num_tf, num_features),
    ('cat', cat_tf, cat_features)
])
```

In [24]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

bst_mod = AdaBoostRegressor(
  # Default base estimator is DecisionTreeRegressor with max_depth = 3
  base_estimator = DecisionTreeRegressor(max_depth = 3),
  # Number of trees (to be tuned)
  n_estimators = 50,
  # Learning rate (to be tuned)
  learning_rate = 1.0,
  random_state = 425
  )
```

In [25]:
```
bst_pipe = Pipeline(steps = [
    ("col_tf", bst_col_tf),
    ("model", bst_mod)
])
```

In [26]:
```
# Tune hyper-parameter(s)
d_grid = [
  DecisionTreeRegressor(max_depth = 7),
  DecisionTreeRegressor(max_depth = 8),
  DecisionTreeRegressor(max_depth = 9)
```

```
  ]
B_grid = [200, 300, 400, 500, 600]
lambda_grid = [0.025, 0.05, 0.075]
bst_tuned_parameters = {
  "model__base_estimator": d_grid,
  "model__n_estimators": B_grid,
  "model__learning_rate": lambda_grid
  }
```

In [27]:
```
bst_search = GridSearchCV(
  bst_pipe,
  bst_tuned_parameters,
  cv = TimeSeriesSplit(5),
  scoring = 'r2',
  refit = True
)
```

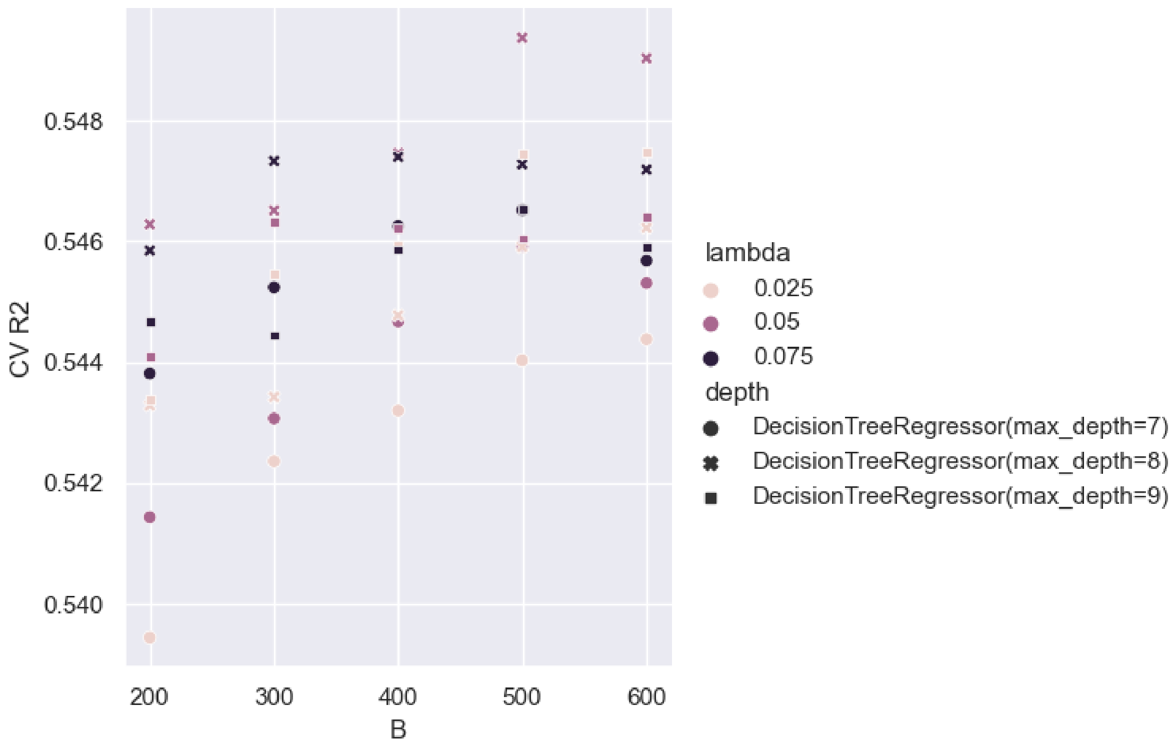
In [28]:
```
tic = time.time()
bst_search.fit(X_train, y_train)
toc = time.time()
print('Execution time: ', toc-tic, "seconds")
```

```
Execution time:  1464.862160205841 seconds
```

In [29]:
```
cv_res = pd.DataFrame({
  "B": np.array(bst_search.cv_results_["param_model__n_estimators"]),
  "r2": bst_search.cv_results_["mean_test_score"],
  "lambda": bst_search.cv_results_["param_model__learning_rate"],
  "depth": bst_search.cv_results_["param_model__base_estimator"],
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "B",
  y = "r2",
  hue = "lambda",
  style = "depth"
  ).set(
    xlabel = "B",
    ylabel = "CV R2"
);
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



In [30]:
```
bst_search.best_estimator_
```

Out[30]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('std',
                                                                   StandardS
caler())]),
                                                  Index(['DJ_return_lag1',
'DJ_return_lag2', 'DJ_return_lag3', 'DJ_return_lag4',
       'DJ_return_lag5', 'log_volatility_lag1', 'log_volatility_lag2',
       'log_volatility_lag3', 'log_volatility_lag4', 'log_volatility_lag5',
       'log_volume_lag1', 'log_volume_lag2', 'log_volume_lag3',
       'log_volume_lag4', 'log_volume_lag5'],
      dtype='object')),
                                                 ('cat',
                                                  Pipeline(steps=[('encode
r',
                                                                   OneHotEnc
oder(drop='first')),
                                                                  ('std',
                                                                   StandardS
caler(with_mean=False))]),
                                                  Index(['day_of_week'], dty
pe='object'))])),
                ('model',
                 AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_
depth=8),
                                   learning_rate=0.05, n_estimators=500,
                                   random_state=425))])
```

In [31]:
```
r2_train_bst = r2_score(
  y_train,
  bst_search.best_estimator_.predict(X_train)
)
r2_train_bst
```

Out[31]:
```
0.8220634337134233
```

In [32]:
```python
r2_test_bst = r2_score(
  y_test,
  bst_search.best_estimator_.predict(X_test)
)
r2_test_bst
```

Out[32]:
```
0.425401004429537
```

In [46]:
```python
new_row = {'Method': 'Boosting', 'In sample R^2': r2_train_bst, 'Out of samp
df = df.append(new_row, ignore_index = True)
df
```

```
/var/folders/g8/bnlxdn656x5f3c7v8y4v67lr0000gn/T/ipykernel_706/1878409706.p
y:2: FutureWarning: The frame.append method is deprecated and will be remove
d from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index = True)
```

Out[46]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263
1	ElasticNet	0.599719	0.459556
2	MLP	0.603924	0.424319
3	LSTM	0.406506	0.345503
4	Random Forest	0.944456	0.425340
5	Boosting	0.822063	0.425401

Summary

In [47]:
```python
df
```

Out[47]:

	Method	In sample R^2	Out of sample R^2
0	Straw Man	0.419939	0.180263
1	ElasticNet	0.599719	0.459556
2	MLP	0.603924	0.424319
3	LSTM	0.406506	0.345503
4	Random Forest	0.944456	0.425340
5	Boosting	0.822063	0.425401

From the result we can see, In sense of in-sample R^2, Random Forest has the highest 0.94, however it's too high and clearly overfitting. Also, LSTM is weaker than the baseline method, this might be not well-trained. In sense of out-of-sample R^2, Boosting performed slightly better than Random Forest, while elasticnet perform the best, this again tell us simple method can do a great job. besides, LSTM perform weakly but still better than baseline method, this might also be not well-trained.