Predicting Dow Jones Trading Volume

In [145…

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import io
import requests
import time
import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
warnings.filterwarnings('ignore')

sns.set(font_scale = 1.)
pd.set_option('display.max_columns', None)
```

In [207…

```
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.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from keras.wrappers.scikit_learn import KerasRegressor

import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, LSTM

import xgboost as xgb
```

Data Visualization and Preparation

In [6]:

```
url = "https://raw.githubusercontent.com/ucla-econ-425t/2023winter/master/slides/data/NYSE.csv"
s = requests.get(url).content.decode('utf-8')
nyse = pd.read_csv(io.StringIO(s), index_col = 0)
nyse
```

Out[6]:

	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

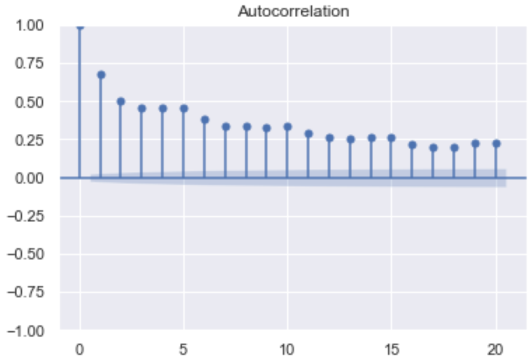
Autocorrelation

In [7]:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

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

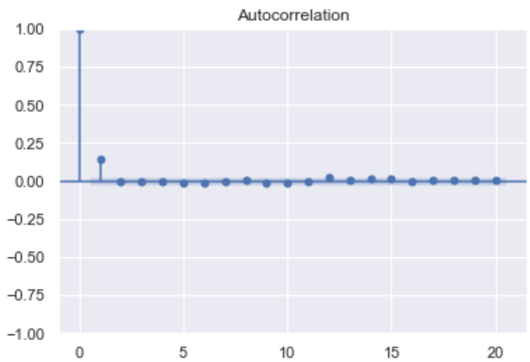
<Figure size 432x288 with 0 Axes>



In [8]:

```
plt.figure()
plot_acf(nyse['DJ_return'], lags = 20)
plt.show()
```

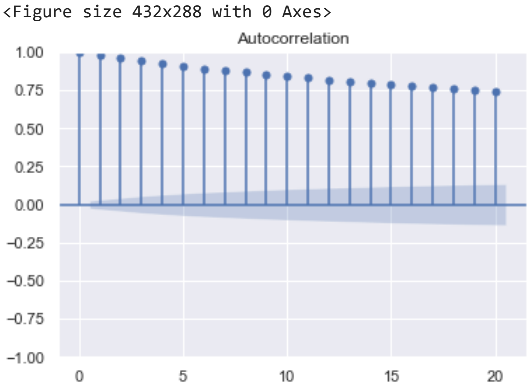
```
<Figure size 432x288 with 0 Axes>
```



In [9]:

```
plt.figure()
plot_acf(nyse['log_volatility'], lags = 20)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Create Lagged Variables

In [10]:

```
L = 5
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)

nyse = nyse.reindex(sorted(nyse.columns), axis = 1)
```

In [11]:

```
nyse
```

Out[11]:

date	DJ_return	DJ_return_lag1	DJ_return_lag2	DJ_return_lag3	DJ_return_lag4	DJ_return_lag5	day_of_week	log_volatility	log_volatility_lag1	log_volatil
1962-12-03	-0.004461	NaN	NaN	NaN	NaN	NaN	mon	-13.127403	NaN	
1962-12-04	0.007813	-0.004461	NaN	NaN	NaN	NaN	tues	-11.749305	-13.127403	
1962-12-05	0.003845	0.007813	-0.004461	NaN	NaN	NaN	wed	-11.665609	-11.749305	-13
1962-12-06	-0.003462	0.003845	0.007813	-0.004461	NaN	NaN	thur	-11.626772	-11.665609	-11
1962-12-07	0.000568	-0.003462	0.003845	0.007813	-0.004461	NaN	fri	-11.728130	-11.626772	-11
...	...	...	...	...	...	...	...	...	...	
1986-12-24	0.006514	-0.006150	-0.001385	0.008345	-0.002866	-0.009262	wed	-9.807366	-9.782214	-9
1986-12-26	0.001825	0.006514	-0.006150	-0.001385	0.008345	-0.002866	fri	-9.906025	-9.807366	-9
1986-12-29	-0.009515	0.001825	0.006514	-0.006150	-0.001385	0.008345	mon	-9.827660	-9.906025	-9
1986-12-30	-0.001837	-0.009515	0.001825	0.006514	-0.006150	-0.001385	tues	-9.926091	-9.827660	-9
1986-12-31	-0.006655	-0.001837	-0.009515	0.001825	0.006514	-0.006150	wed	-9.935527	-9.926091	-9

6051 rows × 20 columns

Correlation

In [14]:

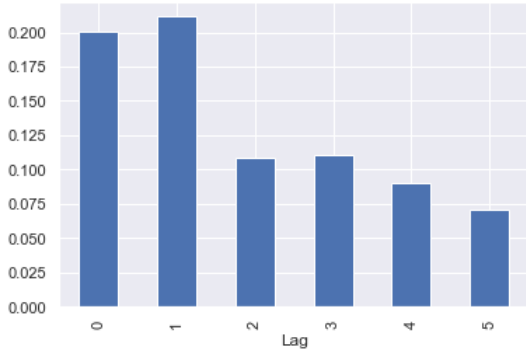
```
corr = nyse.filter(regex = 'log_volume*|DJ_return*|log_volatility*').corr()
corr.style.background_gradient(cmap = 'coolwarm')
```

Out[14]:

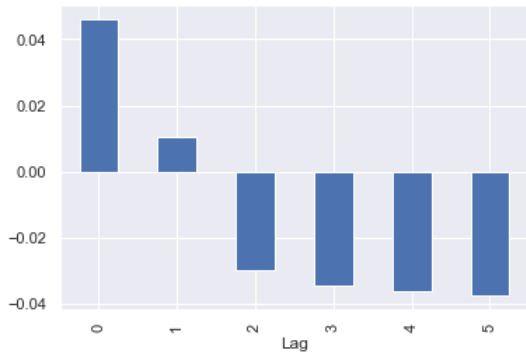
	DJ_return	DJ_return_lag1	DJ_return_lag2	DJ_return_lag3	DJ_return_lag4	DJ_return_lag5	log_volatility	log_volatility_lag1	log_volatility
DJ_return	1.000000	0.143388	-0.003597	-0.005304	-0.005528	-0.014239	0.026793	0.014177	0.01
DJ_return_lag1	0.143388	1.000000	0.143365	-0.003752	-0.005278	-0.005428	0.021996	0.026778	0.01
DJ_return_lag2	-0.003597	0.143365	1.000000	0.143336	-0.003744	-0.005249	0.017013	0.021991	0.02
DJ_return_lag3	-0.005304	-0.003752	0.143336	1.000000	0.143389	-0.003602	0.011547	0.016992	0.02
DJ_return_lag4	-0.005528	-0.005278	-0.003744	0.143389	1.000000	0.143372	0.000350	0.011551	0.01
DJ_return_lag5	-0.014239	-0.005428	-0.005249	-0.003602	0.143372	1.000000	-0.006393	0.000366	0.01
log_volatility	0.026793	0.021996	0.017013	0.011547	0.000350	-0.006393	1.000000	0.980054	0.96
log_volatility_lag1	0.014177	0.026778	0.021991	0.016992	0.011551	0.000366	0.980054	1.000000	0.98
log_volatility_lag2	0.013557	0.014163	0.026773	0.021972	0.016995	0.011565	0.960890	0.980054	1.00
log_volatility_lag3	0.010581	0.013561	0.014164	0.026780	0.021972	0.016994	0.943046	0.960892	0.98
log_volatility_lag4	0.008297	0.010570	0.013557	0.014149	0.026783	0.021983	0.926665	0.943046	0.96
log_volatility_lag5	0.009971	0.008303	0.010572	0.013568	0.014148	0.026778	0.910997	0.926668	0.94
log_volume	0.200892	0.211669	0.108032	0.110325	0.090132	0.070240	0.046306	0.010394	-0.02
log_volume_lag1	0.047600	0.200776	0.211648	0.107846	0.110372	0.090282	0.036211	0.046288	0.01
log_volume_lag2	0.015934	0.047396	0.200756	0.211410	0.107921	0.110601	0.027911	0.036186	0.04
log_volume_lag3	0.008729	0.015730	0.047345	0.200523	0.211503	0.108141	0.021485	0.027885	0.03
log_volume_lag4	-0.004202	0.007997	0.015549	0.046402	0.201233	0.212769	0.015350	0.021425	0.02
log_volume_lag5	0.002981	-0.004333	0.007959	0.015366	0.046438	0.201379	0.003120	0.015331	0.02

In [16]:

```
plt.figure()
corr['log_volume'].filter(regex = 'DJ_return*').plot(
    x = range(1, L+1),
    kind = 'bar',
    use_index = False
    ).set_xlabel('Lag')
plt.show()
```



In [17]:
```
plt.figure()
corr['log_volume'].filter(regex = 'log_volatility*').plot(
    x = range(1, L+1),
    kind = 'bar',
    use_index = False
    ).set_xlabel('Lag')
plt.show()
```



Data split

In [21]:
```
nyse_other = nyse[nyse['train'] == True].dropna()
print(nyse_other.shape)
nyse_test = nyse[nyse['train'] == False]
print(nyse_test.shape)
```

```
(4276, 20)
(1770, 20)
```

In [23]:
```
X_other = nyse_other.drop(['train', 'DJ_return', 'log_volume',
                           'log_volatility', 'day_of_week'], axis = 1)
y_other = nyse_other['log_volume']
X_test = nyse_test.drop(['train', 'DJ_return', 'log_volume',
                         'log_volatility', 'day_of_week'], axis = 1)
y_test = nyse_test['log_volume']
```

In [24]:
```
print('Train set shape: ', X_other.shape, y_other.shape)
print('Test set shape: ', X_test.shape, y_test.shape)
```

```
Train set shape:  (4276, 15) (4276,)
Test set shape:  (1770, 15) (1770,)
```

In [90]:
```
scalar = StandardScaler()
```

Baseline Method

In [25]:
```
r2_train_strawman = r2_score(y_other, X_other['log_volume_lag1'])
print('Strawman train R2:', r2_train_strawman)
```

```
Strawman train R2: 0.4199386914132621
```

In [28]:
```
r2_test_strawman = r2_score(y_test, X_test['log_volume_lag1'])
print('Strawman test R2:', r2_test_strawman)
```

```
Strawman test R2: 0.18026287838158628
```

Elastic net

In [384…
```
enet = ElasticNet(
    alpha = 1.0,
```

```
    l1_ratio = 0.5,
    max_iter = 1000,
    warm_start = True,
    random_state = 425,
)
```

In [31]:

```
enet_pipe = Pipeline(steps = [
    ('std', scalar),
    ('model', enet)
])
enet_pipe
```

Out[31]:

```
Pipeline
  StandardScaler
  ElasticNet
```

In [32]:

```
alpha_grid = np.logspace(start = -7, stop = 2, num = 20)
l1_ratio_grid = np.logspace(start = 0.0, stop = 1.0, num = 20)
enet_params = {
    'model__alpha': alpha_grid,
    'model__l1_ratio': l1_ratio_grid
}
enet_params
```

Out[32]:

```
{'model__alpha': array([1.00000000e-07, 2.97635144e-07, 8.85866790e-07, 2.63665090e-06,
        7.84759970e-06, 2.33572147e-05, 6.95192796e-05, 2.06913808e-04,
        6.15848211e-04, 1.83298071e-03, 5.45559478e-03, 1.62377674e-02,
        4.83293024e-02, 1.43844989e-01, 4.28133240e-01, 1.27427499e+00,
        3.79269019e+00, 1.12883789e+01, 3.35981829e+01, 1.00000000e+02]),
 'model__l1_ratio': array([ 1.        ,  1.12883789,  1.27427499,  1.43844989,  1.62377674,
         1.83298071,  2.06913808,  2.33572147,  2.6366509 ,  2.97635144,
         3.35981829,  3.79269019,  4.2813324 ,  4.83293024,  5.45559478,
         6.15848211,  6.95192796,  7.8475997 ,  8.8586679 , 10.        ])}
```

In [33]:

```
enet_search = GridSearchCV(
    enet_pipe,
    enet_params,
    cv = TimeSeriesSplit(5),
    scoring = 'r2',
    refit = True
)
```

In [43]:

```
tic = time.time()
enet_search.fit(X_other, y_other)
toc = time.time()
print('Execution time:', toc-tic, 'seconds')
```

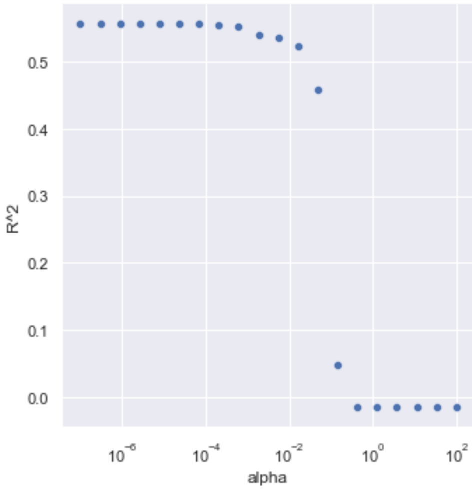
```
Execution time: 9.677505493164062 seconds
```

In [83]:

```
cv_res_enet = pd.DataFrame({
  "alpha": enet_search.cv_results_['param_model__alpha'],
  "r2": enet_search.cv_results_["mean_test_score"]
  })

plt.figure()
sns.relplot(
  data = cv_res_enet,
  x = "alpha",
  y = "r2"
  ).set(
    xlabel = "alpha",
    ylabel = "R^2",
    xscale = "log"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [377…

```
r2_enet_cv = enet_search.best_score_
r2_enet_cv
```

Out[377]:

```
0.556290406384362
```

In [44]:

```
r2_enet_tr = r2_score(
    y_other,
    enet_search.best_estimator_.predict(X_other)
)
r2_enet_tr
```

Out[44]:

```
0.5707150391213718
```

In [45]:

```
r2_enet_ts = r2_score(
    y_test,
    enet_search.best_estimator_.predict(X_test)
)
r2_enet_ts
```

Out[45]:

```
0.41289138590716656
```

Multilayer Perceptron

In [383…

```
mlp = MLPRegressor(
  hidden_layer_sizes = (10, 5),
  activation = 'tanh',
  solver = 'adam',
  batch_size = 16,
  warm_start = True,
  random_state = 425
)
```

In [98]:

```
mlp_pipe = Pipeline(steps = [
    ('std', scalar),
    ('model', mlp)
])
mlp_pipe
```

Out[98]:

```
▸ Pipeline
▸ StandardScaler
▸ MLPRegressor
```

In [99]:

```
hls_grid = [(5), (10), (12), (10, 5), (12, 6), (20, 10)]
bs_grid = [1, 5, 10, 12, 16, 20, 24, 28]
mlp_params = {
  "model__hidden_layer_sizes": hls_grid,
  "model__batch_size": bs_grid
  }
mlp_params
```

Out[99]:

```
{'model__hidden_layer_sizes': [5, 10, 12, (10, 5), (12, 6), (20, 10)],
 'model__batch_size': [1, 5, 10, 12, 16, 20, 24, 28]}
```

In [100…

```
mlp_search = GridSearchCV(
    mlp_pipe,
```

```
    mlp_params,
    cv = TimeSeriesSplit(5),
    scoring = 'r2',
    refit = True,
)
```

In [101…

```
tic = time.time()
mlp_search.fit(X_other, y_other)
toc = time.time()
print('Execution time:', toc-tic, 'seconds')
```

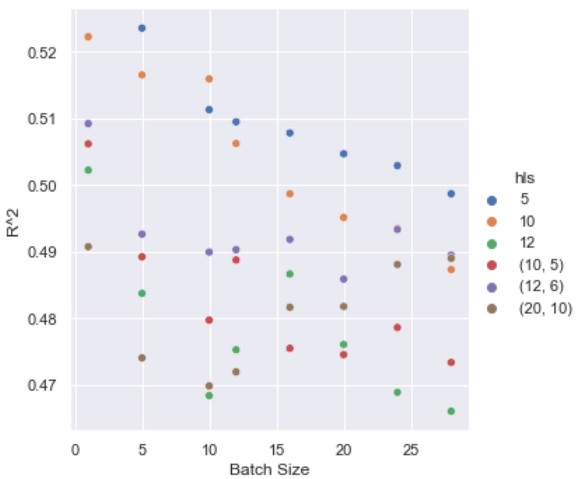
```
Execution time: 735.2154622077942 seconds
```

In [102…

```
cv_res_mlp = pd.DataFrame({
  "bs": np.array(mlp_search.cv_results_["param_model__batch_size"]),
  "r2": mlp_search.cv_results_["mean_test_score"],
  "hls": mlp_search.cv_results_["param_model__hidden_layer_sizes"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res_mlp,
  x = "bs",
  y = "r2",
  hue = "hls"
  ).set(
    # xscale = "log",
    xlabel = "Batch Size",
    ylabel = "R^2"
    );
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [103…

```
r2_mlp_cv = mlp_search.best_score_
r2_mlp_cv
```

Out[103]:

```
0.523593962772549
```

In [104…

```
r2_mlp_tr = r2_score(
    y_other,
    mlp_search.best_estimator_.predict(X_other)
)
r2_mlp_tr
```

Out[104]:

```
0.5749142600453868
```

In [105…

```
r2_mlp_ts = r2_score(
    y_test,
    mlp_search.best_estimator_.predict(X_test)
)
r2_mlp_ts
```

Out[105]:

```
0.40707935871836953
```

LSTM

In [302…

```
def lstm_data(data, lags):
    X = np.zeros((len(data) - lags, lags, 1))
    y = data[lags:]
    for i in range(len(y)):
```

```
        X[i] = data[i:i+lags].values.reshape(-1, 1)
    return X, y

lags = 5
X_other_lstm, y_other_lstm = lstm_data(nyse_other['log_volume'], lags)
X_test_lstm, y_test_lstm = lstm_data(nyse_test['log_volume'], lags)
```

In [318…

```
X_other_lstm_scaled = scalar.fit_transform(X_other_lstm.reshape(-1, lags)).reshape(-1, lags, 1)
X_test_lstm_scaled = scalar.transform(X_test_lstm.reshape(-1, lags)).reshape(-1, lags, 1)
y_other_lstm_scaled = scalar.fit_transform(y_other_lstm.values.reshape(-1, 1)).reshape(-1)
y_test_lstm_scaled = scalar.transform(y_test_lstm.values.reshape(-1, 1)).reshape(-1)
```

In [359…

```
def create_model():
    model = Sequential()
    model.add(LSTM(50, input_shape=(X_other_lstm_scaled.shape[1], X_other_lstm_scaled.shape[2]), activation='tanh'))
    model.add(Dense(1, activation='tanh'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return(model)
```

In [360…

```
lstm = KerasRegressor(build_fn=create_model, verbose=0)
lstm
```

Out[360]:

```
<keras.wrappers.scikit_learn.KerasRegressor at 0x1e7d67d3e20>
```

In [361…

```
lstm_pipe = Pipeline(steps = [
    ('model', lstm)
])
lstm_pipe
```

Out[361]:

Pipeline

KerasRegressor

In [369…

```
e_grid = [10, 20, 50]
bs_grid = [1, 16, 32, 64]
lstm_params = {
    'model__epochs': e_grid,
    'model__batch_size': bs_grid
  }
lstm_params
```

Out[369]:

```
{'model__epochs': [10, 20, 50], 'model__batch_size': [1, 16, 32, 64]}
```

In [370…

```
lstm_search = GridSearchCV(
    lstm_pipe,
    lstm_params,
    cv = TimeSeriesSplit(5),
    scoring = 'r2',
    n_jobs= -1,
    verbose= 1,
    refit = True,
)
```

In [371…

```
tic = time.time()
lstm_search.fit(X_other_lstm_scaled, y_other_lstm_scaled)
toc = time.time()
print('Execution time:', toc-tic, 'seconds')
```

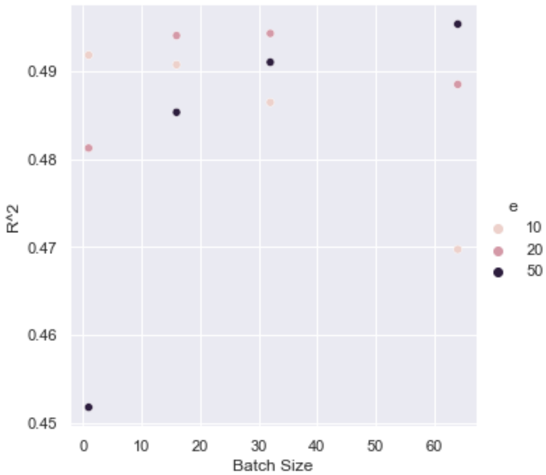
```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Execution time: 1688.1097552776337 seconds
```

In [372…

```
cv_res_lstm = pd.DataFrame({
  "bs": np.array(lstm_search.cv_results_["param_model__batch_size"]),
  "r2": lstm_search.cv_results_["mean_test_score"],
  "e": lstm_search.cv_results_["param_model__epochs"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res_lstm,
  x = "bs",
  y = "r2",
  hue = "e"
  ).set(
    # xscale = "log",
    xlabel = "Batch Size",
    ylabel = "R^2"
    );
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [373…
```
r2_lstm_cv = lstm_search.best_score_
r2_lstm_cv
```

Out[373]:
```
0.49535823048267325
```

In [374…
```
r2_lstm_tr = r2_score(
    y_other_lstm_scaled,
    lstm_search.best_estimator_.predict(X_other_lstm_scaled)
)
r2_lstm_tr
```

Out[374]:
```
0.5126004371056312
```

In [375…
```
r2_lstm_ts = r2_score(
    y_test_lstm_scaled,
    lstm_search.best_estimator_.predict(X_test_lstm_scaled)
)
r2_lstm_ts
```

Out[375]:
```
0.35193385562715396
```

Random Forest

In [264…
```
rf =  RandomForestRegressor(
  n_estimators = 100,
  criterion = 'squared_error',
  max_features = 'sqrt',
  oob_score = True,
  warm_start = True,
  random_state = 425
)
```

In [108…
```
rf_pipe = Pipeline(steps = [
  ("model", rf)
  ])
rf_pipe
```

Out[108]:
```
Pipeline
RandomForestRegressor
```

In [125…
```
B_grid = np.linspace(start = 100, stop = 500, num = 15, dtype=int)
m_grid = ['sqrt', 'log2', 1.0]
rf_params = {
  "model__n_estimators": B_grid,
  "model__max_features": m_grid
  }
rf_params
```

Out[125]:
```
{'model__n_estimators': array([100, 128, 157, 185, 214, 242, 271, 300, 328, 357, 385, 414, 442,
        471, 500]),
 'model__max_features': ['sqrt', 'log2', 1.0]}
```

In [126…
```
rf_search = GridSearchCV(
    rf_pipe,
    rf_params,
    cv = TimeSeriesSplit(5),
    scoring = 'r2',
```

```
    refit = True
)
```

In [127…

```
tic = time.time()
rf_search.fit(X_other, y_other)
toc = time.time()
print('Execution time:', toc-tic, 'seconds')
```

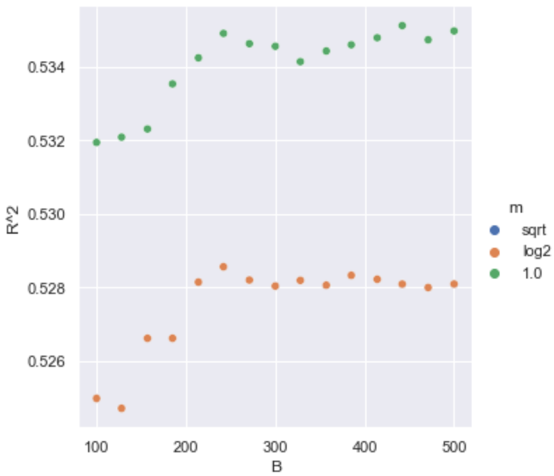
```
Execution time: 1552.903362751007 seconds
```

In [128…

```
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 = "R^2"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [129…

```
r2_rf_cv = rf_search.best_score_
r2_rf_cv
```

Out[129]:

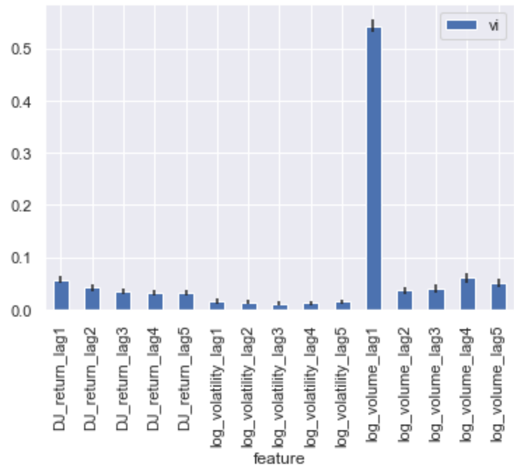
```
0.5351289417184615
```

In [130…

```
rf_vi_df = pd.DataFrame({
  "feature": X_other.columns,
  "vi": rf_search.best_estimator_['model'].feature_importances_,
  "vi_std": np.std([tree.feature_importances_ for tree in rf_search.best_estimator_['model'].estimators_], axis = 0)
  })

plt.figure()
rf_vi_df.plot.bar(x = "feature", y = "vi", yerr = "vi_std")
plt.xticks(rotation = 90);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [131…
```
r2_rf_tr = r2_score(
    y_other,
    rf_search.best_estimator_.predict(X_other)
)
r2_rf_tr
```

Out[131]:
```
0.9412516931565202
```

In [132…
```
r2_rf_ts = r2_score(
    y_test,
    rf_search.best_estimator_.predict(X_test)
)
r2_rf_ts
```

Out[132]:
```
0.40007407761094915
```

Boosting

In [155…
```
xgb =  xgb.XGBRegressor(
  estimator=DecisionTreeRegressor(max_depth=3),
  n_estimators=50,
  verbosity = 0,
  learning_rate=1.0,
  silent = True,
  warm_start = True,
  random_state = 425
)
```

In [156…
```
xgb_pipe = Pipeline(steps = [
  ("model", xgb)
  ])
xgb_pipe
```

Out[156]:
```
Pipeline
  model: XGBRegressor
  estimator: DecisionTreeRegressor
    DecisionTreeRegressor
```

In [157…
```
d_grid = [
  DecisionTreeRegressor(max_depth = 1),
  DecisionTreeRegressor(max_depth = 2),
  DecisionTreeRegressor(max_depth = 3),
  DecisionTreeRegressor(max_depth = 4)
  ]
B_grid = np.linspace(start = 30, stop = 200, num = 20, dtype=int)
lambda_grid = np.linspace(start = 0, stop = 1, num = 6)
xgb_params = {
  "model__estimator": d_grid,
  "model__n_estimators": B_grid,
  "model__learning_rate": lambda_grid
  }
xgb_params
```

Out[157]:

```
{'model__estimator': [DecisionTreeRegressor(max_depth=1),
  DecisionTreeRegressor(max_depth=2),
  DecisionTreeRegressor(max_depth=3),
  DecisionTreeRegressor(max_depth=4)],
 'model__n_estimators': array([ 30,  38,  47,  56,  65,  74,  83,  92, 101, 110, 119, 128, 137,
        146, 155, 164, 173, 182, 191, 200]),
 'model__learning_rate': array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])}
```

In [158…

```
xgb_search = GridSearchCV(
    xgb_pipe,
    xgb_params,
    cv = TimeSeriesSplit(5),
    scoring = 'r2',
    refit = True
)
```

In [159…

```
tic = time.time()
xgb_search.fit(X_other, y_other)
toc = time.time()
print('Execution time:', toc-tic, 'seconds')
```

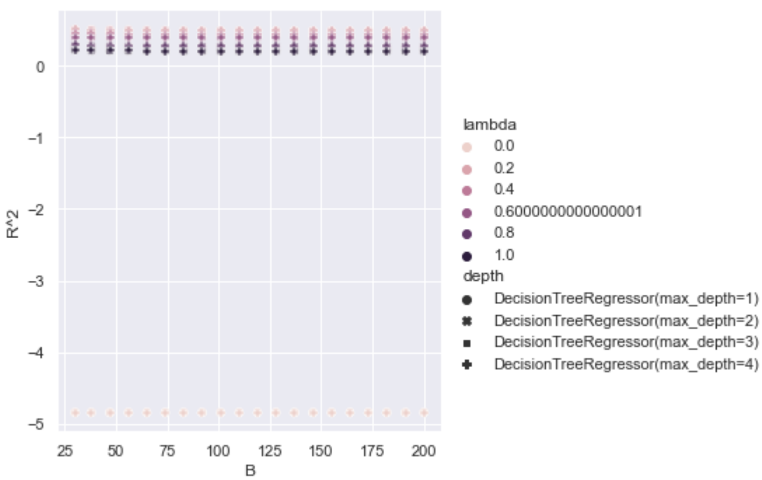
```
Execution time: 1512.7632925510406 seconds
```

In [160…

```
xgb_cv_res = pd.DataFrame({
  "B": np.array(xgb_search.cv_results_["param_model__n_estimators"]),
  "r2": xgb_search.cv_results_["mean_test_score"],
  "lambda": xgb_search.cv_results_["param_model__learning_rate"],
  "depth": xgb_search.cv_results_["param_model__estimator"],
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = xgb_cv_res,
  x = "B",
  y = "r2",
  hue = "lambda",
  style = "depth"
  ).set(
    xlabel = "B",
    ylabel = "R^2"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [161…

```
r2_xgb_cv = xgb_search.best_score_
r2_xgb_cv
```

Out[161]:

```
0.5031755488035357
```

In [162…

```
r2_xgb_tr = r2_score(
    y_other,
    xgb_search.best_estimator_.predict(X_other)
)
r2_xgb_tr
```

Out[162]:

```
0.7920584557040472
```

In [163…

```
r2_xgb_ts = r2_score(
    y_test,
    xgb_search.best_estimator_.predict(X_test)
```

```
)
r2_xgb_ts
```

Out[163]:

```
0.3841513684639135
```

Comparison

In [381…

```
### There is no CV for the Baseline method, I used train R^2 instead

d = {'Model': ["Baseline", "ENet", "MLP", "LSTM", "Random Forest", "Boosting"],
     'CV R^2': [r2_train_strawman, r2_enet_cv, r2_mlp_cv, r2_lstm_cv, r2_rf_cv, r2_xgb_cv],
     'Test R^2': [r2_test_strawman, r2_enet_ts, r2_mlp_ts, r2_lstm_ts, r2_rf_ts, r2_xgb_ts]}
outcome = pd.DataFrame(data=d)
outcome
```

Out[381]:

	Model	CV R^2	Test R^2
0	Baseline	0.419939	0.180263
1	ENet	0.556290	0.412891
2	MLP	0.523594	0.407079
3	LSTM	0.495358	0.351934
4	Random Forest	0.535129	0.400074
5	Boosting	0.503176	0.384151

Elastic Net is has both the highest cross-validation and test R squared.

In [ ]: