Youssef Mahmoud 905854027

In [1]:
```
import pandas as pd
import io
import requests
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV
from sklearn import set_config
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor


set_config(display="diagram")
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
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[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 [5]:

```
ccf1 = ccf(NYSE['log_volume'], NYSE['DJ_return'])
ccf1
```

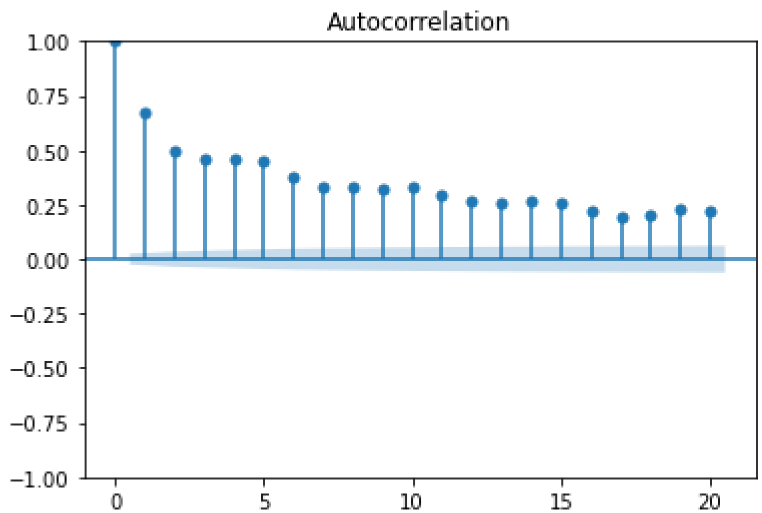
Out[5]:

```
array([ 0.20089212,  0.21169208,  0.10804011, ..., -0.36176712,
       -0.05321751,  0.60390789])
```

In [6]:
```
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 [7]:
```
ccf2 = ccf(NYSE['log_volume'], NYSE['log_volatility'])
ccf2
```

Out[7]:
```
array([ 0.04630603,  0.01039609, -0.02991791, ...,  4.50216451,
        4.90663188,  4.78567114])
```

In [8]:
```
def split_dataframe(df, bool_col='train'):
    train = df[df[bool_col]]
    test = df[~df[bool_col]]
    return train, test
```

In [9]:
```
NYSE_train, NYSE_test = split_dataframe(NYSE, bool_col='train')

```

In [11]:
```
def straw_man_forecast(data):
    return data.shift(1)
```

In [12]:
```
NYSE_test2 = NYSE_test.copy()
```

In [13]:
```
NYSE_test2['forecast'] = straw_man_forecast(NYSE_test2['log_volume'])
```

In [15]:
```
r2_sm = r2_score(NYSE_test2['log_volume'].iloc[1:], NYSE_test2['forecast'].iloc[1:])
r2_sm
```

Out[15]: 0.18073700785807378

In [16]:
```
tscv = TimeSeriesSplit(n_splits=10)
r2_scores = []
NYSE_train2 = NYSE_train.copy()

for train_index, val_index in tscv.split(NYSE_train2):
    train_subset = NYSE_train2.iloc[train_index]
    val_subset = NYSE_train2.iloc[val_index]
    val_subset = val_subset.copy()
    val_subset['forecast'] = straw_man_forecast(val_subset['log_volume'])
    r2 = r2_score(val_subset['log_volume'].iloc[1:], val_subset['forecast'].iloc[1:])
    r2_scores.append(r2)
cv_r2_sm = np.mean(r2_scores)
cv_r2_sm
```

Out[16]: 0.39950110948135525

In [19]:

```
def ar_lags(data, order):
    Y = data.iloc[order:]
    X = np.zeros((len(Y), order + 1))
    X[:, 0] = 1
    columns = ['const'] + [f'lag_{i+1}' for i in range(order)]

    for i in range(order):
        X[:, i + 1] = data.iloc[order - i - 1: -(i + 1)].values

    return pd.DataFrame(X, columns=columns, index=Y.index), pd.DataFrame(Y.values, columns=['val

order = 5
X_train, Y_train = ar_lags(NYSE_train['log_volume'], order)
X_test, Y_test = ar_lags(NYSE_test['log_volume'], order)
```

In [22]:

```
scalar = StandardScaler()
enet = ElasticNet(max_iter=10000)
```

In [23]:

```

pipe_enet = Pipeline(steps = [
  ("std_tf", scalar),
  ("model", enet)
  ])
pipe_enet
```

Out[23]:

```
▸ Pipeline
  ▸ StandardScaler
  ▸ ElasticNet
```

In [25]:
```
alphas = np.logspace(start = -3, stop = 2, base = 10, num = 100)
l1_ratio = np.linspace(0,1,11)
enet_tuned_parameters = {"model__alpha": alphas,"model__l1_ratio":l1_ratio}
```

In [26]:
```
search_enet = GridSearchCV(
  pipe_enet,
  enet_tuned_parameters,
  cv = TimeSeriesSplit(5),
  scoring = "r2",
  refit = True
  )
```

In [27]:
```
search_enet.fit(X_train,Y_train)
```

Out[27]:
```
▸ GridSearchCV
  ▸ StandardScaler
    ▸ ElasticNet
```

In [28]:
```
cv_r2_enet = search_enet.best_score_
cv_r2_enet
```

Out[28]: 0.5325094557515659

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

Out[30]:
```
▸ Pipeline
  ▸ StandardScaler
    ▸ ElasticNet
```

In [32]:
```
r2_enet = r2_score(Y_test, search_enet.best_estimator_.predict(X_test))
r2_enet
```

Out[32]:
```
0.37501629623522603
```

In [33]:
```
mlp = MLPRegressor(
  hidden_layer_sizes = (8, 4),
  activation = 'relu',
  solver = 'adam',
  batch_size = 16,
  random_state = 425
  )
# Create Pipeline
pipe_mlp = Pipeline(steps = [
  ("std_tf", scalar),
  ("model", mlp)
  ])
pipe_mlp
```

Out[33]:

▸ **Pipeline**

▸ StandardScaler

▸ MLPRegressor

In [34]:
```
# Tune hyper-parameter(s)
hls_grid = [(4), (8), (12), (4, 2), (8, 4), (12, 6)] # hidden layer size
bs_grid = [4, 8, 12, 16, 20, 24, 28, 32] # batch sizes
tuned_parameters_mlp = {
  "model__hidden_layer_sizes": hls_grid,
  "model__batch_size": bs_grid
  }
tuned_parameters_mlp
```

Out[34]:
```
{'model__hidden_layer_sizes': [4, 8, 12, (4, 2), (8, 4), (12, 6)],
 'model__batch_size': [4, 8, 12, 16, 20, 24, 28, 32]}
```

In [35]:
```
search_mlp = GridSearchCV(
  pipe_mlp,
  tuned_parameters_mlp,
  cv = TimeSeriesSplit(5),
  scoring = "r2",
  refit = True
  )
```

In [36]:
```
search_mlp.fit(X_train,Y_train)
```

Out[36]:

▸ **GridSearchCV**

▸ StandardScaler

▸ MLPRegressor

In [37]:
```
cv_r2_mlp = search_mlp.best_score_
cv_r2_mlp
```

Out[37]: 0.5201477116339358

In [39]:
```
r2_mlp = r2_score(Y_test, search_mlp.best_estimator_.predict(X_test))
r2_mlp
```

Out[39]: 0.355097424341939

In [94]:
```
def lstm_lags(data, order):
    X = np.zeros((len(data) - order, order, 1))
    y = data[order:]

    for i in range(len(y)):
        X[i] = data[i:i+order].values.reshape(-1, 1)

    return X, y

order = 5
X_train_LSTM, y_train_LSTM = lstm_lags(NYSE_train['log_volume'], order)
X_test_LSTM, y_test_LSTM = lstm_lags(NYSE_test['log_volume'], order)
```

In [119]:
```
def build_lstm_model(n_units, input_shape, optimizer):
    model = Sequential()
    model.add(LSTM(n_units, input_shape=input_shape))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    return model
```

In [121]:
```
y_train_LSTM_scaled = scalar.fit_transform(y_train_LSTM.values.reshape(-1, 1)).reshape(-1)
y_test_LSTM_scaled = scalar.transform(y_test_LSTM.values.reshape(-1, 1)).reshape(-1)
X_train_LSTM_scaled = scalar.fit_transform(X_train_LSTM.reshape(-1, order)).reshape(-1, order, 1
X_test_LSTM_scaled = scalar.transform(X_test_LSTM.reshape(-1, order)).reshape(-1, order, 1)

```

In [123]:
```
# Tuning the LSTM parameters and fitting the model

input_shape = (X_train_LSTM_scaled.shape[1], X_train_LSTM_scaled.shape[2])

param_grid = {
    'n_units': [10, 50, 100],
    'input_shape': [input_shape],
    'optimizer': ['adam', 'rmsprop'],
    'batch_size': [8, 16],
    'epochs': [50, 100]
}

lstm_model = KerasRegressor(build_fn=build_lstm_model, verbose=0)
n_folds = 5
search_LSTM = GridSearchCV(lstm_model, param_grid, scoring='r2', cv=n_folds, n_jobs=-1, verbose=
search_LSTM.fit(X_train_LSTM_scaled, y_train_LSTM_scaled)

best_lstm = search_LSTM.best_estimator_.model

```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

In [124]:
```
cv_r2_lstm = search_LSTM.best_score_
print(f"LSTM CV R^2 score: {cv_r2_lstm}")
```

```
LSTM CV R^2 score: 0.5397877376445614
```

In [125]:
```
r2_lstm = r2_score(y_test_LSTM_scaled, search_LSTM.best_estimator_.predict(X_test_LSTM_scaled))
print(f"LSTM R^2 test score: {r2_lstm}")
```

```
LSTM R^2 test score: 0.37767030882012886
```

In [40]:
```
rf =  RandomForestRegressor(
  n_estimators = 100,
  criterion = 'squared_error',
  max_features = 'sqrt',
  oob_score = True,
  random_state = 425
  )
```

In [41]:
```
pipe_rf = Pipeline(steps = [
  ("model", rf)
  ])
pipe_rf
```

Out[41]:
```
▸ Pipeline
▸ RandomForestRegressor
```

In [42]:
```
B_grid = [50, 100, 150, 200, 250, 300]
m_grid = ['sqrt', 'log2', 1.0]
tuned_parameters_rf = {
  "model__n_estimators": B_grid,
  "model__max_features": m_grid
  }
tuned_parameters_rf
```

Out[42]:
```
{'model__n_estimators': [50, 100, 150, 200, 250, 300],
 'model__max_features': ['sqrt', 'log2', 1.0]}
```

In [43]:
```
search_rf = GridSearchCV(
  pipe_rf,
  tuned_parameters_rf,
  cv = TimeSeriesSplit(5),
  scoring = "r2",
  refit = True
  )
```

In [44]:
```
search_rf.fit(X_train,Y_train)
```

Out[44]:
```
▸ GridSearchCV
 ▸ RandomForestRegressor
```

In [45]:
```
cv_r2_rf = search_rf.best_score_
cv_r2_rf
```

Out[45]: 0.5050974275425025

In [48]:
```
r2_rf = r2_score(Y_test, search_rf.best_estimator_.predict(X_test))
r2_rf
```

Out[48]: 0.3474070466946525

In [49]:
```
bst =  AdaBoostRegressor(
  base_estimator = DecisionTreeRegressor(max_depth = 3),
  n_estimators = 50,
  learning_rate = 1.0,
  random_state = 425
  )
```

In [50]:
```
pipe_bst = Pipeline(steps = [
  ("model", bst)
  ])
pipe_bst
```

Out[50]:

▸ **Pipeline**

▸ **model: AdaBoostRegressor**

▸ DecisionTreeRegressor

In [51]:
```
d_grid = [
  DecisionTreeRegressor(max_depth = 1),
  DecisionTreeRegressor(max_depth = 2),
  DecisionTreeRegressor(max_depth = 3),
  DecisionTreeRegressor(max_depth = 4)
  ]
B_grid = [50, 100, 150, 200, 250, 300, 350, 400]
lambda_grid = [0.2, 0.4, 0.6, 0.8, 1.0]
tuned_parameters_bst = {
  "model__base_estimator": d_grid,
  "model__n_estimators": B_grid,
  "model__learning_rate": lambda_grid
  }
tuned_parameters_bst
```

Out[51]:
```
{'model__base_estimator': [DecisionTreeRegressor(max_depth=1),
  DecisionTreeRegressor(max_depth=2),
  DecisionTreeRegressor(max_depth=3),
  DecisionTreeRegressor(max_depth=4)],
 'model__n_estimators': [50, 100, 150, 200, 250, 300, 350, 400],
 'model__learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0]}
```

In [52]:
```
search_bst = GridSearchCV(
  pipe_bst,
  tuned_parameters_bst,
  cv = TimeSeriesSplit(5),
  scoring = "r2",
  refit = True
  )
```

In [53]:
```
search_bst.fit(X_train,Y_train)
```

Out[53]:
```
GridSearchCV
  model: AdaBoostRegressor
    DecisionTreeRegressor
```

In [54]:
```
cv_r2_bst = search_bst.best_score_
cv_r2_bst
```

Out[54]: 0.5055175769773629

In [56]:
```
r2_bst = r2_score(Y_test, search_bst.best_estimator_.predict(X_test))
r2_bst
```

Out[56]: 0.33981927842268433

Model	CV R^2	Test R^2
Baseline	0.399	0.181
ENET	0.532	0.375
MLP	0.520	0.355
LSTM	0.539	0.378
Random Forest	0.505	0.347

Model	CV R^2	Test R^2
Boosting	0.506	0.340