Forecasting Project

Econ 425 Assignment 6

Name	ID
Youssef Mahmoud	905854027

Project Overview

In [166]:
```
results = pd.DataFrame(columns=['Method', 'CV R2', 'Test R2', 'RMSE
```

packages

In [1]:
```
# Load data analysis library
import pandas as pd
import numpy as np

# plotting library
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)

import warnings
warnings.filterwarnings("ignore")
```

Metrics

In [6]:
```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_
```

A) Data importation & Preparation

In [32]:
```
# Read in NYSE data from url
url = "https://raw.githubusercontent.com/ucla-econ-425t/2023winter/
s = requests.get(url).content.decode('utf-8')
df = pd.read_csv(io.StringIO(s), index_col = 0)
df
```

Out[32]:

date	day_of_week	DJ_return	log_volume	log_volatility	train
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 [35]:
```
# check for missing values
df.isnull().sum().sum()
```

Out[35]: 0

In [36]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6051 entries, 1962-12-03 to 1986-12-31
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   day_of_week     6051 non-null   object 
 1   DJ_return       6051 non-null   float64
 2   log_volume      6051 non-null   float64
 3   log_volatility  6051 non-null   float64
 4   train           6051 non-null   bool   
dtypes: bool(1), float64(3), object(1)
memory usage: 242.3+ KB
```

Creating Lagged Values

In [33]:

```python
## create lagged values
lags = range(1,6)

for L in lags:
    df['DJ_return_L%d' % L] = df['DJ_return'].shift(L)
    df['log_volume_L%d' % L] = df['log_volume'].shift(L)
    df['log_volatility_L%d' % L] = df['log_volatility'].shift(L)
```

In [34]:

```python
df
```

Out[34]:

	day_of_week	DJ_return	log_volume	log_volatility	train	DJ_return_L1	log_volume_L1	l
date								
1962-12-03	mon	-0.004461	0.032573	-13.127403	True	NaN	NaN	
1962-12-04	tues	0.007813	0.346202	-11.749305	True	-0.004461	0.032573	
1962-12-05	wed	0.003845	0.525306	-11.665609	True	0.007813	0.346202	
1962-12-06	thur	-0.003462	0.210182	-11.626772	True	0.003845	0.525306	
1962-12-07	fri	0.000568	0.044187	-11.728130	True	-0.003462	0.210182	
...	...	...	...	...	...	...	...	
1986-12-24	wed	0.006514	-0.236104	-9.807366	False	-0.006150	0.450780	
1986-12-26	fri	0.001825	-1.322425	-9.906025	False	0.006514	-0.236104	
1986-12-29	mon	-0.009515	-0.371237	-9.827660	False	0.001825	-1.322425	
1986-12-30	tues	-0.001837	-0.385638	-9.926091	False	-0.009515	-0.371237	
1986-12-31	wed	-0.006655	-0.264986	-9.935527	False	-0.001837	-0.385638	

6051 rows × 20 columns

In [35]:

```python
df.dropna(inplace = True)
```

In [100]:
```
train = df[df['train']==True]
test = df[df['train']==False]
```

In [37]:
```
print(f' the length of training dataset is {len(train)} \n the leng
print(f' the fraction of test size is {round(len(test)/len(train),2
```

```
the length of training dataset is 4276
the length of test dataset is 1770
the fraction of test size is 0.41
```

Baseline Model (Random Walk)

In [38]:
```
Baseline_forecast = test['log_volume_L1']
```

In [39]:
```
baseline_r2 = round(r2_score(test['log_volume'], Baseline_forecast)
```

In [40]:
```
baseline_RMSE = round(np.sqrt(mean_squared_error(test['log_volume']
```

In [41]:
```
def accuracy(forecasted_series):
    true = test['log_volume'].apply(lambda x: 1 if x>0 else 0)
    pred = forecasted_series.apply(lambda x: 1 if x>0 else 0)
    acc = accuracy_score(true, pred)
    return acc
```

In [42]:
```
baseline_acc = round(accuracy(Baseline_forecast),3)
```

Save baseline results

In [46]:
```
results = pd.DataFrame()
```

In [47]:
```
results = results.append({'Method':'Baseline', 'CV R2': 'NA', 'Test
               'Accuracy': baseline_acc}, ignore_index=True)
```

In [48]:
```
results
```

Out[48]:

	Method	CV R2	Test R2	RMSE	Accuracy
0	Baseline	NA	0.18	0.217	0.72

Model 1) Autoregression (AR) [Elastic Net]

we will use autogressive model of five lags with Elastic Net penalty which will combine both penalties from Lasso and Ridge which might help in identifying more stable model. Elastic-Net has two parameters to be set λ and α. α is the mixing parameter between ridge and lasso while λ is the penalty coefficient. we will be using 10-K-cross validation over a wide range of hyperparameters. Worth to note in Sckit-learn library, L1_ratio represents alpha, the mixing parameter, while alpha represents lambda, the penalty coefficient

$$\frac{\sum_{i=1}^{n}[y_i-\sum_{j=1}^{p} x_{i,j}\beta_i]^2}{2n} + \lambda((\frac{1-\alpha}{2}) \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j|)$$

1) check for stationarity

In [49]:
```
from statsmodels.tsa.stattools import adfuller
```

In [88]:
```
# ADF Test for Stationarity
adf = adfuller(train["log_volume"])[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else
```

```
p value:1.3193650889835212e-12 , Series is Stationary
```

In [89]:
```
# ADF Test for Stationarity
adf = adfuller(train["DJ_return"])[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else
```

```
p value:0.0 , Series is Stationary
```

In [90]:
```
# ADF Test for Stationarity
adf = adfuller(train["log_volatility"])[1]
print(f"p value:{adf}", ", Series is Stationary" if adf <0.05 else
```

```
p value:2.5673959372114234e-06 , Series is Stationary
```

Since all series are stationary, we can deploy autoresressive models to exploit any serieal correlation in our data to predict the trading volume

2) test for autocorrelation using box test

In [9]:
```
from statsmodels.stats.diagnostic import acorr_ljungbox as ljung
```

In [91]:
```
ljung_p = np.mean(ljung(x=train["log_volume"], lags = 5))[1].round(
print("Ljung Box, p value:", ljung_p, ", Series are uncorrelated" i
```

```
Ljung Box, p value: 0.0 , Series are correlated
```

In [92]:
```
ljung_p = np.mean(ljung(x=train["DJ_return"], lags = 5))[1].round(3
print("Ljung Box, p value:", ljung_p, ", Series are uncorrelated" i
```

```
Ljung Box, p value: 0.0 , Series are correlated
```

In [93]:
```
ljung_p = np.mean(ljung(x=train["log_volatility"], lags = 5))[1].ro
print("Ljung Box, p value:", ljung_p, ", Series are uncorrelated" i
```

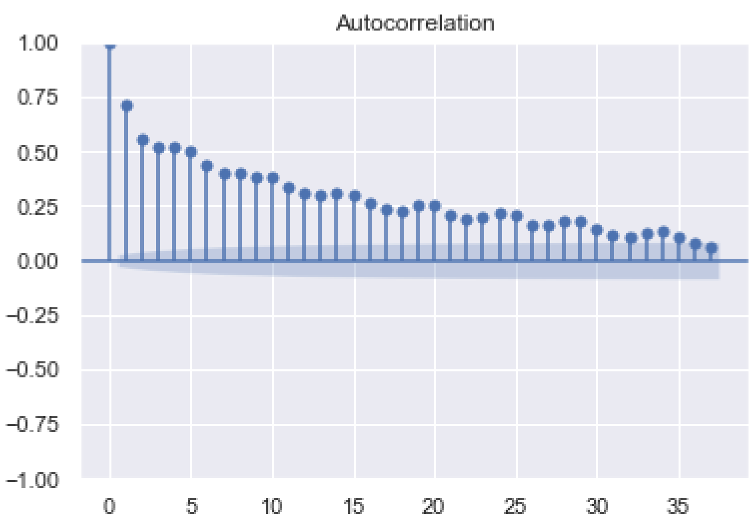
```
Ljung Box, p value: 0.0 , Series are correlated
```

from Ljung-Box statistic we rejected the null hypothesis that the autocorrelation of lagged values are equal to zero. Hence, we can exploit the linear dependency to estimate AR models

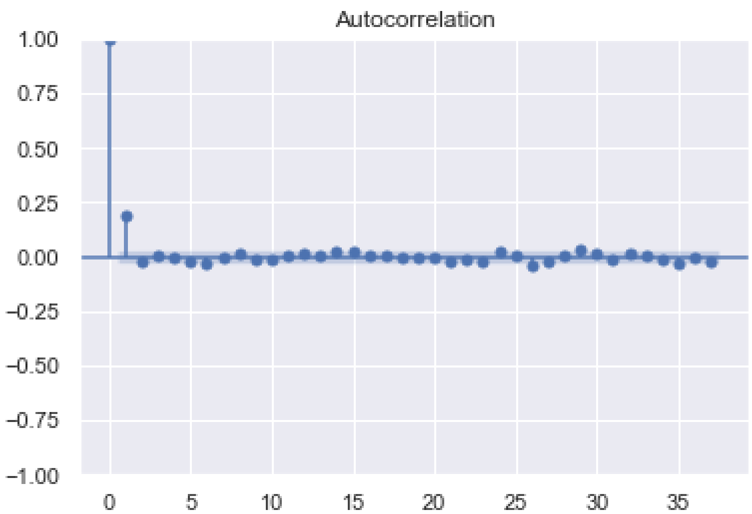
3) check autocorrelation plot

In [21]:
```
# Plot the ACF
from statsmodels.graphics.tsaplots import month_plot, seasonal_plot
```

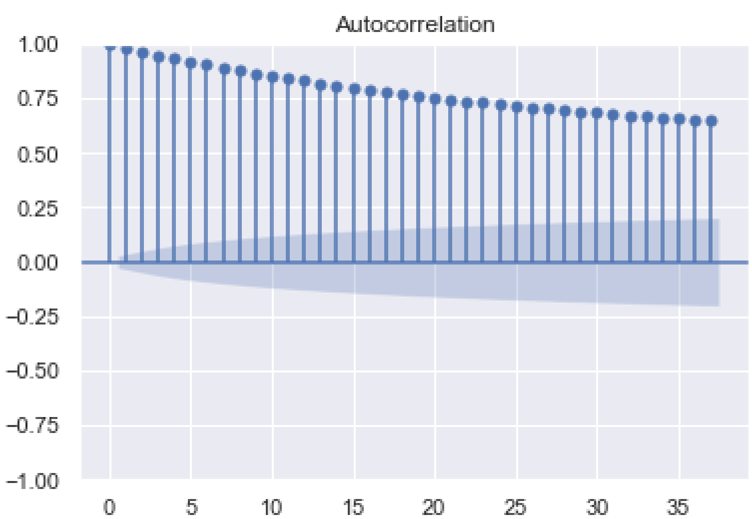
In [94]:
```
plot_acf(train["log_volume"]);
```



In [95]:
```
plot_acf(train["DJ_return"]);
```



In [96]:
```
plot_acf(train["log_volatility"]);
```



We can see that all series have storng autocorrelation with past values for all lags except for DJ_return which indicates only the first lag is statistically significant

Model building

In [50]:
```
# machine learning library
from sklearn.linear_model import ElasticNet
```

Pipeline

In [51]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

from sklearn import set_config
set_config(display='diagram')
```

In [52]:
```
pipe = Pipeline(steps = [
  ("col_tf", StandardScaler()),
  ("model", ElasticNet())
  ])
```

In [53]:
```
pipe
```

Out[53]:
```
Pipeline(steps=[('col_tf', StandardScaler()), ('model', ElasticNet
())])
```
Please rerun this cell to show the HTML repr or trust the notebook.

In [54]:
```
tuned_parameters = {'model__alpha':np.linspace(1, 200, 100), 'model
```

GridSearch

In [101]:
```
X_train = train.iloc[:,5:]
X_test = test.iloc[:,5:]
```

In [102]:
```
y_train = train[['log_volume']]
y_test = test[['log_volume']]
```

In [57]:
```
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
```

In [58]:
```
search = GridSearchCV(pipe,
                          tuned_parameters,
                          cv = TimeSeriesSplit(n_splits=5),
                          scoring = "r2")

eNet_best = search.fit(X_train, y_train)
```

In [59]:
```
eNet_best.best_estimator_
```

Out[59]:
```
Pipeline(steps=[('col_tf', StandardScaler()),
                ('model', ElasticNet(l1_ratio=0.01))])
```
Please rerun this cell to show the HTML repr or trust the notebook.

CV Results

In [60]:
```
# CV-R2
eNet_CV = round(abs(eNet_best.best_score_),3)
```

In [61]:
```
eNet_CV
```

Out[61]: 0.462

Out-of Sample Results

In [62]:
```
pred = eNet_best.predict(X_test)
```

In [63]:
```
pred = pd.Series(pred)
```

In [64]:
```
eNet_RMSE = round(np.sqrt(mean_squared_error(y_test, pred)),3)
```

In [65]:
```
eNet_r2 = round(r2_score(test['log_volume'], pred),3)
```

In [66]:
```
eNet_acc = round(accuracy(pred),3)
```

Save resutls for AR (Elastic Net)

In [67]:
```
results = results.append({'Method':'AR-eNet', 'CV R2':eNet_CV , 'Te
                    'Accuracy': eNet_acc}, ignore_index=True)
```

In [68]:
```
results
```

Out[68]:

	Method	CV R2	Test R2	RMSE	Accuracy
0	Baseline	NA	0.180	0.217	0.720
1	AR-eNet	0.462	0.336	0.195	0.712

Model 2) Autoregression (AR) [MLP]

In [69]:
```
from sklearn.neural_network import MLPRegressor
```

Pipeline

In [70]:
```
pipe = Pipeline(steps = [
  ("col_tf", StandardScaler()),
  ("model", MLPRegressor())
  ])
```

In [71]:
```
pipe
```

Out[71]:
```
Pipeline(steps=[('col_tf', StandardScaler()), ('model', MLPRegressor
())])
```

Please rerun this cell to show the HTML repr or trust the notebook.

In [72]:
```
tuned_parameters = {'model__hidden_layer_sizes': [(10,),(5,20,), (1
                    'model__activation': ['relu', 'tanh', 'logistic
                    'model__alpha': [0.001, 0.01]}
```

GridSearch

In [73]:
```
search = GridSearchCV(pipe,
                          tuned_parameters,
                          cv = TimeSeriesSplit(n_splits=5),
                          scoring = "r2")
```

In [74]:
```
MLP_best = search.fit(X_train, y_train)
```

In [75]:
```
MLP_best.best_estimator_
```

Out[75]:
```
Pipeline(steps=[('col_tf', StandardScaler()),
                ('model',
                 MLPRegressor(activation='logistic', alpha=0.001,
                              hidden_layer_sizes=(10, 50)))])
```

Please rerun this cell to show the HTML repr or trust the notebook.

CV Results

In [76]:
```
# CV-R2
MLP_CV = round(abs(MLP_best.best_score_),3)
```

In [77]:
```
MLP_CV
```

Out[77]: 0.518

Out-of Sample Results

In [78]:
```
pred = MLP_best.predict(X_test)
pred = pd.Series(pred)
```

In [79]:
```
MLP_RMSE = round(np.sqrt(mean_squared_error(y_test, pred)),3)
MLP_r2 = round(r2_score(test['log_volume'], pred),3)
MLP_acc = round(accuracy(pred),3)
```

save results

In [80]:
```
results = results.append({'Method':'AR-MLP', 'CV R2':MLP_CV , 'Test
                'Accuracy': MLP_acc}, ignore_index=True)
```

In [81]:
```
results
```

Out[81]:

	Method	CV R2	Test R2	RMSE	Accuracy
0	Baseline	NA	0.180	0.217	0.720
1	AR-eNet	0.462	0.336	0.195	0.712
2	AR-MLP	0.518	0.404	0.185	0.733

In [82]:
```
results.to_csv('forecasting_results.csv')
```

In [83]:
```
results = pd.read_csv('forecasting_results.csv')
```

In [84]:
```
results.drop('Unnamed: 0', axis =1, inplace = True)
```

In [85]:
```
results
```

Out[85]:

	Method	CV R2	Test R2	RMSE	Accuracy
0	Baseline	NaN	0.180	0.217	0.720
1	AR-eNet	0.462	0.336	0.195	0.712
2	AR-MLP	0.518	0.404	0.185	0.733

Model 3) LSTM

In [86]:
```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.wrappers.scikit_learn import KerasRegressor
```

In [116]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
```

Building RNN Architecture

In [91]:
```
X_train.iloc[:,:3].head()
```

Out[91]:

	DJ_return_L1	log_volume_L1	log_volatility_L1
date			
1962-12-10	0.000568	0.044187	-11.728130
1962-12-11	-0.010824	0.133246	-10.872526
1962-12-12	0.000124	-0.011528	-10.977797
1962-12-13	0.003358	0.001607	-11.012360
1962-12-14	-0.003296	-0.106437	-11.047108

In [103]:
```
train_X = np.array(X_train.iloc[:,:3])
```

In [104]:
```
train_X[0:5,:]
```

Out[104]:
```
array([[ 5.68000000e-04,  4.41870000e-02, -1.17281302e+01],
       [-1.08240000e-02,  1.33246000e-01, -1.08725263e+01],
       [ 1.24000000e-04, -1.15280000e-02, -1.09777968e+01],
       [ 3.35800000e-03,  1.60700000e-03, -1.10123599e+01],
       [-3.29600000e-03, -1.06437000e-01, -1.10471081e+01]])
```

In [105]:
```
train_y = np.array(y_train)
```

In [106]:
```
train_y
```

Out[106]:
```
array([[ 0.133246],
       [-0.011528],
       [ 0.001607],
       ...,
       [-0.137014],
       [-0.041932],
       [-0.125945]])
```

In [107]:
```
x_train = []
y_train = []

###############################################################
####Pick your input size and edit to make binary forecast####
###############################################################
input_size = 5

for i in range(input_size, len(train_X)):
    x_train.append(train_X[i-input_size:i, :])
    y_train.append(train_y[i, 0])
```

In [114]:
```
x_train.shape
```

Out[114]:
```
(4271, 5, 3)
```

In [115]:
```
y_train.shape
```

Out[115]:
```
(4271,)
```

Creating LSTM Model

In [ ]:

```
# Define the Keras model
###Edit here to create your optimizer
def create_model():
    model = Sequential()
    model.add(Dense(10, input_dim=60, activation='LSTM'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='mean_squared_error', optimizer='adam', metr
    return(model)

# Wrap the Keras model in a scikit-learn compatible estimator
model = KerasRegressor(build_fn=create_model, verbose=0)

# Define the hyperparameters to search over
####EXAMPLE###
param_grid = {'batch_size': [10, 20, 100], 'epochs': [10, 100], 'ne

```

In [ ]:

```
# Perform the grid search over the hyperparameters
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=
grid_result = grid.fit(x_train, y_train)
```

In [ ]:

```
# Print the results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.b
```