Brian Ferrell
ECON 641

Appendix

A. Loss function and using it for example 1

```
def brianLoss(current_price, ten_day_price, forecast_price):
   d1 = None
   d2 = None
   if forecast_price >= ((current_price * .01) + current_price):
       d1 = 1
   elif forecast_price <= ((current_price * .01) + current_price):
       d1 = 0
   if ten_day_price > current_price:
       d2 = 1
   elif ten_day_price < current_price:
       d2 = 0

   loss = (-d1 * ((ten_day_price - current_price) * np.floor(1000 / current_price))) \
          + (1 - d1) * (d2 * ((ten_day_price - current_price) * np.floor(1000 /
current_price)) - .50)
   return round(loss,2)

brianLoss(current_price=181,ten_day_price=184,forecast_price=183)
-15.0
```

B. ADF Implementation

```
# CDW - Trend
adf1summary = ADF(df['CDW'], lags=2, trend='ct', method='t-stat').summary()
adf1summary

# XLK - Trend
adf2summary = ADF(df['XLK'], lags=4, trend='ct', method='t-stat').summary()
adf2summary

# SP500 - Trend
adf3summary = ADF(df['SP500'], lags=8, trend='ct', method='t-stat').summary()
adf3summary

# CDW - No trend
adf1 = ADF(df['CDW'].diff().dropna(), lags=0, trend='n',
method='t-stat').regression
adf1.summary()
# XLK - No trend
adf2 = ADF(df['XLK'].diff().dropna(), lags=0, trend='n',
method='t-stat').regression
adf2.summary()
```

```
# SP500 - No trend
adf3 = ADF(df['SP500'].diff().dropna(), lags=0, trend='n',
method='t-stat').regression
adf3.summary()
```

C. Augmented Dickey-Fuller Test Results for CDW (without transformation):

Test Statistic	-2.512
P-value	0.322
Lags	2

Critical Values: -4.01 (1%), -3.44 (5%), -3.14 (10%)

The 5% critical value is -3.44, p-value at .322, and the test statistic is -2.51, which means we are inside the critical value zone.

D. Augmented Dickey-Fuller Test Results for XLK (without transformation):

Test Statistic	-0.909
P-value	0.955
Lags	4

Critical Values: -4.01 (1%), -3.44 (5%), -3.14 (10%)

The 5% critical value is -3.44, p-value at .955, and the test statistic is -.909, which means we are inside the critical value zone.

E. Augmented Dickey-Fuller Test Results for SP500 (without transformation):

Test Statistic	-1.063
P-value	0.935
Lags	8

Critical Values: -4.01 (1%), -3.44 (5%), -3.14 (10%)

The 5% critical value is -3.44, p-value at .935, and the test statistic is -1.063, which means we are inside the critical value zone.

F. Augmented Dickey-Fuller Test Results for CDW (transformation):

Test Statistic	-13.647
P-value	9.327775150638277e-25
Lags	0

Critical Values: -2.58 (1%), -1.94 (5%), -1.62 (10%)

The 5% critical value is -1.94, p-value extremely small, and the test statistic is -13.647, which means we are outside the critical value zone.

G. Augmented Dickey-Fuller Test Results for XLK (transformation):

Test Statistic	-13.041
P-value	9.21331593333097e-24
Lags	0

Critical Values: -2.58 (1%), -1.94 (5%), -1.62 (10%)

The 5% critical value is -1.94, p-value extremely small, and the test statistic is -13.041, which means we are outside the critical value zone.

H. Augmented Dickey-Fuller Test Results for SP500 (transformation):

Test Statistic	-12.765
P-value	2.775214831077567e-23

Lags	0

Critical Values: -2.58 (1%), -1.94 (5%), -1.62 (10%)

The 5% critical value is -1.94, p-value extremely small, and the test statistic is -12.765, which means we are outside the critical value zone.

I. P-value matrix implementation and causality tests

```
from statsmodels.tsa.stattools import grangercausalitytests
maxlag=12
test = 'ssr_chi2test'
def grangers_causation_matrix(data, variables, test='ssr_chi2test', verbose=False):
   df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables,
index=variables)
   for c in df.columns:
       for r in df.index:
           test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag,
verbose=False)
           p_values = [round(test_result[i+1][0][test][1],4) for i in
range(maxlag)]
           if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
           min_p_value = np.min(p_values)
           df.loc[r, c] = min_p_value
   df.columns = [var + '_x' for var in variables]
   df.index = [var + '_y' for var in variables]
   return df
new_df = df.drop(columns=['datetime'])
grangers_causation_matrix(new_df.diff().dropna(), variables = new_df.columns)
```

```
grangercausalitytests(new_df.diff().dropna()[['CDW', 'SP500']], maxlag=[12])
# Output:
Granger Causality
ssr based F test:         F=1.4980  , p=0.1314  , df_denom=142, df_num=12
ssr based chi2 test:   chi2=21.1410 , p=0.0484  , df=12
likelihood ratio test: chi2=19.9060 , p=0.0689  , df=12
parameter F test:         F=1.4980  , p=0.1314  , df_denom=142, df_num=12
```

```
grangercausalitytests(new_df.diff().dropna()[['CDW', 'XLK']], maxlag=[12])
# Output:
Granger Causality
ssr based F test:         F=1.4987  , p=0.1311  , df_denom=142, df_num=12
ssr based chi2 test:   chi2=21.1509 , p=0.0482  , df=12
likelihood ratio test: chi2=19.9148 , p=0.0687  , df=12
parameter F test:         F=1.4987  , p=0.1311  , df_denom=142, df_num=12
```

J. ARMA model implementation

```
pdq = ((0, 1, 0), (0, 1, 2),
      (0, 1, 3),(1, 1, 6),(1, 1, 7),
      (2, 1, 7), (4, 1, 3),(4, 1, 4))

resids = []
max_abs_roots = []
predictions_list = []
confidence_intervals = []

for param in pdq:
   try:
       model_arima = sm.tsa.arima.ARIMA(df['CDW'], order=param)
       model_arima_fit = model_arima.fit()
       resids.append(model_arima_fit.resid)
       predictions_list.append((model_arima_fit.forecast(steps=10)))

       result_ci = model_arima_fit.get_forecast(steps=10)
       result_ci = result_ci.conf_int(alpha=.05)
       confidence_intervals.append(result_ci.values[-1])


       max_abs_roots.append(max(abs(model_arima_fit.arroots)))
   except:
       continue
```

K. DM Test

```
dm = []
real_values = [177.990005, 169.100006, 171.300003, 177.529999,
              172.460007, 169.410004, 174.020004, 174.350006,
              171.570007, 165.44]
for param in pdq:
   try:
       model_arima = sm.tsa.arima.ARIMA(df['CDW'], order=param)
       model_arima_fit = model_arima.fit()

dm.append(dm_test(real_values,predictions_list[0],model_arima_fit.forecast(steps=10
)))
   except:
       continue


def autocovariance(Xi, N, k, Xs):
   autoCov = 0
   T = float(N)
   for i in np.arange(0, N - k):
```

```
        autoCov += ((Xi[i + k]) - Xs) * (Xi[i] - Xs)
    return (1 / (T)) * autoCov

gamma = []
for lag in range(0, h):
    gamma.append(autocovariance(d_lst, len(d_lst), lag, mean_d))  # 0, 1, 2
V_d = (gamma[0] + 2 * sum(gamma[1:])) / T
DM_stat = V_d ** (-0.5) * mean_d
harvey_adj = ((T + 1 - 2 * h + h * (h - 1) / T) / T) ** (0.5)
DM_stat = harvey_adj * DM_stat
DM_stat = np.round(DM_stat, 2)
# Find p-value
p_value = 2 * t.cdf(-abs(DM_stat), df=T - 1)
p_value = np.round(p_value, 4)

dm_return = collections.namedtuple('dm_return', 'DM p_value')

rt = dm_return(DM=np.round(DM_stat, 2), p_value=p_value)
```

L. Q-Stat with p-value

```
qstat = []
pvalue = []
for i in range(len(resids)):
    qstat.append(sm.stats.acorr_ljungbox(resids[i].values[1:],
return_df=False,lags=[10])[0])
    pvalue.append(sm.stats.acorr_ljungbox(resids[i].values[1:],
return_df=False,lags=[10])[1])
```

M. VAR Model Implementation

```
'''VAR model'''
'''Select number of lags'''
# correlation
df.corr()
from statsmodels.tsa.api import VAR
for i in [0,1,2,3,4,5,6,7,8,9,10]:
    model = VAR(df[['CDW', 'SP500', 'XLK']].diff().dropna())
    var_result = model.fit(i)
    print('Order =', i)
# For sake of VAR I did 1 lag even though it is white noise model
var_result = model.fit(1)
var_result.summary()

lag_order = var_result.k_ar
print(lag_order)
```

```
# Input data for forecasting
forecast_input = df[['CDW', 'SP500', 'XLK']].diff().values[-lag_order:]
forecast_input

# Forecast
pred = var_result.forecast(y=forecast_input, steps=10)
pred = pd.DataFrame(pred, index=df.index[-10:], columns=df[['CDW', 'SP500',
'XLK']].columns + '_1d')
pred
```

N. Reverse transformation for VAR

```
def invert_transformation(df_train, df_forecast):
   df_fc = df_forecast.copy()
   columns = df_train.columns
   for col in columns:
       # Roll back 1st Diff
       df_fc[str(col)+'_forecast'] = df_train[col].iloc[-1] +
df_fc[str(col)+'_1d'].cumsum()
   return df_fc

df_results = invert_transformation(df[['CDW', 'SP500', 'XLK']], pred)
```

O. Plots (no transformation)

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

plt.rcParams.update({'figure.figsize':(12,12)})

# CDW
fig, axes = plt.subplots(3, 3)
axes[0, 0].plot(df['CDW']); axes[0, 0].set_title('CDW Closing Prices')
plot_acf(df['CDW'], ax=axes[0, 1])
plot_pacf(df['CDW'], ax=axes[0, 2])

# XLK
axes[1, 0].plot(df['XLK']); axes[1, 0].set_title('XLK Closing Prices')
plot_acf(df['XLK'], ax=axes[1, 1])
plot_pacf(df['XLK'], ax=axes[1, 2])

# SP500
axes[2, 0].plot(df['SP500']); axes[2, 0].set_title('SP500 Closing Prices')
plot_acf(df['SP500'], ax=axes[2, 1])
plot_pacf(df['SP500'], ax=axes[2, 2])
```

```
plt.savefig("Final_Station_Obser.png")
plt.show()
```

P. Plots (transformation)

```
plt.rcParams.update({'figure.figsize':(12,12)})

# CDW
fig, axes = plt.subplots(3, 3)
axes[0, 0].plot(df['CDW'].diff().dropna()); axes[0, 0].set_title('CDW Closing
Prices')
plot_acf(df['CDW'].diff().dropna(), ax=axes[0, 1])
plot_pacf(df['CDW'].diff().dropna(), ax=axes[0, 2])

# XLK closing
axes[1, 0].plot(df['XLK'].diff().dropna()); axes[1, 0].set_title('XLK Closing
Prices')
plot_acf(df['XLK'].diff().dropna(), ax=axes[1, 1])
plot_pacf(df['XLK'].diff().dropna(), ax=axes[1, 2])

# SP500
axes[2, 0].plot(df['SP500'].diff().dropna()); axes[2, 0].set_title('SP500 Closing
Prices')
plot_acf(df['SP500'].diff().dropna(), ax=axes[2, 1])
plot_pacf(df['SP500'].diff().dropna(), ax=axes[2, 2])
```

Q. Residual plots

```
plt.rcParams.update({'figure.figsize':(12,9)})
plt.plot(resids[0][1:])
plt.title("Best ARIMA Residuals Plot")
plt.xlabel("t")
plt.ylabel("Residuals")
plt.savefig("best_arma_residual_values")
```

R. Summary of VAR

```
Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
---------------------------------------------------------------------
No. of Equations:         3.00000
Nobs:                     178.000
---------------------------------------------------------------------
Results for equation CDW
=============================================================================
             coefficient       std. error           t-stat            prob
-----------------------------------------------------------------------------
```

```
const           0.085304         0.210615           0.405           0.685
L1.CDW         -0.078211         0.096288          -0.812           0.417
L1.SP500       -0.004312         0.015062          -0.286           0.775
L1.XLK          0.236676         0.276397           0.856           0.392
================================================================================

Results for equation SP500
================================================================================
             coefficient       std. error           t-stat            prob
--------------------------------------------------------------------------------
const           0.663833         3.009120           0.221           0.825
L1.CDW         -1.638100         1.375694          -1.191           0.234
L1.SP500        0.026386         0.215189           0.123           0.902
L1.XLK          1.778966         3.948966           0.450           0.652
================================================================================

Results for equation XLK
================================================================================
             coefficient       std. error           t-stat            prob
--------------------------------------------------------------------------------
const           0.076421         0.152427           0.501           0.616
L1.CDW         -0.073840         0.069686          -1.060           0.289
L1.SP500        0.002141         0.010900           0.196           0.844
L1.XLK          0.031097         0.200035           0.155           0.876
================================================================================
```