Q1)

Importing packages and the dataset

Q1

```
]: #Importing necessary packages
   import pandas as pd
   import numpy as np
   from datetime import datetime
```

```
]: #Importing dataset

   df_Energy=pd.read_excel("/Users/metuhead/Desktop/FE 520-Python/Assignments/HW5/Assignment5_data/Energy.xlsx")
```

```
]: #Showing dataset
   df_Energy
```

Creating spit function to split the dataset into test and train dataset.
I only selected columns from ”Accumulated Other Comprehensive Income (Loss)” to column ”Selling, General and Administrative Expenses”.

```
#Creating a function to split the dataset

def split(EndYear,StartYear=2012):

    import pandas as pd
    import numpy as np
    from datetime import datetime

    #Importing dataset
    df_Energy=pd.read_excel("/Users/metuhead/Desktop/FE 520-Python/Assignments/HW5/Assignment5_data/Energy.xlsx")



    # Only selecting columns form "Accumulated Other Comprehensive Income (Loss)" to
    # "Selling, General and Administrative Expenses"]

    df_Selected=df_Energy.loc[:,"Accumulated Other Comprehensive Income (Loss)": "Selling, General and Administrative E
    #df_Selected
    # Creating a data frame only includes  columns
    df_Dates=df_Energy.loc[:,"Data Date":"Fiscal Year"]
    #df_Dates

    # Joining df_Selected and df_Dates
    df_new=df_Dates.join(df_Selected)
    #df_new
```

```
if EndYear==None:

    Test=df_new[df_new["Fiscal Year"]==StartYear]

    Train=df_new[df_new["Fiscal Year"]!=StartYear]

elif EndYear != None:

    Test= df_new[(df_new["Fiscal Year"] >= StartYear) & (df_new["Fiscal Year"] <= EndYear)]

    Train=df_new[~((df_new["Fiscal Year"] >= StartYear) & (df_new["Fiscal Year"] <= EndYear))]



return(Test, Train)
```

Results shown below

Test data end Date with 2013

```
split(2013)
```

	Data Date	Fiscal Year	Accumulated Other Comprehensive Income (Loss)	Current Assets - Other - Total	Current Assets - Total	Other Long-term Assets	Non-Current Assets - Total	Assets Netting & Other Adjustments	Accum Other Comp Inc - Derivatives Unrealized Gain/Loss	Accum Other Comp Inc - Other Adjustments	...	Implied Option EPS Diluted	Implied Option EPS Diluted Preliminary	Implied Option EPS Basic 12MM	Implied Option 12MM EPS Basic Preliminary
0	20100331	2010	-1749.0	1502.0	8780.0	2568.0	21169.0	-299.0	-1312.0	-33.0	...	0.0	0.0	0.0	0.0
1	20100630	2010	-1603.0	1434.0	8296.0	2587.0	21204.0	-254.0	-1058.0	-148.0	...	0.0	0.0	0.0	0.0
2	20100930	2010	-1377.0	865.0	8839.0	2742.0	24646.0	-228.0	-962.0	-27.0	...	0.0	0.0	0.0	0.0
3	20101231	2010	-1159.0	1002.0	8780.0	2638.0	26616.0	-517.0	-786.0	11.0	...	0.0	0.0	0.0	0.0
4	20110331	2011	-873.0	759.0	9436.0	2602.0	27201.0	-789.0	-688.0	192.0	...	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
839	20151231	2015	-318.0	183.0	9471.0	119.0	33644.0	-62.0	4.0	0.0	...	NaN	NaN	NaN	NaN
840	20160331	2016	-318.0	204.0	8097.0	886.0	33661.0	-67.0	4.0	0.0	...	NaN	NaN	NaN	NaN
841	20160630	2016	-321.0	142.0	10304.0	875.0	33829.0	-146.0	4.0	0.0	...	NaN	NaN	NaN	NaN
842	20160930	2016	-327.0	176.0	9545.0	848.0	33748.0	-278.0	4.0	0.0	...	NaN	NaN	NaN	NaN
843	20161231	2016	-234.0	236.0	10401.0	107.0	34012.0	-688.0	4.0	0.0	...	NaN	NaN	NaN	NaN

604 rows × 362 columns

Train data end Date with 2013

```
split(2013)
```

	Data Date	Fiscal Year	Accumulated Other Comprehensive Income (Loss)	Current Assets - Other - Total	Current Assets - Total	Other Long-term Assets	Non-Current Assets - Total	Assets Netting & Other Adjustments	Accum Other Comp Inc - Derivatives Unrealized Gain/Loss	Accum Other Comp Inc - Other Adjustments	...	Implied Option EPS Diluted	Implied Option EPS Diluted Preliminary	Implied Option EPS Basic 12MM	Implied Option 12MM EPS Basic Preliminary
8	20120331	2012	-1057.0	1421.0	8212.0	3675.0	32435.0	-350.0	-547.0	107.0	...	0.0	0.0	0.0	0.0
9	20120630	2012	-779.0	1699.0	7925.0	3417.0	32689.0	-305.0	78.0	-270.0	...	0.0	0.0	0.0	0.0
10	20120930	2012	-675.0	2167.0	8157.0	3302.0	34055.0	-186.0	-152.0	67.0	...	0.0	0.0	0.0	0.0
11	20121231	2012	-493.0	2148.0	8387.0	3596.0	35054.0	-227.0	-24.0	170.0	...	0.0	0.0	0.0	0.0
12	20130331	2013	-452.0	8448.0	11418.0	3104.0	30961.0	-205.0	-11.0	28.0	...	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
827	20121231	2012	-464.0	110.0	13029.0	254.0	14194.0	84.0	0.0	0.0	...	0.0	NaN	NaN	NaN
828	20130331	2013	-382.0	172.0	15297.0	320.0	15476.0	138.0	4.0	0.0	...	0.0	NaN	NaN	NaN
829	20130630	2013	-254.0	190.0	14252.0	328.0	15443.0	-30.0	4.0	0.0	...	0.0	NaN	NaN	NaN
830	20130930	2013	-205.0	206.0	13531.0	320.0	15472.0	-62.0	4.0	0.0	...	0.0	NaN	NaN	NaN
831	20131231	2013	-204.0	197.0	12737.0	324.0	15648.0	-21.0	4.0	0.0	...	0.0	NaN	NaN	NaN

240 rows × 362 columns

Q2)

Importing packages and the dataset.
Creating a time period for date index only including business days

Q2

```
#Importing necessary packages
import pandas as pd
import numpy as np
from pandas import DataFrame,Series
import datetime
```

```
#Importing data
df=pd.read_csv("/Users/metuhead/Desktop/FE 520-Python/Assignments/HW5/Assignment5_data/HW5_Q2_data.csv")
```

```
#Creating a time period for date index only including business days
period=pd.bdate_range(start='1/3/1990',end='11/1/1993',freq='B')

num=0
for date in period:
    if str(date)[0:-9] != str(df["Date"][num]):
        holiday=[str(date)[0:-9]]
        break
    num = num + 1
```

Creating a new time period (first cell)
Modifying the data (second cell);
Filling na with 0 value and replacing all zeros with NaN, dropping columns whose values include more than 50% na
Resampling the data by BM and getting the mean of the resampled data
Taking average of each row, and calculating monthly return

```
#Creating  a new time period
df_new= pd.bdate_range(start='1/3/1990', end='11/1/1993', holidays=holiday, freq='C', weekmask='Mon Tue Wed Thu Fri')

#Excluding the column  date
df_data=df.drop(columns='Date')

#Adding the column date which is created previously
df_data["dates"]=df_new
```

```
##Modifiying the data
#Filling na with 0 value and replacing all zeros with NaN
fil_data=df_data.fillna(0)
fil_data=fil_data.replace({0:np.nan})

#Dropping colums whose values include more than 50% na
fil_data=fil_data.dropna(axis=1,thresh=fil_data.shape[1]*0.5)

#Resampling the data by BM and getting the mean of the resampled data
df_monthly=fil_data.resample("BM",on="dates").mean()
#Taking average of each row, and calculating montly return
df_monthly["average"]=df_monthly.mean(axis=1)
```

Finding the Mean Reversion and Momentum periods in data frame

```
#Finding the Mean Reversion and Momentum periods in data frame
hucre=["start"]
x=1
for difference in df_monthly["difference"]:
    if(df_monthly["difference"][x+1]>0 and df_monthly["difference"][x]>0):
        hucre.append("mmt")
    elif(df_monthly["difference"][x]>0 and df_monthly["difference"][x-1]>0):
         hucre.append("mmt")
    elif(df_monthly["difference"][x+1]<0 and df_monthly["difference"][x]<0):
        hucre.append("mmt")
    elif(df_monthly["difference"][x]<0 and df_monthly["difference"][x-1]<0):
        hucre.append("mmt")
    else:
        hucre.append("Reversion")

    x=x+1

    if x== len(df_monthly["difference"])-1:

        if(df_monthly["difference"][x]>0 and df_monthly["difference"][x-1]>0):
            hucre.append("mmt")
        elif(df_monthly["difference"][x]<0 and df_monthly["difference"][x-1]<0):
            hucre.append("mmt")
        else:
            hucre.append("Reversion")

        break
```

Changes in returns are added to the hucre
Finding the strategy change in months
Changes in returns are added to the df_monthly

```
#Changes in returns are added to the hucre
df_monthly["strat"] = hucre

#Finding the strategy change in months
now=1

logchange=["start"]

for stra in df_monthly["strat"]:
    if str(df_monthly["strat"][now]) != str(df_monthly["strat"][now-1]):
        alterstring="difference "+df_monthly["strat"][now-1]+" to"+df_monthly["strat"][now];
        logchange.append(alterstring);
    else:

        logchange.append("same")

    now=now+1

    if now==len(df_monthly["strat"]):
        break

#Changes in returns are added to the df_monthly
df_monthly["newstrat"]=logchange

#To easily see the newstrat
strat_rever=df_monthly["newstrat"][::-1]
```

Months which the returns shifted from exhibiting mean reversion to exhibiting momentum, or from momentum to mean reversion. Results are show below

```
#Q2.1
plc=0
num=0

for i in strat_rever:

    if str(strat_rever[plc]) != "same":

        print(df_monthly["newstrat"].iloc[[len(df_monthly["strat"])-plc-1]])
        num=num+1

    plc=plc+1

    if num==2:
        break
```

```
dates
1993-09-30    difference Reversion tommt
Freq: BM, Name: newstrat, dtype: object
dates
1993-08-31    difference mmt toReversion
Freq: BM, Name: newstrat, dtype: object
```

Time periods where stock returns had momentum property, and the average momentum. Results shown below

```
#Q2.2
summ=0
num=0
plc=0

for stra in df_monthly["strat"]:

    if str(stra)=="mmt":

        summ=summ+df_monthly["average"][plc]

        num=num+1

    plc=plc+1


print("Momentum strategy return summation", summ)

print("Momentum strategy months", num)

print("Momentum strategy average return", summ*1/num)
```

```
Momentum strategy return summation 23.925597968894902
Momentum strategy months 35
Momentum strategy average return 0.6835885133969972
```

Time periods where stock returns had reversionproperty, and the average mean reversion . Results shown below

```
#Q2.3
summ=0
num=0
plc=0
for stra in df_monthly["strat"]:

    if str(stra)=="Reversion":

        summ=summ+df_monthly["average"][plc]

        num=num+1

    plc=plc+1


print("Mean Reversion return summation", summ)
print("Mean Reversion strategy months", num)
print("Mean Reversion strategy average return", summ*1/num)
```

```
Mean Reversion return summation 7.480196790139958
Mean Reversion strategy months 11
Mean Reversion strategy average return 0.6800178900127235
```

Q3)

Importing packages and the dataset

Q3

```
import sklearn
from sklearn import datasets
from sklearn import svm
from sklearn import linear_model
from sklearn import preprocessing
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import linear_model
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
```

```
#Q3.1
#Load the data
diabets_df=datasets.load_diabetes()
```

Q3.1)

Loading the dataset

```
#Q3.1
#Load the data
diabets_df=datasets.load_diabetes()
```

```
#Show the data
diabets_df
```

Q3.2)

Randomly splitting the data into training set (80%) and testing set (20%).

```
#Q3.2

#Randomly split the data into training set (80%) and testing set (20%).
new_df= diabets_df.data[:,np.newaxis,2]

varx_test=new_df[-20:]
varx_train=new_df[:-20]

vary_test=diabets_df.target[-20:]
vary_train=diabets_df.target[:-20]
```

Q3.3)

Creating a linear regression model,showing coefficients and Finding residuals and r square
Result are shown below

```
# Q3.3
# Creating  a linear regression model
lm=linear_model.LinearRegression()
lm.fit(varx_train,vary_train)

model_y= lm.predict(varx_test)
model_y
```

```
# Showing coefficients and Finding residuals and r square
coef=lm.coef_

ms_error=mean_squared_error(vary_test,model_y)

r_square=r2_score(vary_test,model_y)

print(coef)
print(ms_error)
print(r_square)
```

```
[938.23786125]
2548.0723987259707
0.47257544798227125
```

Q3.4)

Fitting and validating linear regression models on the whole data set.

```
#Q3.4
#Fitting and validating  linear regression models on the whole data set.

fold=10
new_lm=lm.fit(varx_train,vary_train)
rang=list(range(1,11))

for results in rang:

    result=cross_val_score(new_lm,varx_train, vary_train,cv=10)
    print(result)
```

```
[0.28527158 0.05133253 0.24112318 0.44536036 0.23478297 0.38300126
 0.43946172 0.25480749 0.32934734 0.31765983]
[0.28527158 0.05133253 0.24112318 0.44536036 0.23478297 0.38300126
 0.43946172 0.25480749 0.32934734 0.31765983]
[0.28527158 0.05133253 0.24112318 0.44536036 0.23478297 0.38300126
 0.43946172 0.25480749 0.32934734 0.31765983]
[0.28527158 0.05133253 0.24112318 0.44536036 0.23478297 0.38300126
 0.43946172 0.25480749 0.32934734 0.31765983]
[0.28527158 0.05133253 0.24112318 0.44536036 0.23478297 0.38300126
 0.43946172 0.25480749 0.32934734 0.31765983]
```