In [3]:
```
# Import the `pandas` library as `pd`
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load in the data with `read_csv()`
data = pd.read_csv(r'C:\Users\ankur\OneDrive\Desktop\Machine Learning\Group Pr
oject\MLF_GP2_EconCycle.csv')
data.head()

#data.info()
```

Out[3]:

	T1Y Index	T2Y Index	T3Y Index	T5Y Index	T7Y Index	T10Y Index	CP1M	CP3M	CP6M	CP1M_T1Y	CP3M_T1Y	CP6I
0	10.41	9.86	9.50	9.20	9.14	9.10	9.75	9.95	10.01	0.936599	0.955812	0.9
1	10.24	9.72	9.29	9.13	9.11	9.10	9.74	9.90	9.96	0.951172	0.966797	0.9
2	10.25	9.79	9.38	9.20	9.15	9.12	9.72	9.85	9.87	0.948293	0.960976	0.9
3	10.12	9.78	9.43	9.25	9.21	9.18	9.86	9.95	9.98	0.974308	0.983202	0.9
4	10.12	9.78	9.42	9.24	9.23	9.25	9.77	9.76	9.71	0.965415	0.964427	0.9

In [55]:
```
data.describe()
```

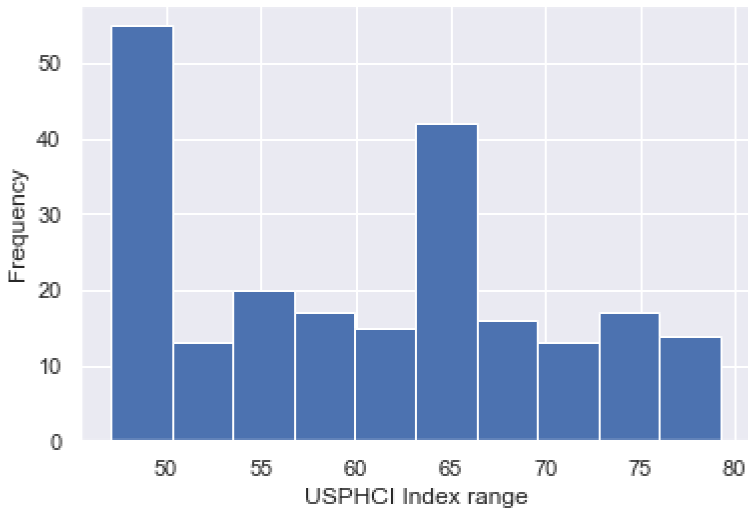
Out[55]:

	T1Y Index	T2Y Index	T3Y Index	T5Y Index	T7Y Index	T10Y Index	CP1M	
count	222.000000	222.000000	222.000000	222.000000	222.000000	222.000000	222.000000	222
mean	8.011081	8.395360	8.551802	8.799099	8.971577	9.066396	7.917838	7
std	3.152041	2.952225	2.821269	2.649868	2.545475	2.450753	3.393263	3
min	3.180000	3.840000	4.170000	4.710000	5.050000	5.330000	3.110000	3
25%	5.727500	6.180000	6.410000	6.692500	6.962500	7.172500	5.602500	5
50%	7.670000	7.990000	8.130000	8.325000	8.500000	8.600000	7.725000	7
75%	9.817500	10.027500	10.235000	10.410000	10.552500	10.677500	9.332500	9
max	16.720000	16.460000	16.220000	15.930000	15.650000	15.320000	18.950000	18

In [5]:
```
#Preprocessing - Handling 0 and missing data
import numpy as np
from sklearn.impute import SimpleImputer
#changing zero value to NAN
data.PCT9MOFWD.replace(0, np.nan, inplace=True)
#Drop the NA value since its only one row
data = data.dropna()
data.shape
```

Out[5]: (222, 15)

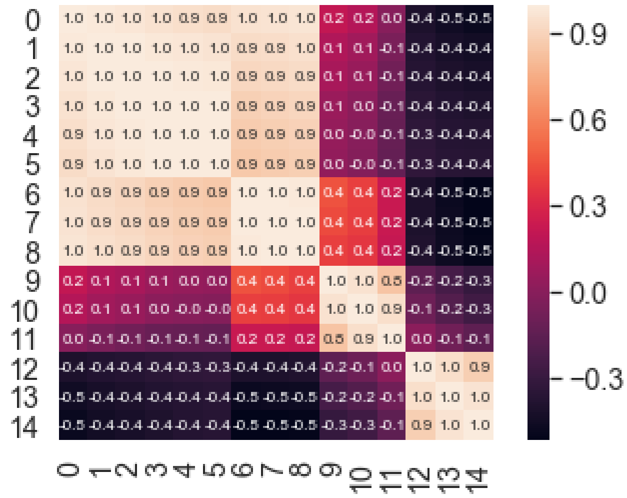
In [22]:
```
#Exploratory Data Analysis
#Histogram to plot the USPHCI index over the years
sns.set()
plt.hist(data['USPHCI'])
plt.xlabel('USPHCI Index range')
plt.ylabel('Frequency')
plt.show()
```



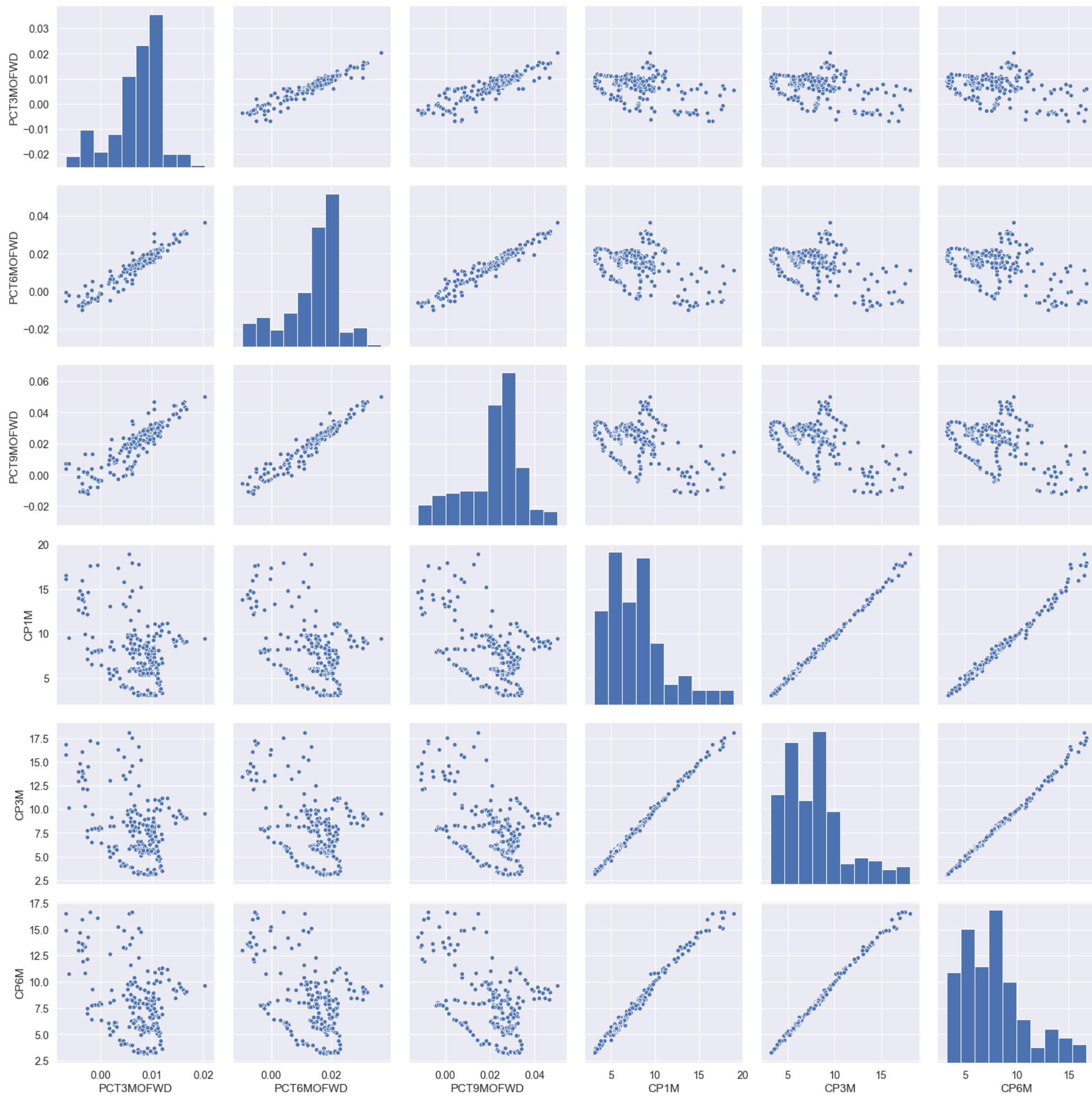
In [62]:
```
corr = data.corr()
print(corr)
```

```
             T1Y Index  T2Y Index  T3Y Index  T5Y Index  T7Y Index  T10Y Index  \
T1Y Index     1.000000   0.992364   0.981588   0.962056   0.947012    0.935681
T2Y Index     0.992364   1.000000   0.997406   0.987203   0.977596    0.969302
T3Y Index     0.981588   0.997406   1.000000   0.995567   0.989214    0.982970
T5Y Index     0.962056   0.987203   0.995567   1.000000   0.998327    0.995376
T7Y Index     0.947012   0.977596   0.989214   0.998327   1.000000    0.999082
T10Y Index    0.935681   0.969302   0.982970   0.995376   0.999082    1.000000
CP1M          0.962641   0.938317   0.920249   0.891523   0.873208    0.860518
CP3M          0.967578   0.945106   0.927676   0.899770   0.881932    0.869407
CP6M          0.972892   0.954110   0.938247   0.912096   0.895172    0.883008
CP1M_T1Y      0.208129   0.142681   0.109464   0.063193   0.045970    0.034947
CP3M_T1Y      0.152748   0.089627   0.057786   0.013663  -0.001902   -0.011444
CP6M_T1Y      0.001319  -0.050497  -0.075841  -0.111213  -0.122058   -0.127925
PCT3MOFWD    -0.406827  -0.382016  -0.367071  -0.350385  -0.335999   -0.326946
PCT6MOFWD    -0.454663  -0.423301  -0.405520  -0.383243  -0.366036   -0.354987
PCT9MOFWD    -0.483731  -0.444485  -0.424463  -0.397559  -0.377605   -0.364879

                CP1M      CP3M      CP6M  CP1M_T1Y  CP3M_T1Y  CP6M_T1Y  \
T1Y Index   0.962641  0.967578  0.972892  0.208129  0.152748  0.001319
T2Y Index   0.938317  0.945106  0.954110  0.142681  0.089627 -0.050497
T3Y Index   0.920249  0.927676  0.938247  0.109464  0.057786 -0.075841
T5Y Index   0.891523  0.899770  0.912096  0.063193  0.013663 -0.111213
T7Y Index   0.873208  0.881932  0.895172  0.045970 -0.001902 -0.122058
T10Y Index  0.860518  0.869407  0.883008  0.034947 -0.011444 -0.127925
CP1M        1.000000  0.998395  0.993283  0.449292  0.393453  0.229515
CP3M        0.998395  1.000000  0.997943  0.427221  0.383779  0.231517
CP6M        0.993283  0.997943  1.000000  0.393722  0.358950  0.221016
CP1M_T1Y    0.449292  0.427221  0.393722  1.000000  0.960717  0.842279
CP3M_T1Y    0.393453  0.383779  0.358950  0.960717  1.000000  0.946781
CP6M_T1Y    0.229515  0.231517  0.221016  0.842279  0.946781  1.000000
PCT3MOFWD  -0.404316 -0.401550 -0.395089 -0.150104 -0.096440  0.010641
PCT6MOFWD  -0.475103 -0.471441 -0.463500 -0.239304 -0.192718 -0.083744
PCT9MOFWD  -0.520132 -0.515032 -0.505840 -0.303912 -0.259039 -0.149062

            PCT3MOFWD  PCT6MOFWD  PCT9MOFWD
T1Y Index   -0.406827  -0.454663  -0.483731
T2Y Index   -0.382016  -0.423301  -0.444485
T3Y Index   -0.367071  -0.405520  -0.424463
T5Y Index   -0.350385  -0.383243  -0.397559
T7Y Index   -0.335999  -0.366036  -0.377605
T10Y Index  -0.326946  -0.354987  -0.364879
CP1M        -0.404316  -0.475103  -0.520132
CP3M        -0.401550  -0.471441  -0.515032
CP6M        -0.395089  -0.463500  -0.505840
CP1M_T1Y    -0.150104  -0.239304  -0.303912
CP3M_T1Y    -0.096440  -0.192718  -0.259039
CP6M_T1Y     0.010641  -0.083744  -0.149062
PCT3MOFWD    1.000000   0.952417   0.883160
PCT6MOFWD    0.952417   1.000000   0.968295
PCT9MOFWD    0.883160   0.968295   1.000000
```

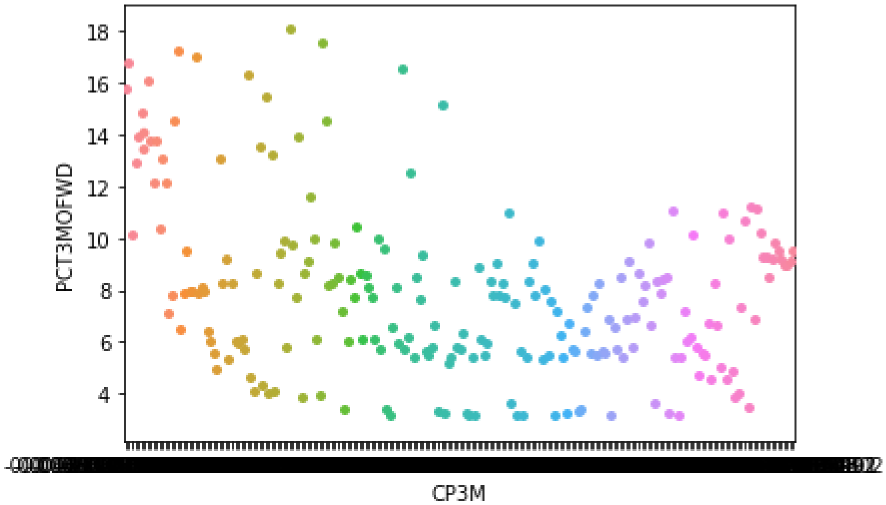
In [63]:
```
import numpy as np
cm = np.corrcoef(data.values.T)
sns.set(font_scale=1.3)
hm = sns.heatmap(cm,cbar=True,annot=True,square=True,fmt='.1f',annot_kws={'size': 8},yticklabels=True,xticklabels=True)
plt.show()
```



In [64]:
```
#Scatter plot for the above variables which show strong +ve and -ve correlatio
n
import matplotlib.pyplot as plt
import seaborn as sns

cols = ['PCT3MOFWD','PCT6MOFWD', 'PCT9MOFWD', 'CP1M', 'CP3M','CP6M']
sns.pairplot(data[cols], size=3.5)
plt.tight_layout()
plt.show()
```



In [58]:
```
#Bee Swarm plot
sns.swarmplot(x='PCT3MOFWD', y='CP3M', data=data)
plt.xlabel('CP3M')
plt.ylabel('PCT3MOFWD')
plt.show()
```



In [17]:
```
#Train Test split
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, 6:12]
#print(X)
y3month = data.iloc[:, 12].values
y6month = data.iloc[:, 13].values
y9month = data.iloc[:, 14].values
#print(y6month)
X_train, X_test, y_train, y_test = train_test_split(X, y3month, test_size=0.10
,random_state=42)
#print( X_train.shape, y_train.shape)
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [11]:
```
#########################Models for 3 Month forward#########################
# I. Linear regression model
import statsmodels.api as sm

X = X_train
y = y_train

# Note the difference in argument order
model_train = sm.OLS(y, X).fit()
predictions = model_train.predict(X) # make the predictions by the model

# Print out the statistics
model_train.summary()
#For test set
X = X_test
y = y_test

model_test = sm.OLS(y, X).fit()
predictions = model_test.predict(X) # make the predictions by the model

# Print out the statistics
model_test.summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.446
Model:	OLS	**Adj. R-squared (uncentered):**	-0.415
Method:	Least Squares	**F-statistic:**	0.5181
Date:	Sat, 19 Oct 2019	**Prob (F-statistic):**	0.870
Time:	19:48:45	**Log-Likelihood:**	84.887
No. Observations:	23	**AIC:**	-141.8
Df Residuals:	9	**BIC:**	-125.9
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	0.0791	0.109	0.725	0.487	-0.168	0.326
x2	-0.2136	0.321	-0.665	0.523	-0.940	0.513
x3	0.1093	0.231	0.474	0.647	-0.412	0.631
x4	0.0553	0.167	0.332	0.748	-0.322	0.433
x5	-0.1181	0.245	-0.483	0.641	-0.671	0.435
x6	0.0720	0.165	0.436	0.673	-0.302	0.446
x7	-0.1059	0.241	-0.439	0.671	-0.651	0.439
x8	-0.0089	0.432	-0.021	0.984	-0.986	0.968
x9	0.1254	0.250	0.501	0.628	-0.440	0.691
x10	0.0095	0.031	0.305	0.767	-0.061	0.080
x11	0.0120	0.048	0.248	0.809	-0.097	0.121
x12	-0.0225	0.034	-0.661	0.525	-0.100	0.055
x13	0.0003	0.013	0.025	0.980	-0.030	0.030
x14	0.0081	0.017	0.490	0.636	-0.029	0.045

Omnibus:	0.434	**Durbin-Watson:**	0.641
Prob(Omnibus):	0.805	**Jarque-Bera (JB):**	0.387
Skew:	0.275	**Prob(JB):**	0.824
Kurtosis:	2.682	**Cond. No.**	824.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [13]:
```
# II. Regression Tree Model
from sklearn.tree import DecisionTreeRegressor
# Import mean_squared_error as MSE
from sklearn.metrics import mean_squared_error as MSE
# Instantiate a DecisionTreeRegressor 'dt'
dt = DecisionTreeRegressor(max_depth=4,min_samples_leaf=0.1,random_state=3)
# Fit 'dt' to the training-set
dt.fit(X_train, y_train)
# Predict test-set labels
y_pred = dt.predict(X_test)
# Compute test-set MSE
mse_dt = MSE(y_test, y_pred)
# Compute test-set RMSE
rmse_dt = mse_dt**(1/2)
# Print rmse_dt
print(rmse_dt)
#Print
print('The regression Tree model fits extremely well with a very low rmse')
```

```
0.003853832023830866
The regression Tree model fits extremely well with a very low rmse
```

In [14]:
```
#Cross Validation
from sklearn.model_selection import cross_val_score
# Evaluate the list of MSE ontained by 10-fold CV
# Set n_jobs to -1 in order to exploit all CPU cores in computation
MSE_CV = - cross_val_score(dt, X_train, y_train, cv= 10,scoring='neg_mean_squa
red_error',n_jobs = -1)
# Fit 'dt' to the training set
dt.fit(X_train, y_train)
# Predict the labels of training set
y_predict_train = dt.predict(X_train)
# Predict the labels of test set
y_predict_test = dt.predict(X_test)
# Training set MSE
print('Train MSE: {:.7f}'.format(MSE(y_train, y_predict_train)))
# Test set MSE
print('Test MSE: {:.8f}'.format(MSE(y_test, y_predict_test)))
```

```
Train MSE: 0.0000144
Test MSE: 0.00001485
```

In [15]:
```
#III. SVD Regression
from sklearn.svm import SVR
svm=SVR(kernel='linear',degree=1,gamma=0.5,C=1.0)
svm.fit(X_train,y_train)
print("R-square:" + str(svm.score(X_train,y_train)))
```

```
R-square:-0.005509784364353454
```

In [152]:
```
print("Clearly the Decision Regression Tree is the best model that describes t
he Forward USHCPI index change wrt to CP and the spreads")
```

```
Clearly the Decision Regression Tree is the best model that describes the For
ward USHCPI index change wrt to CP and the spreads
```

In [16]:
```
#Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error as MSE
# Instantiate a random forests regressor 'rf' 400 estimators
rf = RandomForestRegressor(n_estimators=400,min_samples_leaf=0.12,random_state
=1)
# Fit 'rf' to the training set
rf.fit(X_train, y_train)
# Predict the test set labels 'y_pred'
y_pred = rf.predict(X_test)
# Evaluate the test set RMSE
rmse_test = MSE(y_test, y_pred)**(1/2)
# Print the test set RMSE
print('Test set RMSE of rf: {:.6f}'.format(rmse_test))
print("The Random Forest Regressor does a very good job in training the indivi
dual trees and introduces further randomization")
```

```
Test set RMSE of rf: 0.004138
The Random Forest Regressor does a very good job in training the individual t
rees and introduces further randomization
```

In [26]:
```
#####################6 Month Forward##################################
#I. Linear regression model

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, :-1]
#print(X)
#y3month = data.iloc[:, 6:12].values
y6month = data.iloc[:, 13].values
#y9month = data.iloc[:, 14].values
#print(y6month)
X_train6, X_test6, y_train6, y_test6 = train_test_split(X, y6month, test_size=
0.10,random_state=42)
#print( X_train.shape, y_train.shape)
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train6)
X_train6 = scaler.transform(X_train6)
X_test6 = scaler.transform(X_test6)
```

In [25]:
```
import statsmodels.api as sm

X = X_train6
y = y_train6

# Note the difference in argument order
model_train = sm.OLS(y, X).fit()
predictions = model_train.predict(X) # make the predictions by the model

# Print out the statistics
model_train.summary()
#For test set
X = X_test6
y = y_test6

model_test = sm.OLS(y, X).fit()
predictions = model_test.predict(X) # make the predictions by the model

# Print out the statistics
model_test.summary()
```

Out[25]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.429
Model:	OLS	**Adj. R-squared (uncentered):**	-0.458
Method:	Least Squares	**F-statistic:**	0.4837
Date:	Sat, 19 Oct 2019	**Prob (F-statistic):**	0.892
Time:	19:59:24	**Log-Likelihood:**	68.991
No. Observations:	23	**AIC:**	-110.0
Df Residuals:	9	**BIC:**	-94.08
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	0.1580	0.218	0.725	0.487	-0.335	0.651
x2	-0.4263	0.641	-0.665	0.523	-1.877	1.024
x3	0.2182	0.460	0.474	0.647	-0.823	1.260
x4	0.1104	0.333	0.332	0.748	-0.643	0.863
x5	-0.2357	0.488	-0.483	0.641	-1.340	0.869
x6	0.1437	0.330	0.436	0.673	-0.603	0.890
x7	-0.2113	0.481	-0.439	0.671	-1.300	0.877
x8	-0.0177	0.862	-0.021	0.984	-1.967	1.932
x9	0.2502	0.499	0.501	0.628	-0.879	1.379
x10	0.0190	0.062	0.305	0.767	-0.122	0.160
x11	0.0239	0.096	0.248	0.809	-0.193	0.241
x12	-0.0449	0.068	-0.661	0.525	-0.199	0.109
x13	-0.0091	0.026	-0.345	0.738	-0.069	0.051
x14	0.0254	0.033	0.770	0.461	-0.049	0.100

Omnibus:	0.434	**Durbin-Watson:**	0.641
Prob(Omnibus):	0.805	**Jarque-Bera (JB):**	0.387
Skew:	0.275	**Prob(JB):**	0.824
Kurtosis:	2.682	**Cond. No.**	824.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [28]:
```
# II. Regression Tree Model for 6month
from sklearn.tree import DecisionTreeRegressor
# Import mean_squared_error as MSE
from sklearn.metrics import mean_squared_error as MSE
# Instantiate a DecisionTreeRegressor 'dt'
dt = DecisionTreeRegressor(max_depth=4,min_samples_leaf=0.1,random_state=3)
# Fit 'dt' to the training-set
dt.fit(X_train6, y_train6)
# Predict test-set labels
y_pred6 = dt.predict(X_test6)
# Compute test-set MSE
mse_dt = MSE(y_test6, y_pred6)
# Compute test-set RMSE
rmse_dt = mse_dt**(1/2)
# Print rmse_dt
print(rmse_dt)
#Print
print('The regression Tree model for the 6month forward also fits extremely we
ll with a very low rmse')
```

```
0.0016084039691940787
The regression Tree model for the 6month forward also fits extremely well wit
h a very low rmse
```

In [29]:
```
#III. SVD Regression for 6month
from sklearn.svm import SVR

svm=SVR(kernel='linear',degree=1,gamma=0.5,C=1.0)
svm.fit(X_train6,y_train6)
print("R-square:" + str(svm.score(X_train6,y_train6)))
```

```
R-square:-0.007113320049616512
```

In [36]:
```
#Random Forest Regressor for 6month prediction
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error as MSE
# Instantiate a random forests regressor 'rf' 400 estimators
rf = RandomForestRegressor(n_estimators=400,min_samples_leaf=0.12,random_state
=1)
# Fit 'rf' to the training set
rf.fit(X_train6, y_train6)
# Predict the test set labels 'y_pred'
y_pred6 = rf.predict(X_test6)
# Evaluate the test set RMSE
rmse_test = MSE(y_test6, y_pred6)**(1/2)
# Print the test set RMSE
print('Test set RMSE of rf: {:.6f}'.format(rmse_test))
print("The Random Forest Regressor for 6 month also does a very good job in tr
aining the individual trees and introduces further randomization")
```

```
Test set RMSE of rf: 0.002250
The Random Forest Regressor for 6 month also does a very good job in training
the individual trees and introduces further randomization
```

In [41]:
```
#Train Test split
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, 6:12]
#print(X)
y9month = data.iloc[:, 14].values
#print(y6month)
X_train9, X_test9, y_train9, y_test9 = train_test_split(X, y9month, test_size=0.10,random_state=42)
#print( X_train.shape, y_train.shape)
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train9)
X_train = scaler.transform(X_train9)
X_test = scaler.transform(X_test9)
```

In [42]:
```
#for 9month
import statsmodels.api as sm

X = X_train9
y = y_train9

# Note the difference in argument order
model_train = sm.OLS(y, X).fit()
predictions = model_train.predict(X) # make the predictions by the model

# Print out the statistics
model_train.summary()
#For test set
X = X_test9
y = y_test9

model_test = sm.OLS(y, X).fit()
predictions = model_test.predict(X) # make the predictions by the model

# Print out the statistics
model_test.summary()
```

Out[42]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.907
Model:	OLS	**Adj. R-squared (uncentered):**	0.874
Method:	Least Squares	**F-statistic:**	27.56
Date:	Sat, 19 Oct 2019	**Prob (F-statistic):**	7.29e-08
Time:	20:14:57	**Log-Likelihood:**	80.025
No. Observations:	23	**AIC:**	-148.0
Df Residuals:	17	**BIC:**	-141.2
Df Model:	6		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
CP1M	0.0680	0.035	1.922	0.072	-0.007	0.143
CP3M	-0.1681	0.061	-2.754	0.014	-0.297	-0.039
CP6M	0.0996	0.038	2.616	0.018	0.019	0.180
CP1M_T1Y	-0.2578	0.213	-1.209	0.243	-0.708	0.192
CP3M_T1Y	0.6308	0.381	1.656	0.116	-0.173	1.434
CP6M_T1Y	-0.3438	0.271	-1.268	0.222	-0.916	0.228

Omnibus:	4.170	**Durbin-Watson:**	1.470
Prob(Omnibus):	0.124	**Jarque-Bera (JB):**	2.477
Skew:	-0.762	**Prob(JB):**	0.290
Kurtosis:	3.510	**Cond. No.**	3.89e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.89e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [43]:
```
# II. Regression Tree Model for 9month
from sklearn.tree import DecisionTreeRegressor
# Import mean_squared_error as MSE
from sklearn.metrics import mean_squared_error as MSE
# Instantiate a DecisionTreeRegressor 'dt'
dt = DecisionTreeRegressor(max_depth=4,min_samples_leaf=0.1,random_state=3)
# Fit 'dt' to the training-set
dt.fit(X_train9, y_train9)
# Predict test-set labels
y_pred9 = dt.predict(X_test9)
# Compute test-set MSE
mse_dt = MSE(y_test9, y_pred9)
# Compute test-set RMSE
rmse_dt = mse_dt**(1/2)
# Print rmse_dt
print(rmse_dt)
#Print
print('The regression Tree model for the 6month forward also fits extremely we
ll with a very low rmse')
```

```
0.009898814542418925
The regression Tree model for the 6month forward also fits extremely well wit
h a very low rmse
```

In [44]:
```
#III. SVD Regression for 9month
from sklearn.svm import SVR

svm=SVR(kernel='linear',degree=1,gamma=0.5,C=1.0)
svm.fit(X_train9,y_train9)
print("R-square:" + str(svm.score(X_train9,y_train9)))
```

```
R-square:-0.028668012407456755
```

In [46]:
```
#Random Forest Regressor for 9month prediction
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error as MSE
# Instantiate a random forests regressor 'rf' 400 estimators
rf = RandomForestRegressor(n_estimators=400,min_samples_leaf=0.12,random_state
=1)
# Fit 'rf' to the training set
rf.fit(X_train9, y_train9)
# Predict the test set labels 'y_pred'
y_pred9 = rf.predict(X_test9)
# Evaluate the test set RMSE
rmse_test = MSE(y_test9, y_pred9)**(1/2)
# Print the test set RMSE
print('Test set RMSE of rf: {:.6f}'.format(rmse_test))
print("The Random Forest Regressor for 9 month also does a very good job in tr
aining the individual trees and introduces further randomization")
```

```
Test set RMSE of rf: 0.009566
The Random Forest Regressor for 9 month also does a very good job in training
the individual trees and introduces further randomization
```

In [ ]:

```

```