Report Lab

Activity 6

ผู้สอน

รศ.ดร.อรฉัตร จิตต์โสภักตร์

กลุ่ม : OPหลุดเรท

นายฐานพัฒน์	สิทธิพรชัยสกุล	63010256
นายณภัทร	จิรารัตนกุลชัย	63010279

Data Preparation

```
# Get Data
# Read stock data use pandas_datareader.data from web
# Get Stock Data

stk_tickers = ['MSFT', 'IBM', 'GOOGL']
ccy_tickers = ['DEXJPUS', 'DEXUSUK']
idx_tickers = ['SP500', 'DJIA', 'VIXCLS']
stk_data = web.DataReader(stk_tickers, 'yahoo')
ccy_data = web.DataReader(ccy_tickers, 'fred')
idx_data = web.DataReader(idx_tickers, 'fred')
# Select columns
base = stk_data.loc[:, ('Adj Close', 'MSFT')]
X1 = stk_data.loc[:, ('Adj Close', ('GOOGL', 'IBM'))]
X2 = ccy_data
X3 = idx_data

# Standardize Data
# Standardized data (X1, X2, X3) with kept index (date)
standard_scaler = preprocessing.StandardScaler()
std = standard_scaler.fit_transform(X1.values)
std2 = standard_scaler.fit_transform(X2.values)
std3 = standard_scaler.fit_transform(X3.values)
X1 = pd.DataFrame(data=std,index=X1.index,columns=X1.columns)
X2 = pd.DataFrame(data=std2,index=X2.index,columns=X2.columns)
X3 = pd.DataFrame(data=std3,index=X3.index,columns=X3.columns)
X1
```

- เตรียมการข้อมูลโดยการดึงชุดข้อมูลตามที่อาจารย์กำหนด
- เลือกข้อมูลในคอลัมน์ที่ต้องการ
- ทำการ standardize ด้วย StandardScaler()

```
# Calculate ความแตกต่างของค่า ราคา 'Adj Close', 'MSFT')ย้อนหลัง backHisotry วัน
backHistory = [30, 45, 60, 90, 180, 240] # -> ทดลองหยิบ 3 ค่า 3 รูปแบบ
# เพื่อดูระยะเวลาการดูค่าข้อมูลย้อนหลังหลายๆแบบและเปรียบเทียบ MSE
BH1, BH2, BH3 = backHistory[1], backHistory[3], backHistory[4]
return_period = 2
Y = base.shift(-return_period)
X4_BH1 = base.diff( BH1).shift( - BH1)
X4_BH2 = base.diff( BH2).shift( - BH2)
X4_BH3 = base.diff( BH3).shift( - BH3)
X4 = pd.concat([X4_BH1, X4_BH2, X4_BH3], axis=1)
X4.columns = ['MSFT_3DT', 'MSFT_6DT', 'MSFT_12DT']
X4 = pd.DataFrame(standard_scaler.fit_transform(X4.values), index = X4.index,columns=X4.columns)
```

- ทำการหยิบค่ามา 3 ตัว สำหรับทำนายค่า
- จากนั้นนำมาทำ Dataframe

```
# Forming Dataset
X = pd.concat([X1, X2, X3, X4], axis=1)
data = pd.concat([Y, X], axis=1)
data
```

- รวม Dataframe X1 X2 X3 X4 เป็น Dataframe X
- นำ Dataframe X มารวมกับ Y

```
# Drop NA
data.dropna(inplace=True)
# View Statistics
data.describe()
```

	(Adj Close, MSFT)	(Adj Close, GOOGL)	(Adj Close, IBM)	DEXJPUS	DEXUSUK	SP500	DJIA	VIXCLS	MSFT_3DT	MSFT_6DT	MSFT_12DT
count	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000	1061.000000
mean	168.190781	-0.192291	-0.215152	-0.345302	0.158831	-0.183562	-0.169555	-0.109301	0.128483	0.137743	0.000898
std	73.971522	0.939794	0.885424	0.339120	0.875196	0.952513	0.969700	1.029335	0.888353	0.907923	0.995151
min	72.465790	-1.095639	-3.293872	-1.169259	-2.600069	-1.687178	-2.459049	-1.329234	-3.747857	-3.721287	-3.517305
25%	103.008080	-0.871068	-0.707332	-0.586610	-0.411335	-0.906628	-0.875276	-0.775713	-0.262240	-0.225727	-0.393076
50%	145.731384	-0.657421	-0.250051	-0.336729	0.081008	-0.595358	-0.514972	-0.358265	0.100181	0.094162	0.075891
75%	217.759995	0.141950	0.268217	-0.107569	0.890206	0.453799	0.384305	0.216013	0.540417	0.597251	0.626777
max	340.882812	2.102209	1.886851	0.488489	2.014633	2.053207	1.996294	7.152327	2.549654	2.204569	2.356102

- ดรอปข้อมูลที่ใช้ไม่ได้ และตรวจสอบข้อมูล

```python
# Assign X, Y (drop datetime index)
Y = data[data.columns[0]]
X = data[data.columns[1:]]
print(Y)
print(X)
# feature selection (correlation)
# Calculate correlation between variables for only continuous data columns
corr_data = X.corr()
# Reduce Corr() to Lower Matrix
lower_tri = corr_data.where(np.tril(np.ones(corr_data.shape),k=-1).astype(np.bool))
lower_tri.fillna(0, inplace=True)
# Drop columns if |correlation value| > 0.9
to_drop = [column for column in lower_tri.columns if any(lower_tri[column] > 0.9)]

X.drop(columns=to_drop, inplace=True)
X
```

- แบ่งข้อมูลเป็น 2 ส่วนคือ X กับ Y
- จากนั้นนำ Dataframe X มา Correlation
- จากนั้นนำมาทำ Dimensional Reduction เพื่อดรอปค่าที่เกินในส่วนที่เราต้องการ

```python
# Train / Test Preparation (try 2 Option) ปลดคอมเมนต์มาใช้ทีละอัน
# ------------------------------------------------------- #
# Option#1
Test_size = int(np.floor(0.3 * len(X)))
train_size = int(np.floor(0.7 * len(X)))
X_train, X_test = X[0:train_size], X[train_size:len(X)]
Y_train, Y_test = Y[0:train_size], Y[train_size:len(X)]
# ------------------------------------------------------- #
# Option #2
# seed = 4
# X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=0.3, random_state=seed)
# ------------------------------------------------------- #
# Perform Linear Regression -> All variables
lr = LinearRegression()
# Train
lr.fit(X_train, Y_train)
# Validate
# y_pred_lr = lr.predict(x_validate)
# Test
y_test_pred_lr = lr.predict(X_test)
# print(r2_score(y_pred_lr, y_validate))
print(r2_score(y_test_pred_lr, Y_test))
# print(lr.score(x_validate, y_validate))
print(lr.score(X_test, Y_test))
```

- ทำการแยกส่วนที่เอาไว้ Train กับ Test โดยแบ่งเป็น 2 options
- ทำ linear regression เพื่อหา prediction

GridSearchCV

```python
# Create Model List
regression = { 'LR': LinearRegression(), 'SVR': SVR(), }

# Create Parameter Dictionary for Linear Regression
fit_intercept = [True, False]
normalize = [True, False]
params_LR = dict( fit_intercept = fit_intercept, normalize = normalize)

# Create Parameter Dictionary for SVR
kernel = ['linear', 'rbf', 'poly']
C_list = [10, 100]
ep_list = [0.1, 1, 5]
gamma = [0.01, 0.1] # RBF
degree = [2, 3] # polynomial
params_SVR = dict( kernel = kernel, C = C_list, epsilon = ep_list, gamma = gamma, degree = degree )

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

- สร้าง Model สำหรับทำ GridSearch
- สร้าง Parameter Dictionary สำหรับ Linear Regression
- สร้าง Parameter Dictionary สำหรับ SVR
- ส่วนด้านล่างมี warning เล็กน้อยซึ่งทำให้มองยาก กลุ่มผมเลยปิดไว้

```
Output exceeds the size limit. Open the full output data in a text editor
LinearRegression()
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[CV 1/4; 1/4] START fit_intercept=True, normalize=True.........................
[CV 1/4; 1/4] END fit_intercept=True, normalize=True;, score=-534.384 total time=   0.0s
[CV 2/4; 1/4] START fit_intercept=True, normalize=True.........................
[CV 2/4; 1/4] END fit_intercept=True, normalize=True;, score=-808.010 total time=   0.0s
[CV 3/4; 1/4] START fit_intercept=True, normalize=True.........................
[CV 3/4; 1/4] END fit_intercept=True, normalize=True;, score=-1626.169 total time=   0.0s
[CV 4/4; 1/4] START fit_intercept=True, normalize=True.........................
[CV 4/4; 1/4] END fit_intercept=True, normalize=True;, score=-3101.403 total time=   0.0s
[CV 1/4; 2/4] START fit_intercept=True, normalize=False........................
[CV 1/4; 2/4] END fit_intercept=True, normalize=False;, score=-534.384 total time=   0.0s
[CV 2/4; 2/4] START fit_intercept=True, normalize=False........................
[CV 2/4; 2/4] END fit_intercept=True, normalize=False;, score=-808.010 total time=   0.0s
[CV 3/4; 2/4] START fit_intercept=True, normalize=False........................
[CV 3/4; 2/4] END fit_intercept=True, normalize=False;, score=-1626.169 total time=   0.0s
[CV 4/4; 2/4] START fit_intercept=True, normalize=False........................
[CV 4/4; 2/4] END fit_intercept=True, normalize=False;, score=-3101.403 total time=   0.0s
[CV 1/4; 3/4] START fit_intercept=False, normalize=True........................
[CV 1/4; 3/4] END fit_intercept=False, normalize=True;, score=-9790.406 total time=   0.0s
[CV 2/4; 3/4] START fit_intercept=False, normalize=True........................
[CV 2/4; 3/4] END fit_intercept=False, normalize=True;, score=-1734.164 total time=   0.0s
[CV 3/4; 3/4] START fit_intercept=False, normalize=True........................
[CV 3/4; 3/4] END fit_intercept=False, normalize=True;, score=-7267.371 total time=   0.0s
[CV 4/4; 3/4] START fit_intercept=False, normalize=True........................
...
[CV 3/4; 72/72] START C=100, degree=3, epsilon=5, gamma=0.1, kernel=poly........
[CV 3/4; 72/72] END C=100, degree=3, epsilon=5, gamma=0.1, kernel=poly;, score=-221.530 total time=   0.0s
[CV 4/4; 72/72] START C=100, degree=3, epsilon=5, gamma=0.1, kernel=poly........
[CV 4/4; 72/72] END C=100, degree=3, epsilon=5, gamma=0.1, kernel=poly;, score=-12345.404 total time=   0.0s
```

```
# Show Best Parameters for both models
print('Best params: ',grid_result.best_params_)
print('Best score: ', grid_result.best_score_)
```

```
Best params:  {'C': 100, 'degree': 2, 'epsilon': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Best score:  -1280.031214077796
```

- ตรวจสอบ Parameter ที่ทำให้ผลลัพธ์ดีที่สุด

```
# Show Score for each parameter combination for both model
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
zippy = zip(means, stds, params)
data_linear = []
data_rbf = []
data_poly = []
count = 0
for mean, stdev, param in zippy:
    print("%f (%f) with: %r" % (mean, stdev, param))
    if param['kernel'] == 'linear':
        data_linear.append({'means': mean, 'stdev': stdev})
    elif param['kernel'] == 'rbf':
        data_rbf.append({'means': mean, 'stdev': stdev})
    elif param['kernel'] == 'poly':
        data_poly.append({'means': mean, 'stdev': stdev})
```

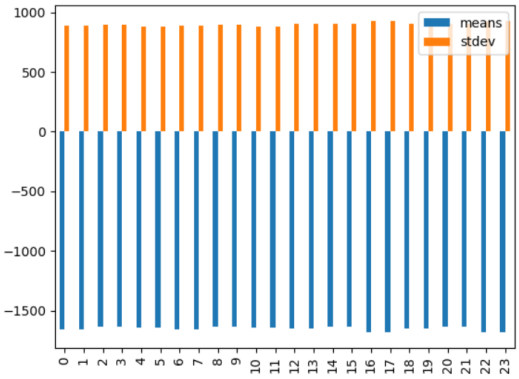
- ทำการแสดงผลค่า Score จากทุกๆ Parameter

```
-1655.639752 (888.955800) with: {'C': 10, 'degree': 2, 'epsilon': 0.1, 'gamma': 0.01, 'kernel': 'linear'}
-1477.386629 (1761.107862) with: {'C': 10, 'degree': 2, 'epsilon': 0.1, 'gamma': 0.01, 'kernel': 'rbf'}
-2725.466607 (2412.863579) with: {'C': 10, 'degree': 2, 'epsilon': 0.1, 'gamma': 0.01, 'kernel': 'poly'}
-1655.639752 (888.955800) with: {'C': 10, 'degree': 2, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'linear'}
-1521.196287 (1331.899225) with: {'C': 10, 'degree': 2, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
-2116.748766 (2381.793921) with: {'C': 10, 'degree': 2, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'poly'}
-1636.307449 (895.018284) with: {'C': 10, 'degree': 2, 'epsilon': 1, 'gamma': 0.01, 'kernel': 'linear'}
-1467.061706 (1727.697490) with: {'C': 10, 'degree': 2, 'epsilon': 1, 'gamma': 0.01, 'kernel': 'rbf'}
-2703.479062 (2404.917102) with: {'C': 10, 'degree': 2, 'epsilon': 1, 'gamma': 0.01, 'kernel': 'poly'}
-1636.307449 (895.018284) with: {'C': 10, 'degree': 2, 'epsilon': 1, 'gamma': 0.1, 'kernel': 'linear'}
-1551.437332 (1347.146993) with: {'C': 10, 'degree': 2, 'epsilon': 1, 'gamma': 0.1, 'kernel': 'rbf'}
-2063.139183 (2287.278636) with: {'C': 10, 'degree': 2, 'epsilon': 1, 'gamma': 0.1, 'kernel': 'poly'}
-1641.967030 (882.276122) with: {'C': 10, 'degree': 2, 'epsilon': 5, 'gamma': 0.01, 'kernel': 'linear'}
-1487.519739 (1668.670326) with: {'C': 10, 'degree': 2, 'epsilon': 5, 'gamma': 0.01, 'kernel': 'rbf'}
-2621.270489 (2336.620949) with: {'C': 10, 'degree': 2, 'epsilon': 5, 'gamma': 0.01, 'kernel': 'poly'}
-1641.967030 (882.276122) with: {'C': 10, 'degree': 2, 'epsilon': 5, 'gamma': 0.1, 'kernel': 'linear'}
-1642.751192 (1416.492291) with: {'C': 10, 'degree': 2, 'epsilon': 5, 'gamma': 0.1, 'kernel': 'rbf'}
-2258.995740 (2556.588467) with: {'C': 10, 'degree': 2, 'epsilon': 5, 'gamma': 0.1, 'kernel': 'poly'}
-1655.639752 (888.955800) with: {'C': 10, 'degree': 3, 'epsilon': 0.1, 'gamma': 0.01, 'kernel': 'linear'}
-1477.386629 (1761.107862) with: {'C': 10, 'degree': 3, 'epsilon': 0.1, 'gamma': 0.01, 'kernel': 'rbf'}
-2727.920978 (2383.479542) with: {'C': 10, 'degree': 3, 'epsilon': 0.1, 'gamma': 0.01, 'kernel': 'poly'}
-1655.639752 (888.955800) with: {'C': 10, 'degree': 3, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'linear'}
-1521.196287 (1331.899225) with: {'C': 10, 'degree': 3, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
-2219.614459 (2148.459171) with: {'C': 10, 'degree': 3, 'epsilon': 0.1, 'gamma': 0.1, 'kernel': 'poly'}
-1636.307449 (895.018284) with: {'C': 10, 'degree': 3, 'epsilon': 1, 'gamma': 0.01, 'kernel': 'linear'}
...
-2624.986122 (2321.243698) with: {'C': 100, 'degree': 3, 'epsilon': 5, 'gamma': 0.01, 'kernel': 'poly'}
-1678.969164 (928.418355) with: {'C': 100, 'degree': 3, 'epsilon': 5, 'gamma': 0.1, 'kernel': 'linear'}
-1389.383677 (1296.285832) with: {'C': 100, 'degree': 3, 'epsilon': 5, 'gamma': 0.1, 'kernel': 'rbf'}
-3576.288651 (5069.899048) with: {'C': 100, 'degree': 3, 'epsilon': 5, 'gamma': 0.1, 'kernel': 'poly'}
```

- การแสดงผลค่า Score จากหลายๆ Parameter

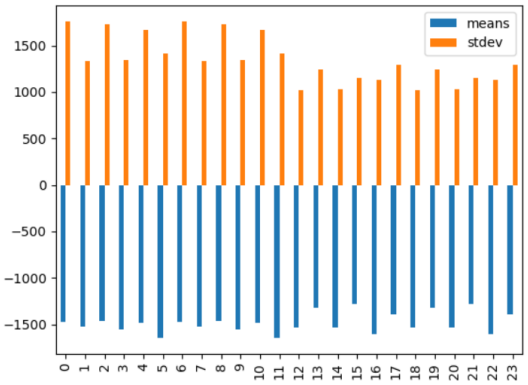
Option 1

```
# Display Mean, std, params
# plot linear
data_linear
df_linear = pd.DataFrame(data=data_linear)
df_linear
df_linear.plot.bar()
```



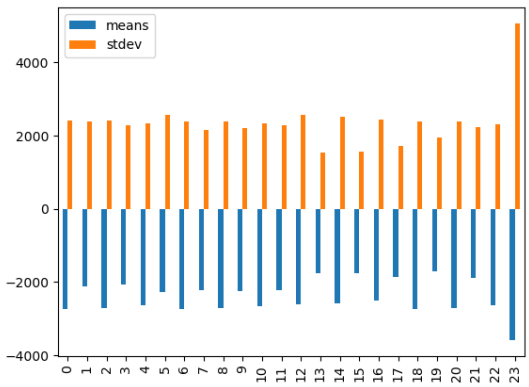
- แสดงผลกราฟ Linear

```
data_rbf
df_rbf = pd.DataFrame(data=data_rbf)
df_rbf.plot.bar()
```



- แสดงผลกราฟ RBF

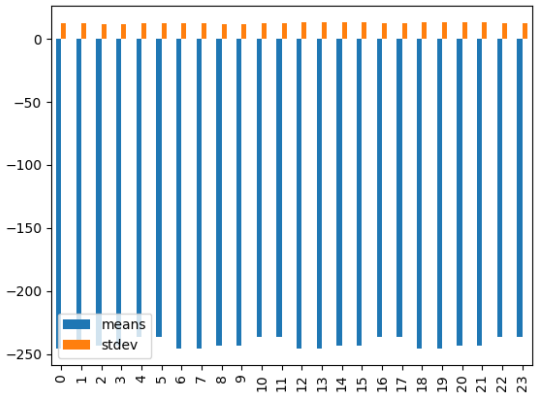
```
data_poly
df_poly = pd.DataFrame(data=data_poly)
df_poly.plot.bar()
```



- แสดงผลกราฟ Poly

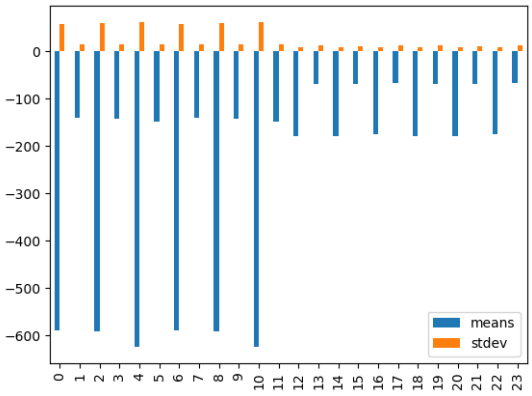
Option 2

```
# Display Mean, std, params
# plot linear
data_linear
df_linear = pd.DataFrame(data=data_linear)
df_linear
df_linear.plot.bar()
```



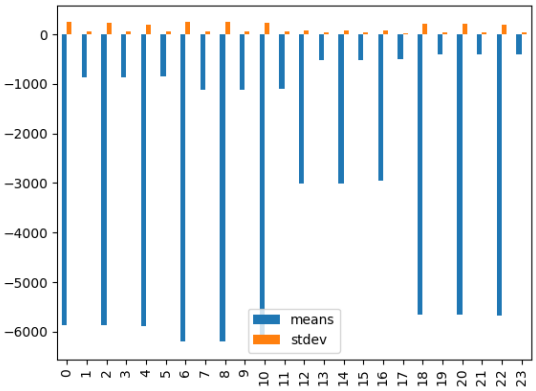
- แสดงผลกราฟ Linear

```
data_rbf
df_rbf = pd.DataFrame(data=data_rbf)
df_rbf.plot.bar()
```



- แสดงผลกราฟ RBF

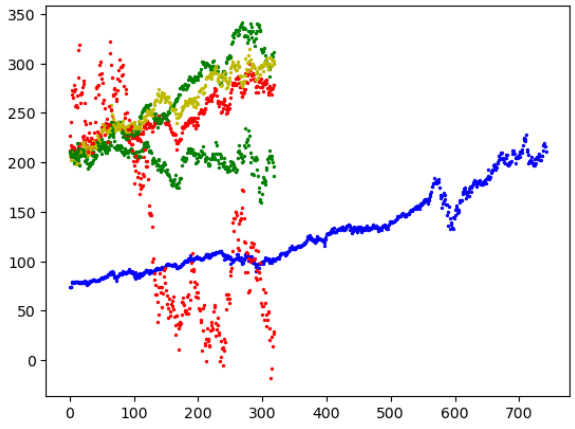
```
data_poly
df_poly = pd.DataFrame(data=data_poly)
df_poly.plot.bar()
```



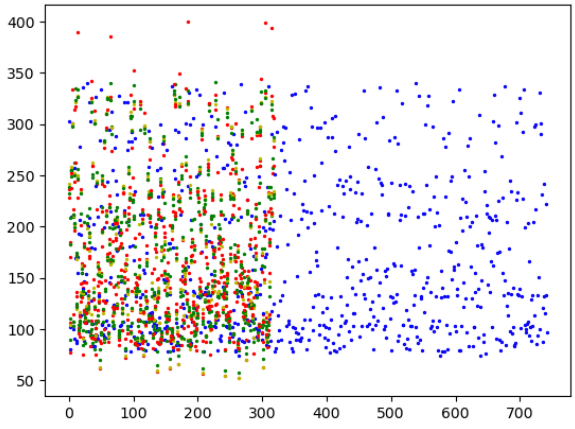
- แสดงผลกราฟ Poly

```python
# Show scatter plot compare y_test vs each model prediction
c_val = 100
svr_lin = SVR(kernel='linear', C=c_val)
svr_rbf = SVR(kernel='rbf', C=c_val, gamma=0.01)
svr_poly = SVR(kernel='poly', C=c_val, degree=2)
SVR_Linear = svr_lin.fit(X_train,Y_train).predict(X_test)
SVR_Rbf = svr_rbf.fit(X_train,Y_train).predict(X_test)
SVR_Poly = svr_poly.fit(X_train,Y_train).predict(X_test)
plt.scatter(np.arange(len(y_test_pred_lr)),y_test_pred_lr, edgecolors='r', s=2)
plt.scatter(np.arange(len(Y_train)),Y_train, edgecolors='b', s=2)
plt.scatter(np.arange(len(Y_test)),Y_test, edgecolors='g', s=2)
plt.scatter(np.arange(len(SVR_Linear)),SVR_Linear, edgecolors='y', s=2)
plt.scatter(np.arange(len(SVR_Rbf)),SVR_Rbf, edgecolors='g', s=2)
plt.scatter(np.arange(len(SVR_Poly)),SVR_Poly, edgecolors='r', s=2)
plt.show()
```

- แสดงผล scatter plot เพื่อเปรียบเทียบ y_test กับค่าที่ทำนายทุกๆค่า



option 1



option 2

RandomizeSearchCV

```python
# Create Model List
regression = { 'LR': LinearRegression(), 'SVR': SVR(), }
```

```python
# Create Parameter Dictionary for Linear Regression
fit_intercept = [True, False]
normalize = [True, False]
params_LR = dict( fit_intercept = fit_intercept, normalize = normalize)
```

```python
# Create Parameter Dictionary for SVR
kernel = ['linear', 'rbf', 'poly']
C_list = list(np.linspace(0.1, 150, 5, dtype = float))
ep_list = list(np.linspace(0.1, 1, 5, dtype = float))
gamma = list(np.linspace(0.01, 0.1, 5, dtype = float))
degree = [2, 3]
params_SVR = dict( kernel = kernel, C = C_list, epsilon = ep_list, gamma = gamma, degree = degree )
```

- สร้าง Model สำหรับทำ RandomizeSearch
- สร้าง Parameter Dictionary สำหรับ Linear Regression
- สร้าง Parameter Dictionary สำหรับ SVR

```python
# GridSearchCV() -> (a)
for EST in regression:
    model = regression[EST]
    if (EST == 'LR'):
        params = params_LR
    else:
        params = params_SVR
    grid_rand = RandomizedSearchCV( estimator=model, n_jobs = 1,
                                    verbose = 10,
                                    cv = 4,
                                    scoring = 'neg_mean_squared_error',
                                    param_distributions = params )
    grid_rand_result = grid_rand.fit(X_train, Y_train)
```

```
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[CV 1/4; 1/4] START fit_intercept=True, normalize=True.....................
[CV 1/4; 1/4] END fit_intercept=True, normalize=True;, score=-220.222 total time=   0.0s
[CV 2/4; 1/4] START fit_intercept=True, normalize=True.....................
[CV 2/4; 1/4] END fit_intercept=True, normalize=True;, score=-239.118 total time=   0.0s
[CV 3/4; 1/4] START fit_intercept=True, normalize=True.....................
[CV 3/4; 1/4] END fit_intercept=True, normalize=True;, score=-209.301 total time=   0.0s
[CV 4/4; 1/4] START fit_intercept=True, normalize=True.....................
[CV 4/4; 1/4] END fit_intercept=True, normalize=True;, score=-236.918 total time=   0.0s
[CV 1/4; 2/4] START fit_intercept=True, normalize=False....................
[CV 1/4; 2/4] END fit_intercept=True, normalize=False;, score=-220.222 total time=   0.0s
[CV 2/4; 2/4] START fit_intercept=True, normalize=False....................
[CV 2/4; 2/4] END fit_intercept=True, normalize=False;, score=-239.118 total time=   0.0s
[CV 3/4; 2/4] START fit_intercept=True, normalize=False....................
[CV 3/4; 2/4] END fit_intercept=True, normalize=False;, score=-209.301 total time=   0.0s
[CV 4/4; 2/4] START fit_intercept=True, normalize=False....................
[CV 4/4; 2/4] END fit_intercept=True, normalize=False;, score=-236.918 total time=   0.0s
[CV 1/4; 3/4] START fit_intercept=False, normalize=True....................
[CV 1/4; 3/4] END fit_intercept=False, normalize=True;, score=-8781.426 total time=   0.0s
[CV 2/4; 3/4] START fit_intercept=False, normalize=True....................
[CV 2/4; 3/4] END fit_intercept=False, normalize=True;, score=-8444.453 total time=   0.0s
[CV 3/4; 3/4] START fit_intercept=False, normalize=True....................
[CV 3/4; 3/4] END fit_intercept=False, normalize=True;, score=-8549.225 total time=   0.0s
[CV 4/4; 3/4] START fit_intercept=False, normalize=True....................
[CV 4/4; 3/4] END fit_intercept=False, normalize=True;, score=-8993.670 total time=   0.0s
...
[CV 3/4; 10/10] START C=37.575, degree=3, epsilon=0.1, gamma=0.0775, kernel=rbf.
[CV 3/4; 10/10] END C=37.575, degree=3, epsilon=0.1, gamma=0.0775, kernel=rbf;, score=-100.739 total time=   0.0s
[CV 4/4; 10/10] START C=37.575, degree=3, epsilon=0.1, gamma=0.0775, kernel=rbf.
[CV 4/4; 10/10] END C=37.575, degree=3, epsilon=0.1, gamma=0.0775, kernel=rbf;, score=-111.328 total time=   0.0s
```

- การแสดงผลค่า Score จากหลายๆ Parameter

```
# Show Best Parameters for both models
print('Best params: ',grid_rand_result.best_params_)
print('Best score: ', grid_rand_result.best_score_)
```
✓ 0.4s

```
Best params:  {'kernel': 'rbf', 'gamma': 0.0775, 'epsilon': 0.1, 'degree': 3, 'C': 75.05}
Best score:  -83.18613035590764
```

- ตรวจสอบ Parameter ที่ทำให้ผลลัพธ์ดีที่สุด

```
# Show Score for each parameter combination for both model
means = grid_rand_result.cv_results_['mean_test_score']
stds = grid_rand_result.cv_results_['std_test_score']
params = grid_rand_result.cv_results_['params']
data_linear = []
data_rbf = []
data_poly = []
all_data = []
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
    all_data.append({'means': mean, 'stdev': stdev})
    if param['kernel'] == 'linear':
        data_linear.append({'means': mean, 'stdev': stdev})
    elif param['kernel'] == 'rbf':
        data_rbf.append({'means': mean, 'stdev': stdev})
    elif param['kernel'] == 'poly':
        data_poly.append({'means': mean, 'stdev': stdev})
```

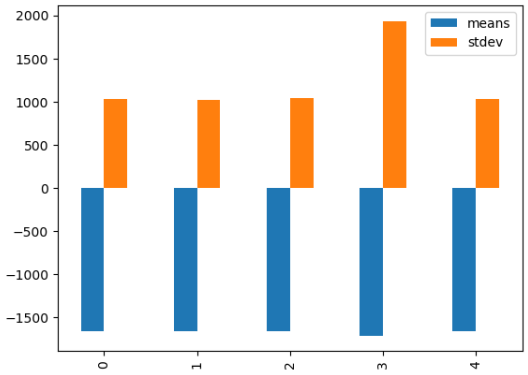
- ทำการแสดงผลค่า Score จากทุกๆ Parameter

```
-1305.164490 (108.298068) with: {'kernel': 'poly', 'gamma': 0.0325, 'epsilon': 1.0, 'degree': 2, 'C': 37.575}
-244.125537 (13.703429) with: {'kernel': 'linear', 'gamma': 0.0325, 'epsilon': 0.775, 'degree': 3, 'C': 37.575}
-243.526485 (13.072570) with: {'kernel': 'linear', 'gamma': 0.01, 'epsilon': 1.0, 'degree': 2, 'C': 37.575}
-569.122928 (33.741152) with: {'kernel': 'poly', 'gamma': 0.1, 'epsilon': 0.1, 'degree': 2, 'C': 37.575}
-83.186130 (13.757380) with: {'kernel': 'rbf', 'gamma': 0.0775, 'epsilon': 0.1, 'degree': 3, 'C': 75.05}
-389.596550 (39.309726) with: {'kernel': 'poly', 'gamma': 0.1, 'epsilon': 1.0, 'degree': 3, 'C': 112.525}
-83.353678 (15.405094) with: {'kernel': 'rbf', 'gamma': 0.05500000000000001, 'epsilon': 0.775, 'degree': 2, 'C': 150.0}
-1622.436035 (182.191029) with: {'kernel': 'linear', 'gamma': 0.0775, 'epsilon': 0.55, 'degree': 3, 'C': 0.1}
-108.338257 (16.173643) with: {'kernel': 'rbf', 'gamma': 0.0325, 'epsilon': 0.55, 'degree': 2, 'C': 112.525}
-96.810104 (10.114276) with: {'kernel': 'rbf', 'gamma': 0.0775, 'epsilon': 0.1, 'degree': 3, 'C': 37.575}
```

- การแสดงผลค่า Score จากหลายๆ Parameter

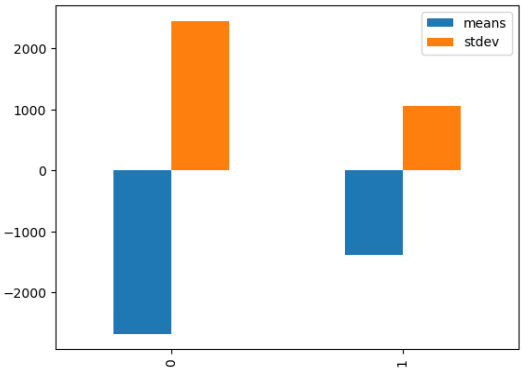
Option 1

```
# Display Mean, std, params
# plot linear
data_linear
df_linear = pd.DataFrame(data=data_linear)
df_linear
df_linear.plot.bar()
```



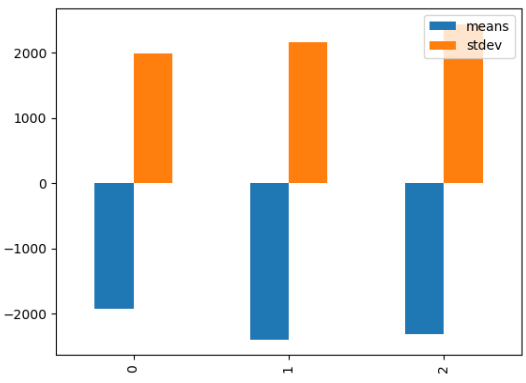
- แสดงผลกราฟ Linear

```
data_rbf
df_rbf = pd.DataFrame(data=data_rbf)
df_rbf.plot.bar()
```



- แสดงผลกราฟ RBF

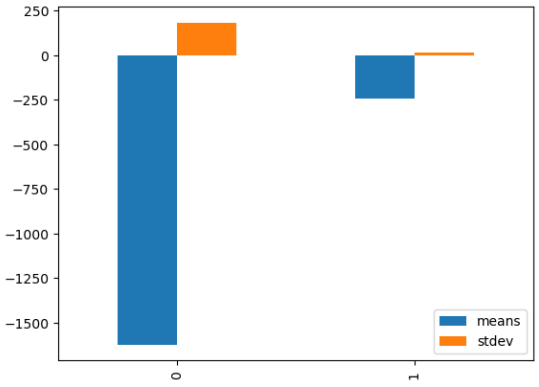
```
data_poly
df_poly = pd.DataFrame(data=data_poly)
df_poly.plot.bar()
```



- แสดงผลกราฟ Poly

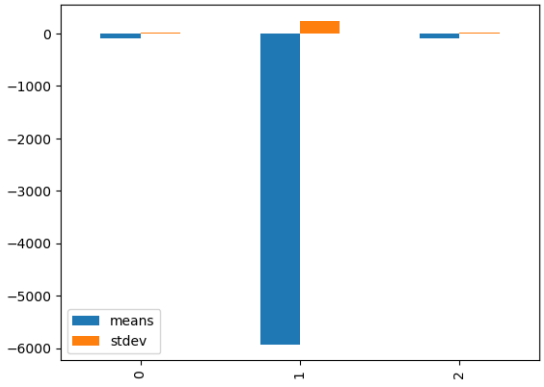
Option 2

```
# Display Mean, std, params
# plot linear
data_linear
df_linear = pd.DataFrame(data=data_linear)
df_linear
df_linear.plot.bar()
```



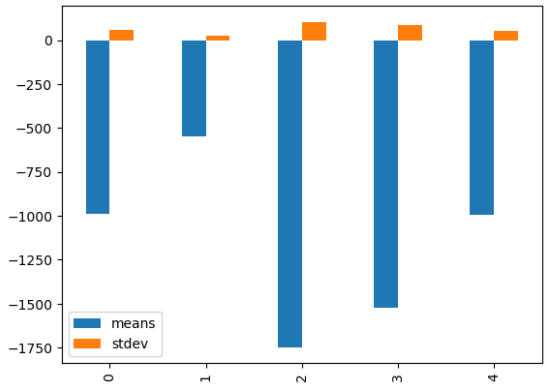
- แสดงผลกราฟ Linear

```
data_rbf
df_rbf = pd.DataFrame(data=data_rbf)
df_rbf.plot.bar()
```



- แสดงผลกราฟ RBF

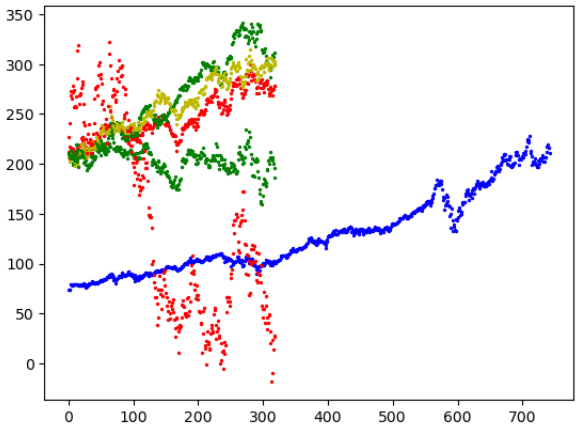
```
data_poly
df_poly = pd.DataFrame(data=data_poly)
df_poly.plot.bar()
```



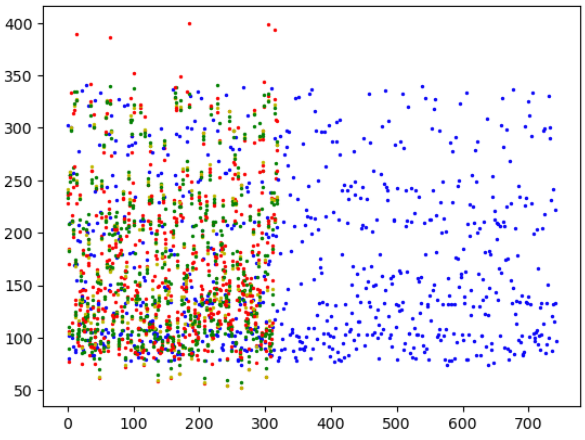
- แสดงผลกราฟ Poly

```python
# Show scatter plot compare y_test vs each model prediction
c_val = 100
svr_lin = SVR(kernel='linear', C=c_val)
svr_rbf = SVR(kernel='rbf', C=c_val, gamma=0.01)
svr_poly = SVR(kernel='poly', C=c_val, degree=2)
SVR_Linear = svr_lin.fit(X_train,Y_train).predict(X_test)
SVR_Rbf = svr_rbf.fit(X_train,Y_train).predict(X_test)
SVR_Poly = svr_poly.fit(X_train,Y_train).predict(X_test)
plt.scatter(np.arange(len(y_test_pred_lr)),y_test_pred_lr, edgecolors='r', s=2)
plt.scatter(np.arange(len(Y_train)),Y_train, edgecolors='b', s=2)
plt.scatter(np.arange(len(Y_test)),Y_test, edgecolors='g', s=2)
plt.scatter(np.arange(len(SVR_Linear)),SVR_Linear, edgecolors='y', s=2)
plt.scatter(np.arange(len(SVR_Rbf)),SVR_Rbf, edgecolors='g', s=2)
plt.scatter(np.arange(len(SVR_Poly)),SVR_Poly, edgecolors='r', s=2)
plt.show()
```

- แสดงผล scatter plot เพื่อเปรียบเทียบ y_test กับค่าที่ทำนายทุกๆค่า



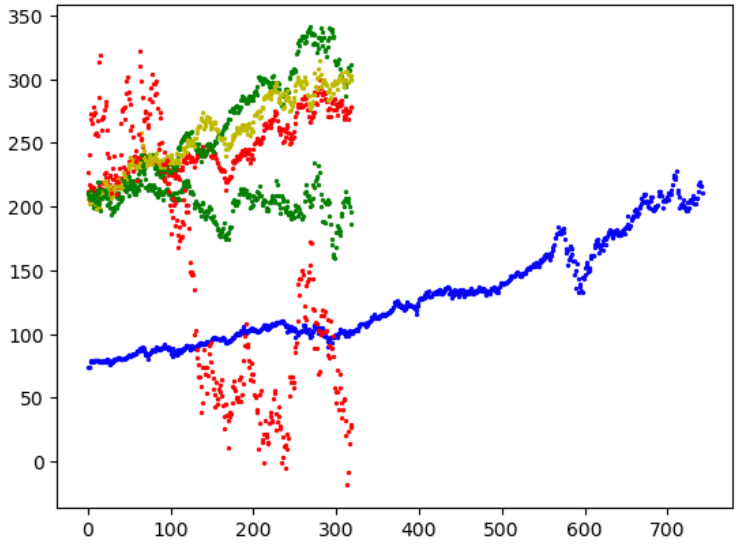
option 1



option 2

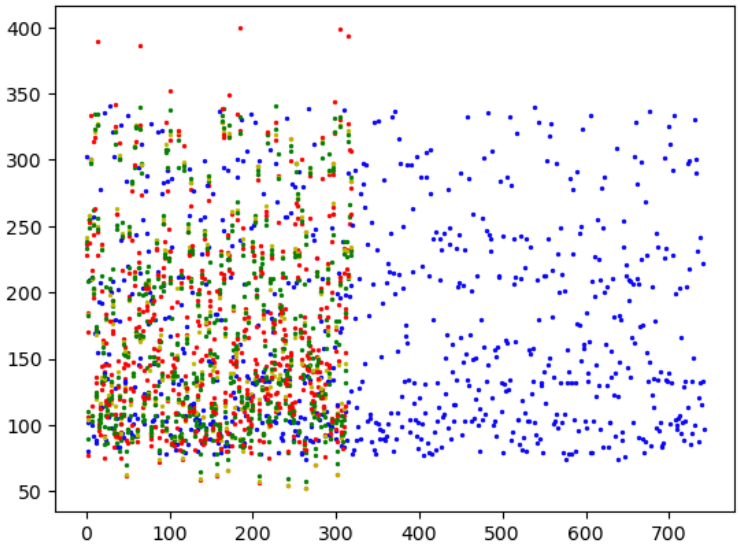
สรุปผลการทดลอง

จากการทดลองที่เราได้กำหนด option มาสองตัวจะสังเกตได้ว่าใน option 1 จะใช้วิธี train test split ซึ่งจะแบ่ง test size กับ train size ออกเป็นอัตราส่วน 3:7 โดยไม่ได้ shuffle จึงทำให้กราฟออกมาค่อนข้างเป็นอัตราส่วนกัน โดยจากภาพด้านล่างจุดสีฟ้าคือ y_train ซึ่งแบ่งมาด้วยอัตราส่วน 0.7 จึงค่อนข้างเยอะกว่าข้อมูลที่เหลือ



Option1

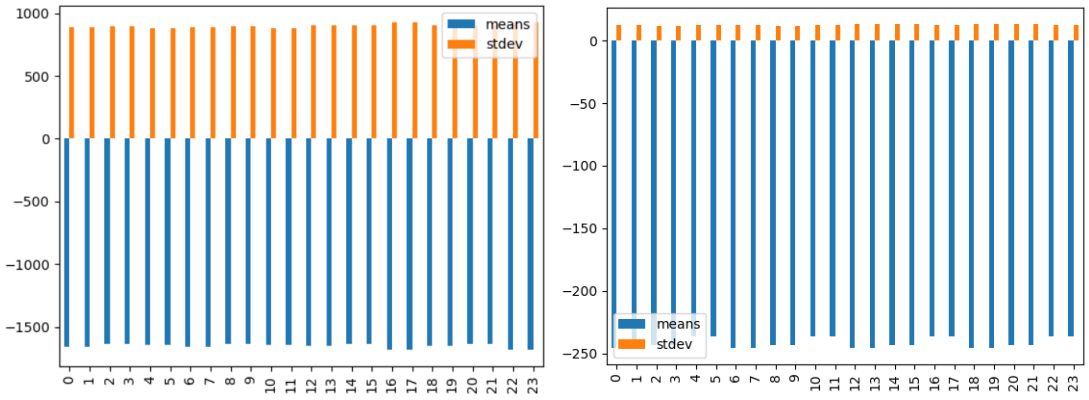
ส่วน option 2 จะใช้วิธี train test split แต่จะทำการ shuffle เพื่อให้ได้ test size กับ train size ออกมา โดยจะสังเกตได้ว่าข้อมูลมีการกระจายพอสมควร เพราะว่ามีการ shuffle ข้อมูลนั่นเอง



Option2

ส่วนความแตกต่างระหว่าง Grid Search และ Randomize Search ก็คือ Gird Search จะค้นหาค่าพารามิเตอร์จากชุดข้อมูลทั้งหมด
ซึ่งอาจจะใช้เวลานานแต่ก็จะได้ค่าที่ดีที่สุดจากข้อมูลทั้งหมด

แต่ Randomize Search จะทำการสุ่มหาชุดข้อมูลมาแล้วทำการหาค่าพารามิเตอร์อีกทีซึ่งจะทำให้ค่าของชุดข้อมูลบางอันค่อนข้างที่จะไม่ใกล้เคียงกันโดยสังเกตได้จากกราฟด้านล่าง



สังเกตได้จากกราฟด้านซ้ายซึ่งเป็นการใช้ Grid Search ส่วนภาพด้านขวาจะเป็นการใช้ Randomize Search