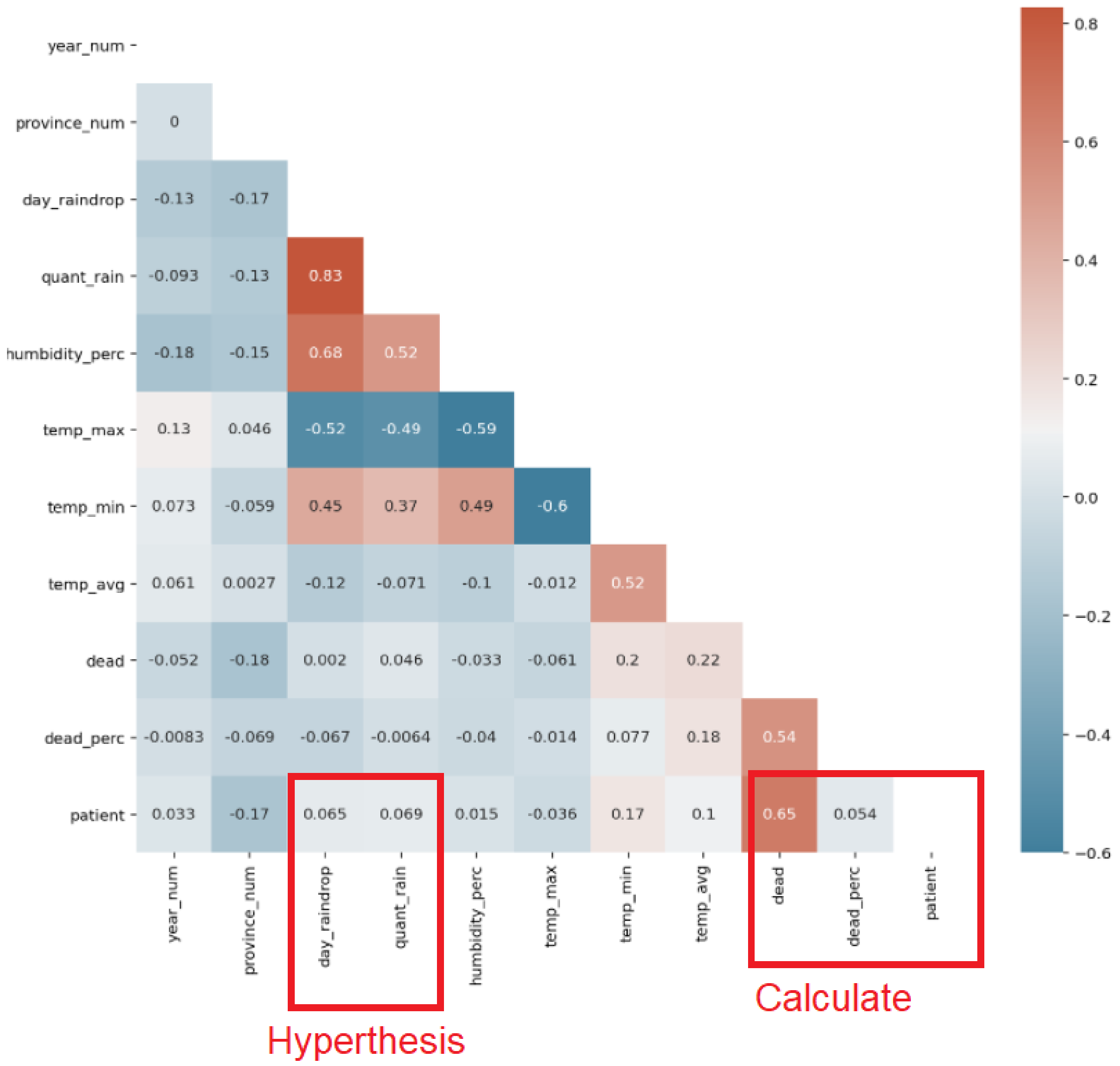
Regression Model Analysis Report

Dataset: dengue_preprocessing.csv

Features:

1. year_num: code of year 2016 – 2020 (1-5)
2. province_num: code of province 1- 77
3. day_raindrop: how many days of rain drop in 365 day
4. quant_rain: total quantity rainwater in unit of millimeters
5. humbidity_perc: average % humbidity in each province, each year
6. temp_max: maximum temp in each province, each year
7. temp_min: minimum temp in each province, each year
8. temp_avg: average temp in each province, each year
9. dead: amount of dead person from dengue
10. dead_perc: percent of dead person from dengue
11. patient: amount of patient from dengue (target variable)

What we want to know? **Rain has affected to dengue patient or not?**

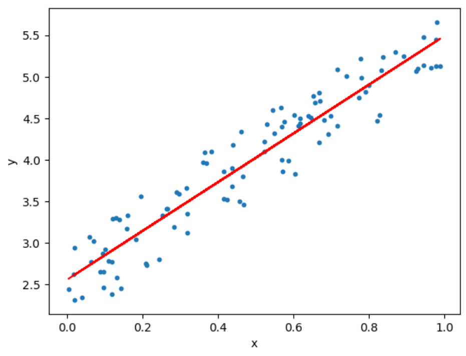


Model testing: Linear Regression Base line

We tried to test model in 5 ways

1. Multiple Linear Regression (original)
2. K-folds Cross Validation(+Repeats)
3. Features selection: Recursive Feature Elimination (RFE)
4. Grid Search CV
5. Polynomial regression

1. Multiple Linear Regression



What is Linear Regression

The objective of a linear regression model is to find a relationship between one or more features(independent variables) and a continuous target variable(dependent variable). When there is only feature it is called *Uni-variate* Linear Regression and if there are multiple features, it is called *Multi*ple Linear Regression.

Hypothesis of Linear Regression

The linear regression model can be represented by the following equation

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

- Y is the predicted value
- θ_0 is the bias term.
- $\theta_1,\ldots,\theta_n$ are the model parameters
- $x_1, x_2,\ldots,x_n$ are the feature values.

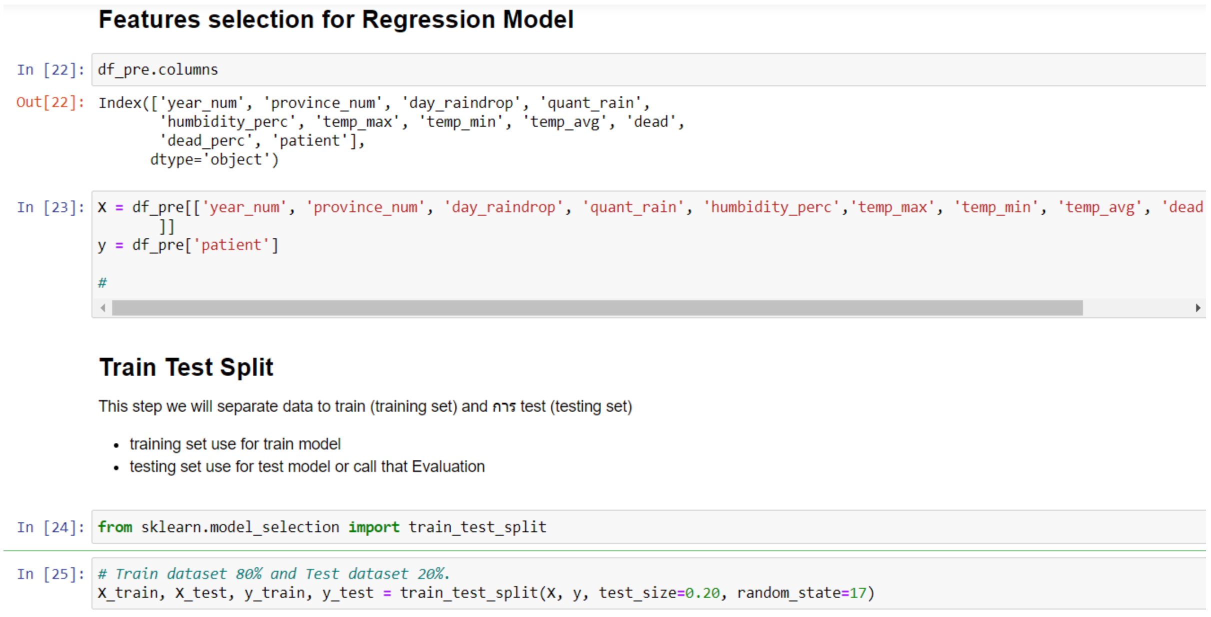
The above hypothesis can also be represented by

$$Y = \theta^T x$$

where

- θ is the model's parameter vector including the bias term θ_0
- x is the feature vector with $x_0 = 1$

Training a Linear Regression model



Features selection for Regression Model

```
In [22]: df_pre.columns

Out[22]: Index(['year_num', 'province_num', 'day_raindrop', 'quant_rain',
                'humbidity_perc', 'temp_max', 'temp_min', 'temp_avg', 'dead',
                'dead_perc', 'patient'],
               dtype='object')
```

```
In [23]: X = df_pre[['year_num', 'province_num', 'day_raindrop', 'quant_rain', 'humbidity_perc','temp_max', 'temp_min', 'temp_avg', 'dead
                 ]]
         y = df_pre['patient']

         #
```

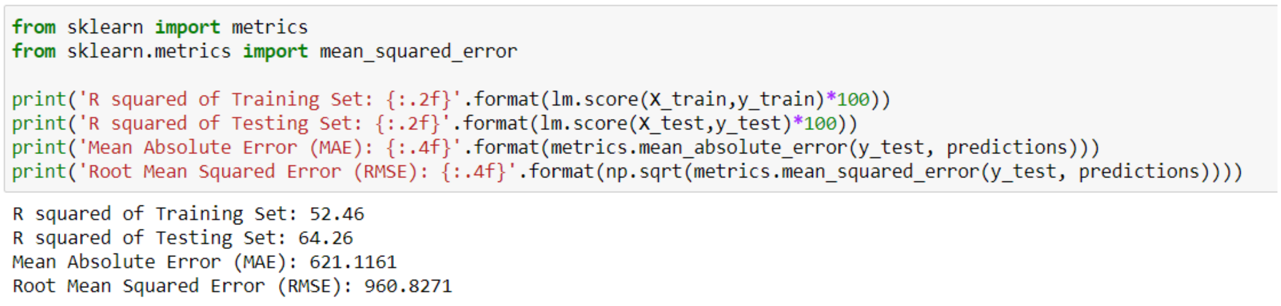
Train Test Split

This step we will separate data to train (training set) and การ test (testing set)

- training set use for train model
- testing set use for test model or call that Evaluation

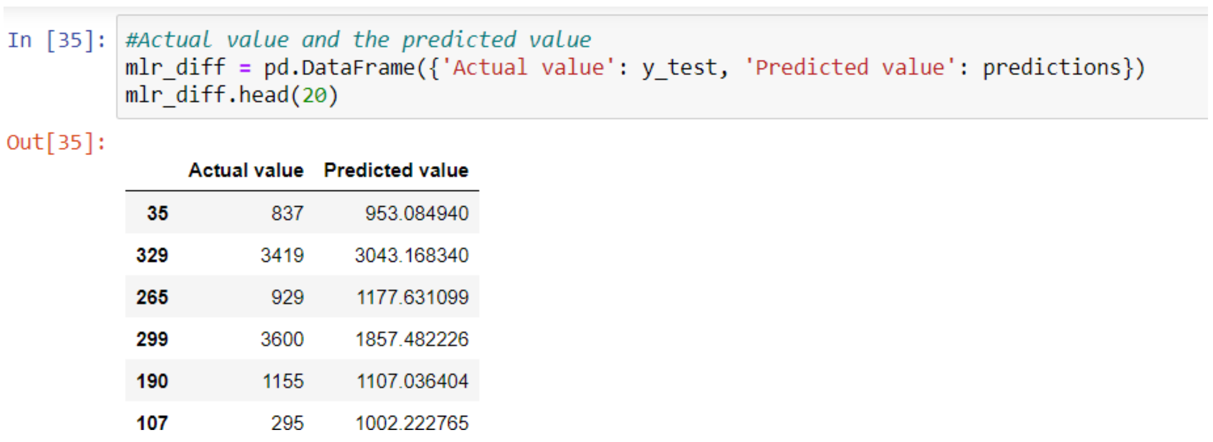
```
In [24]: from sklearn.model_selection import train_test_split
```

```
In [25]: # Train dataset 80% and Test dataset 20%.
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=17)
```



```
from sklearn import metrics
from sklearn.metrics import mean_squared_error

print('R squared of Training Set: {:.2f}'.format(lm.score(X_train,y_train)*100))
print('R squared of Testing Set: {:.2f}'.format(lm.score(X_test,y_test)*100))
print('Mean Absolute Error (MAE): {:.4f}'.format(metrics.mean_absolute_error(y_test, predictions)))
print('Root Mean Squared Error (RMSE): {:.4f}'.format(np.sqrt(metrics.mean_squared_error(y_test, predictions))))
```

```
R squared of Training Set: 52.46
R squared of Testing Set: 64.26
Mean Absolute Error (MAE): 621.1161
Root Mean Squared Error (RMSE): 960.8271
```



```
In [35]: #Actual value and the predicted value
         mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': predictions})
         mlr_diff.head(20)
```

Out[35]:

	Actual value	Predicted value
35	837	953.084940
329	3419	3043.168340
265	929	1177.631099
299	3600	1857.482226
190	1155	1107.036404
107	295	1002.222765

2. Repeats k-folds Cross Validation

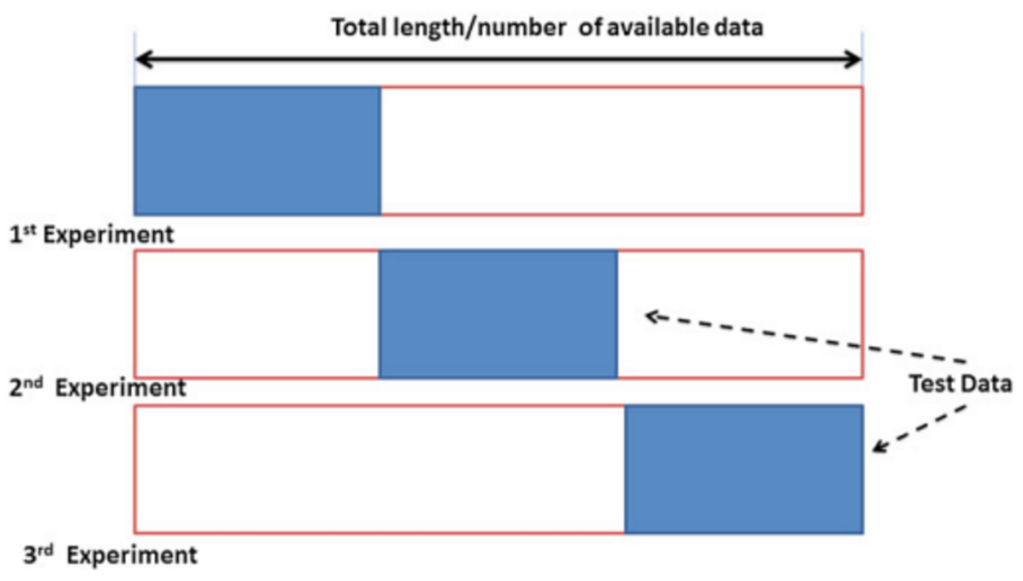
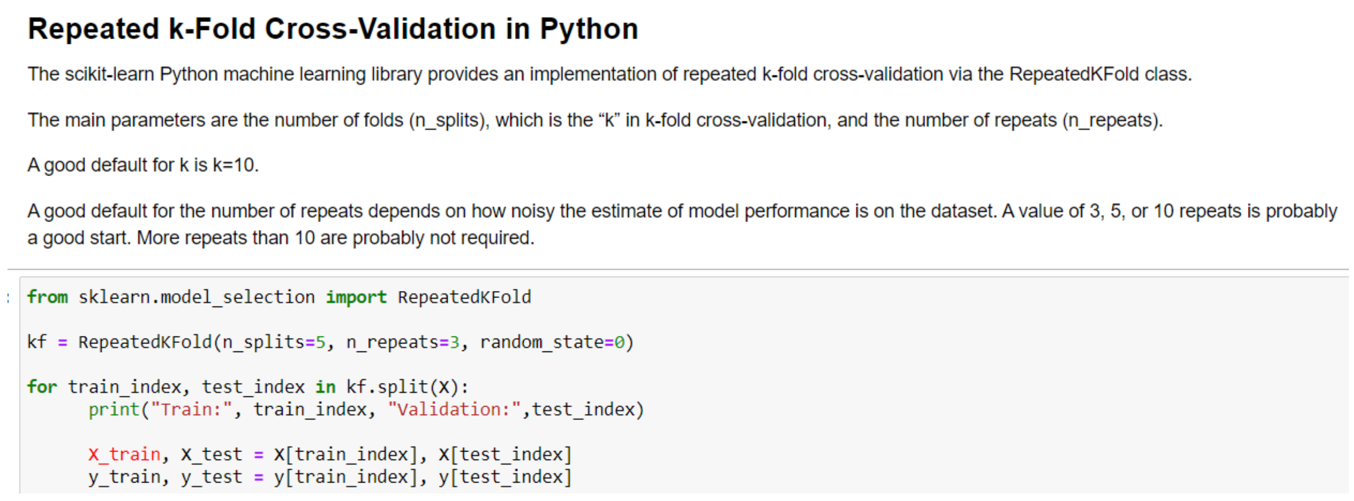


Fig. 3.8 Data splitting in *K*-fold cross-validation

The scikit-learn Python machine learning library provides an implementation of repeated k-fold cross-validation via the Repeated K-Fold class.

The main parameters are the number of folds (*n_splits*), which is the "*k*" in k-fold cross-validation, and the number of repeats (*n_repeats*).
A good default for k is k=10.

A good default for the number of repeats depends on how noisy the estimate of model performance is on the dataset. A value of 3, 5, or 10 repeats is probably a good start. More repeats than 10 are probably not required.



Repeated k-Fold Cross-Validation in Python

The scikit-learn Python machine learning library provides an implementation of repeated k-fold cross-validation via the RepeatedKFold class.

The main parameters are the number of folds (n_splits), which is the "k" in k-fold cross-validation, and the number of repeats (n_repeats).

A good default for k is k=10.

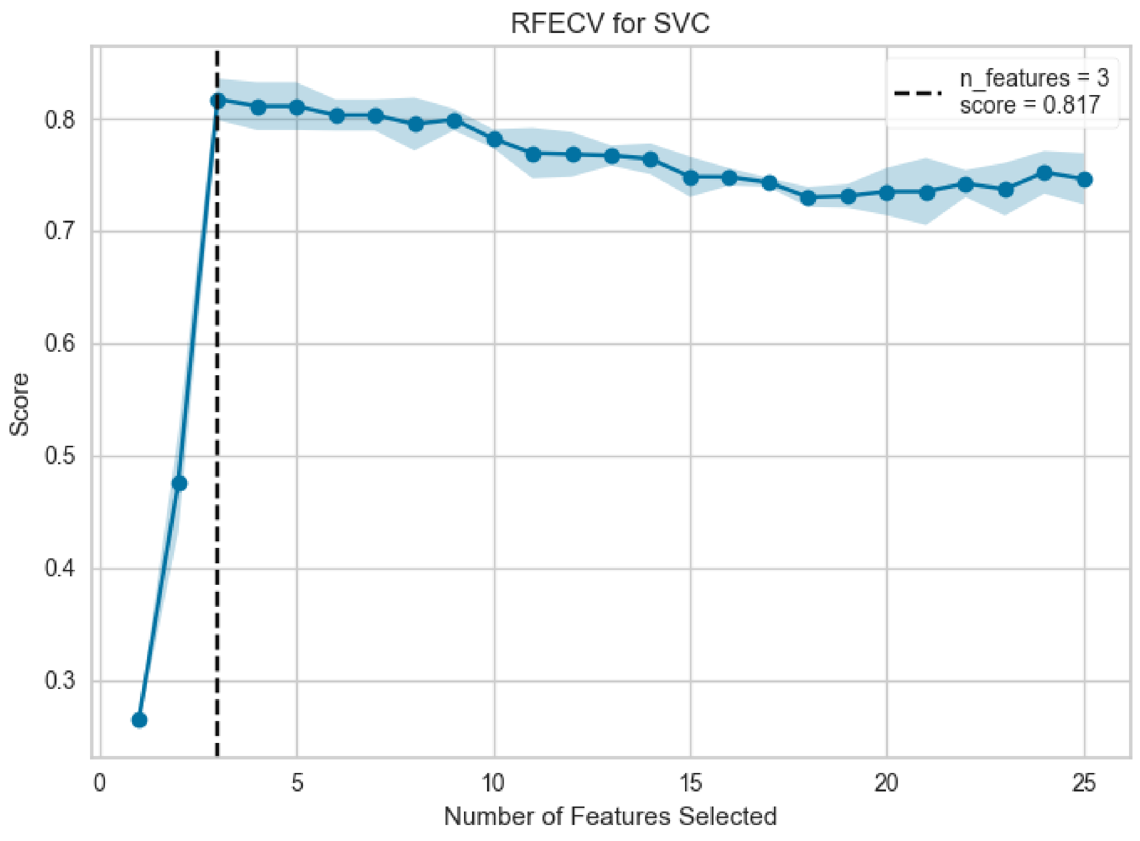
A good default for the number of repeats depends on how noisy the estimate of model performance is on the dataset. A value of 3, 5, or 10 repeats is probably a good start. More repeats than 10 are probably not required.

```
from sklearn.model_selection import RepeatedKFold

kf = RepeatedKFold(n_splits=5, n_repeats=3, random_state=0)

for train_index, test_index in kf.split(X):
      print("Train:", train_index, "Validation:",test_index)

      X_train, X_test = X[train_index], X[test_index]
      y_train, y_test = y[train_index], y[test_index]
```

```python
from sklearn import metrics
from sklearn.metrics import mean_squared_error

print('R squared of Training Set: {:.2f}'.format(lm.score(X_train,y_train)*100))
print('R squared of Test Set: {:.2f}'.format(lm.score(X_test,y_test)*100))
print('Mean Absolute Error (MAE): {:.4f}'.format(metrics.mean_absolute_error(y_test, predictions)))
print('Root Mean Squared Error (RMSE): {:.4f}'.format(np.sqrt(metrics.mean_squared_error(y_test, predictions))))
```

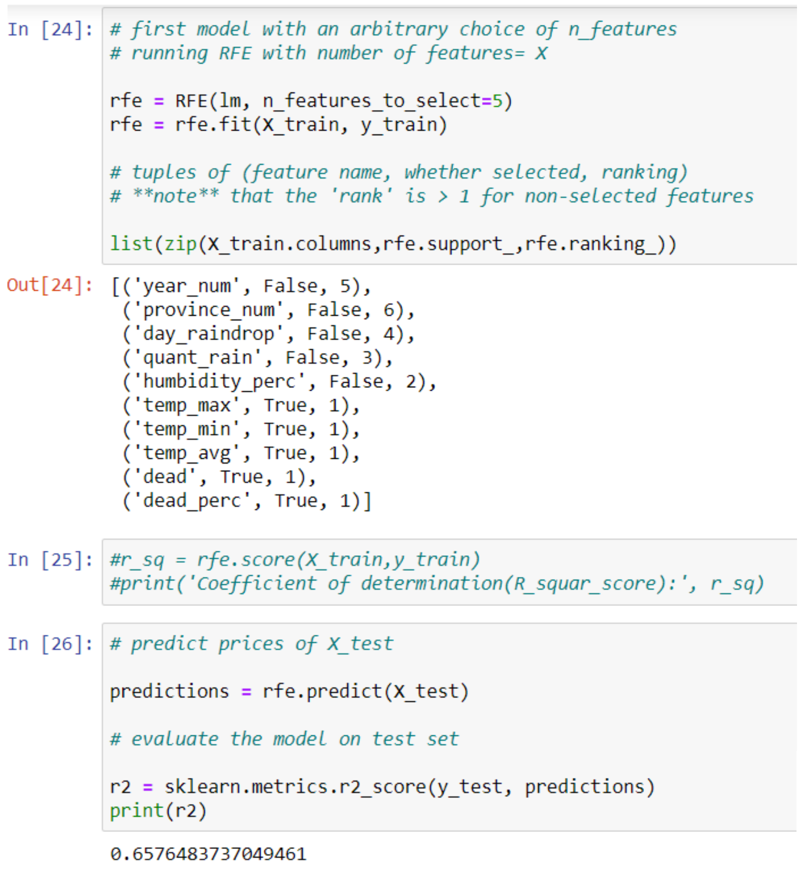
```
R squared of Training Set: 55.25
R squared of Test Set: 57.07
Mean Absolute Error (MAE): 676.5037
Root Mean Squared Error (RMSE): 998.2440
```

3.Features selection: Recursive Feature Elimination (RFE)

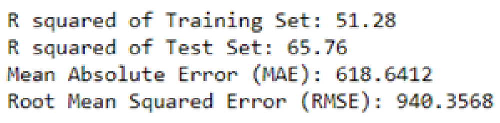


Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. Features are ranked by the model's `coef_` or `feature_importances_` attributes, and by recursively eliminating a small number of features per loop, RFE attempts to eliminate dependencies and collinearity that may exist in the model.

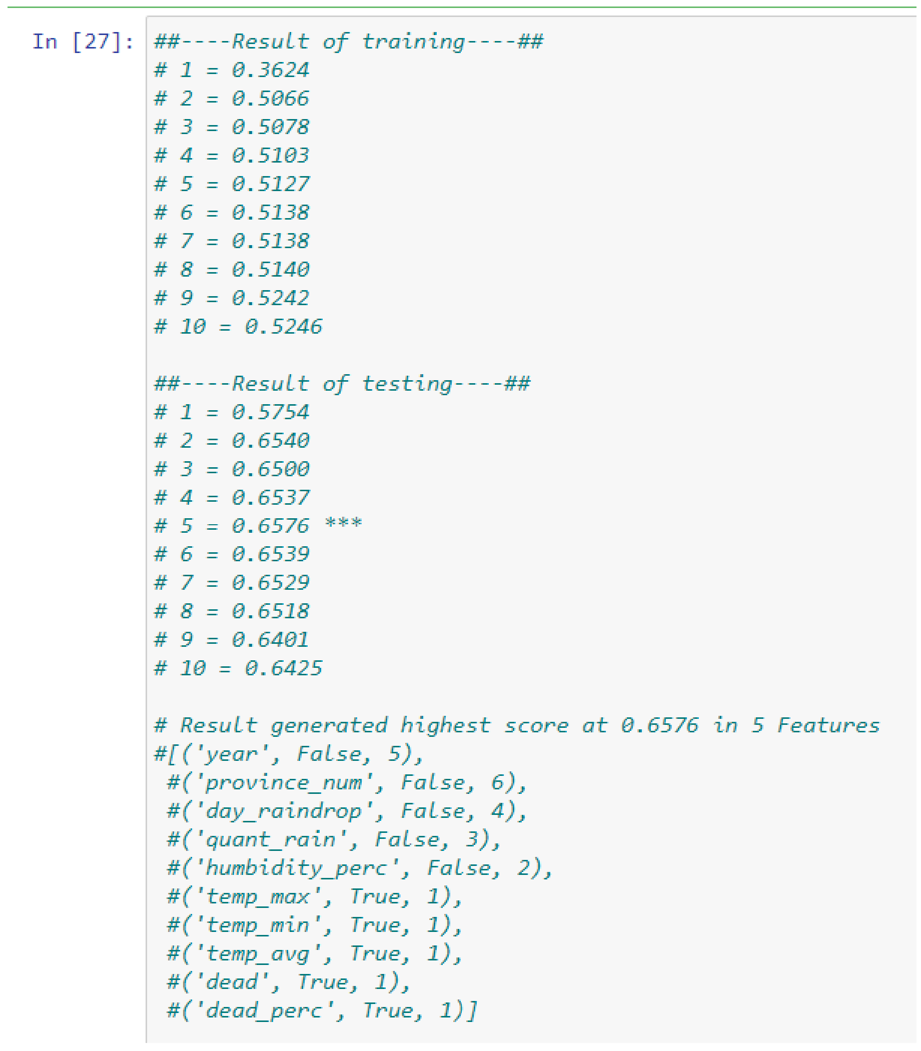
RFE requires a specified number of features to keep, however it is often not known in advance how many features are valid. To find the optimal number of features cross-validation is used with RFE to score different feature subsets and select the best scoring collection of features. The `RFECV` visualizer plots the number of features in the model along with their cross-validated test score and variability and visualizes the selected number of features.



```
In [24]: # first model with an arbitrary choice of n_features
         # running RFE with number of features= X

         rfe = RFE(lm, n_features_to_select=5)
         rfe = rfe.fit(X_train, y_train)

         # tuples of (feature name, whether selected, ranking)
         # **note** that the 'rank' is > 1 for non-selected features

         list(zip(X_train.columns,rfe.support_,rfe.ranking_))

Out[24]: [('year_num', False, 5),
          ('province_num', False, 6),
          ('day_raindrop', False, 4),
          ('quant_rain', False, 3),
          ('humbidity_perc', False, 2),
          ('temp_max', True, 1),
          ('temp_min', True, 1),
          ('temp_avg', True, 1),
          ('dead', True, 1),
          ('dead_perc', True, 1)]

In [25]: #r_sq = rfe.score(X_train,y_train)
         #print('Coefficient of determination(R_squar_score):', r_sq)

In [26]: # predict prices of X_test

         predictions = rfe.predict(X_test)

         # evaluate the model on test set

         r2 = sklearn.metrics.r2_score(y_test, predictions)
         print(r2)

         0.6576483737049461
```



```
R squared of Training Set: 51.28
R squared of Test Set: 65.76
Mean Absolute Error (MAE): 618.6412
Root Mean Squared Error (RMSE): 940.3568
```



```
In [27]: ##----Result of training----##
         # 1 = 0.3624
         # 2 = 0.5066
         # 3 = 0.5078
         # 4 = 0.5103
         # 5 = 0.5127
         # 6 = 0.5138
         # 7 = 0.5138
         # 8 = 0.5140
         # 9 = 0.5242
         # 10 = 0.5246

         ##----Result of testing----##
         # 1 = 0.5754
         # 2 = 0.6540
         # 3 = 0.6500
         # 4 = 0.6537
         # 5 = 0.6576 ***
         # 6 = 0.6539
         # 7 = 0.6529
         # 8 = 0.6518
         # 9 = 0.6401
         # 10 = 0.6425

         # Result generated highest score at 0.6576 in 5 Features
         #[('year', False, 5),
          #('province_num', False, 6),
          #('day_raindrop', False, 4),
          #('quant_rain', False, 3),
          #('humbidity_perc', False, 2),
          #('temp_max', True, 1),
          #('temp_min', True, 1),
          #('temp_avg', True, 1),
          #('dead', True, 1),
          #('dead_perc', True, 1)]
```

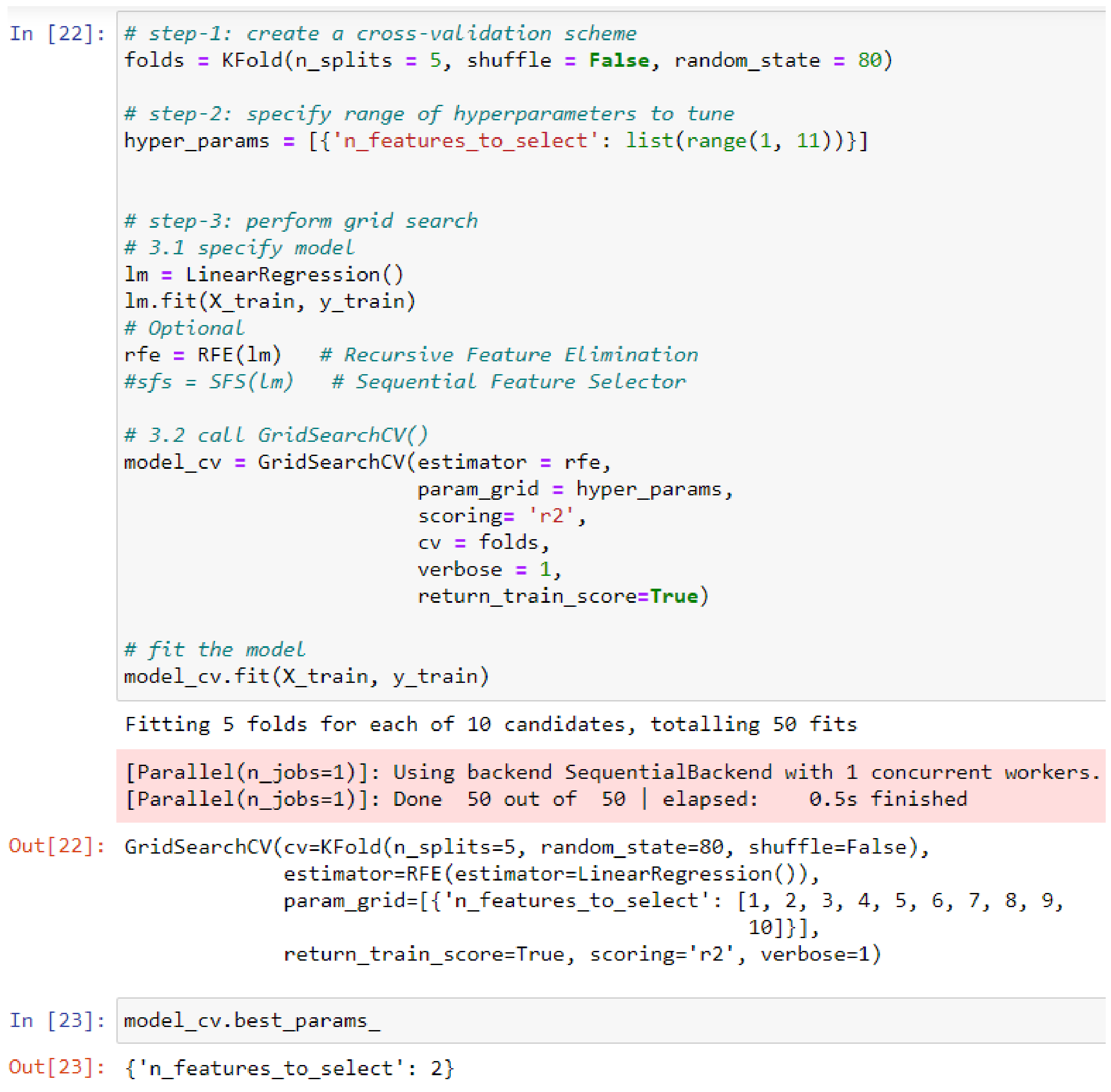
4.Grid Search CV

Grid Search CV is a library function that is a member of sklearn's model selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

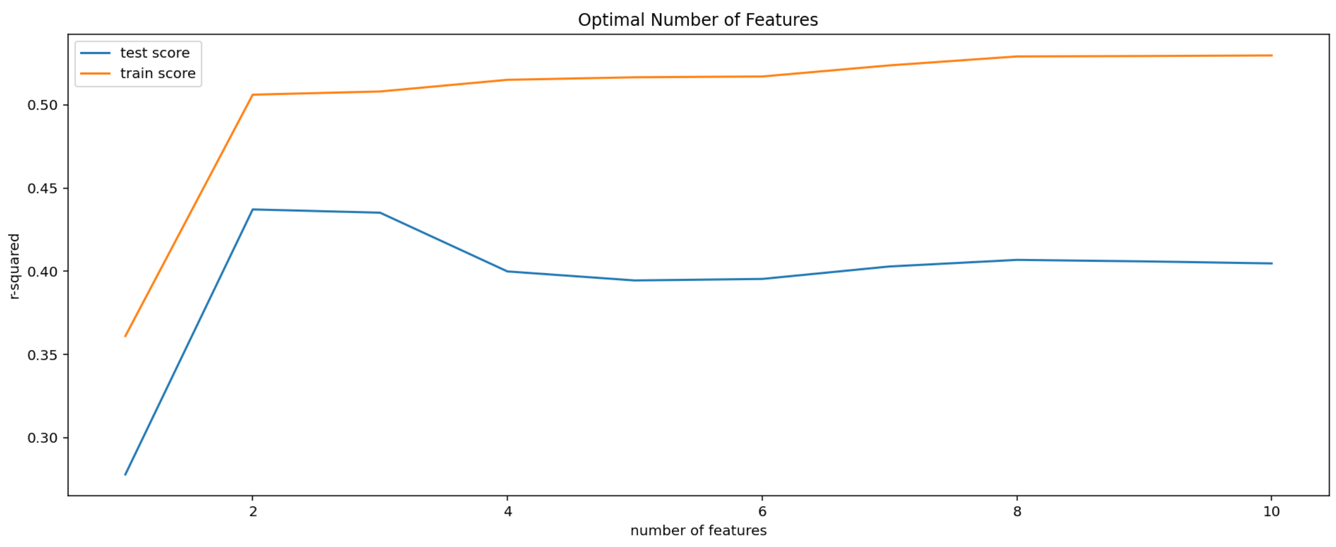
In addition to that, you can specify the number of times for the cross-validation for each set of hyperparameters.

Then all you have to do is create an object of GridSearchCV. Here basically you need to define a few named arguments:

- **estimator**: estimator object you created
- **params_grid**: the dictionary object that holds the hyperparameters you want to try
- **scoring**: evaluation metric that you want to use, you can simply pass a valid string/ object of evaluation metric
- **cv**: number of cross-validation you have to try for each selected set of hyperparameters
- **verbose**: you can set it to 1 to get the detailed print out while you fit the data to GridSearchCV
- **n_jobs**: number of processes you wish to run in parallel for this task if it -1 it will use all available processors.



```
In [22]: # step-1: create a cross-validation scheme
         folds = KFold(n_splits = 5, shuffle = False, random_state = 80)

         # step-2: specify range of hyperparameters to tune
         hyper_params = [{'n_features_to_select': list(range(1, 11))}]


         # step-3: perform grid search
         # 3.1 specify model
         lm = LinearRegression()
         lm.fit(X_train, y_train)
         # Optional
         rfe = RFE(lm)    # Recursive Feature Elimination
         #sfs = SFS(lm)    # Sequential Feature Selector

         # 3.2 call GridSearchCV()
         model_cv = GridSearchCV(estimator = rfe,
                                 param_grid = hyper_params,
                                 scoring= 'r2',
                                 cv = folds,
                                 verbose = 1,
                                 return_train_score=True)

         # fit the model
         model_cv.fit(X_train, y_train)

         Fitting 5 folds for each of 10 candidates, totalling 50 fits

         [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
         [Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    0.5s finished

Out[22]: GridSearchCV(cv=KFold(n_splits=5, random_state=80, shuffle=False),
                      estimator=RFE(estimator=LinearRegression()),
                      param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                            10]}],
                      return_train_score=True, scoring='r2', verbose=1)

In [23]: model_cv.best_params_

Out[23]: {'n_features_to_select': 2}
```

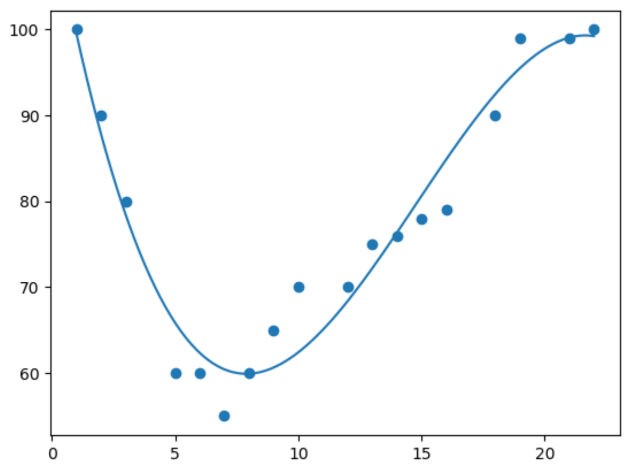


```
R squared of Training Set: 50.66
R squared of Test Set: 65.40
Mean Absolute Error (MAE): 613.4758
Root Mean Squared Error (RMSE): 945.3093
```

5.Polynomial regression (use degree = 2)

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.



Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modelled as an nth degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x)

Train Test Split

This step we will separate data to train (training set) and test (testing set)

- training set use for train model
- testing set use for test model or call that Evaluation

```
In [17]: from sklearn.model_selection import train_test_split

         # Train dataset X0% and Test dataset X0%.
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=17)
```

```
In [18]: X_train, y_train = np.array(X_train), np.array(y_train)
         X_test, y_test = np.array(X_test), np.array(y_test)
```

Creating and Training the Model

```
In [19]: from sklearn.linear_model import LinearRegression
         from sklearn.preprocessing import PolynomialFeatures

         X_train_poly = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X_train)
         X_test_poly = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X_test)
```

```
In [20]: # Try to use simple model: Linear Regression.
         model = LinearRegression().fit(X_train_poly, y_train)
```

Prediction Result

```
In [27]: #Actual value and the predicted value
         mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': predictions})
```

```
In [28]: # Show 21 samples to compare Actual value and the predicted value
         result = pd.concat([pv,years,mlr_diff],axis=1)
         result.head(21)
```

Out[28]:

	Province	Year	Actual value	Predicted value
0	Pattalung	2016	837	670.859924
1	Nakornsrithamarat	2020	3419	4024.796426
2	Pang-Nga	2019	929	1467.842953
3	Surin	2019	3600	2796.975016
4	Rayong	2020	1155	853.749229
5	Prachinburi	2017	295	442.073246
6	Udonthani	2018	1542	1350.976802
7	Songkhla	2017	7314	7472.123127
8	Lamphoon	2017	271	-244.214709
9	Nongbualamphu	2016	1557	845.457907
10	Samutsongkram	2018	368	221.065791
11	Nakornrachasrima	2016	7194	9131.088890
12	Ubonrachathani	2020	2092	2635.773511
13	Kanjanaburi	2020	1491	1769.509950
14	Trang	2020	3211	3225.647126
15	Nakornprathom	2017	580	551.858753
16	Nakornprathom	2018	1950	1488.552027
17	Narathiwat	2017	5770	4025.746853
18	Amnajcharean	2020	409	1911.091981
19	Singburi	2020	790	482.214393
20	Pattalung	2017	1821	1323.725309

```
R squared of Training Set: 83.42
R squared of Test Set: 78.01
Mean Absolute Error (MAE): 544.9456
Root Mean Squared Error (RMSE): 753.6509
```

Evaluation metrics

Evaluation metrics for regression problems as below:

Coefficient of determination (R2 score) is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Mean Absolute Error (MAE): the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Root Mean Squared Error (RMSE): the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Comparison:

- **R2 Score of Training set**
- **R2 Score of Testing set**
- **MAE** is the easiest to understand, because it's the average error.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

Conclusion

From the table as below, we will find result of Model testing in each model.

Method	Model	R2 score(Train)	R2 score(Test)	RMSE	MAE
1	Linear Regression (original)	52.46	64.26	960.83	621.12
2	Repeat K-folds Cross Validation	55.25	57.07	998.24	676.5
3	Recursive Feature Elimination (RFE)	51.28	65.76	940.36	618.64
4	Grid Search CV	50.66	65.4	945.31	613.48
5	Polynomial Regression	83.42	78.01	753.65	544.95

For the results in this section, we used the default settings for all machine learning approaches imported from the sklearn package.

Overall, we have experimented with various machine learning approaches in predicting dengue patient for each province that have difference of environment condition. We showed that polynomial out-perform other approaches and achieve r-squared score greater than 0.78. In future work, we would like to improve our collecting data and add more factor that more correlate to dependent variable for greater and reliable result in prediction.