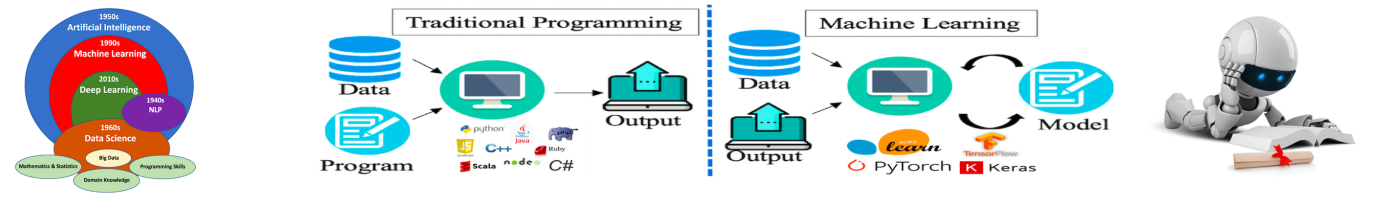
Department of Data Science

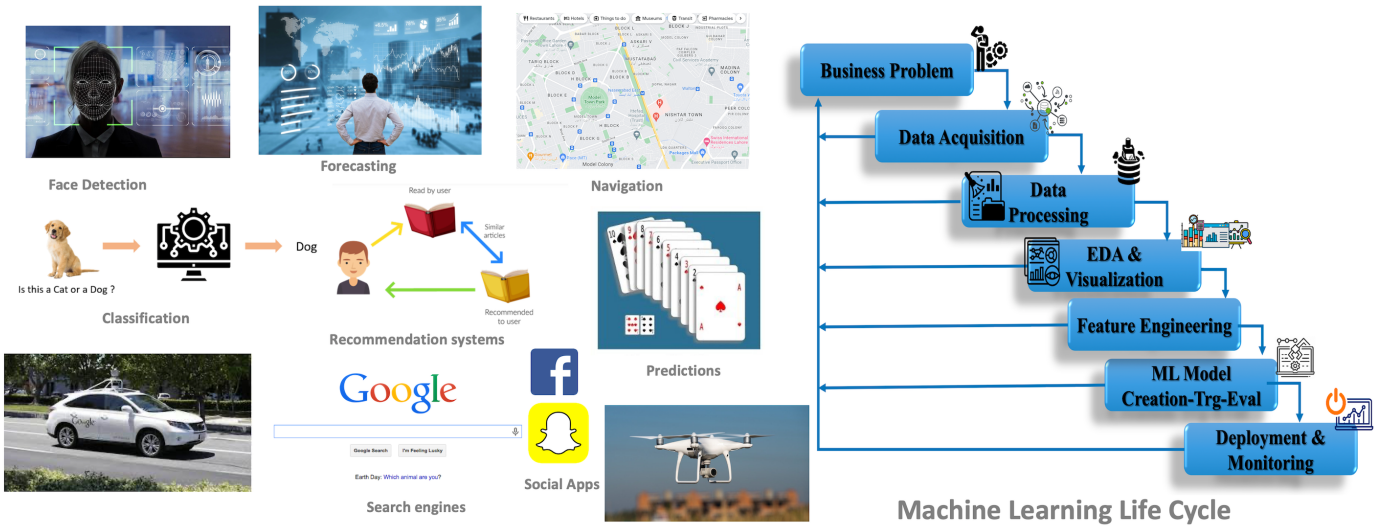
Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.14 (Cross Validation Techniques)

Open in Colab
(https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)





Learning agenda of this notebook

https://scikit-learn.org/stable/modules/cross_validation.html (https://scikit-learn.org/stable/modules/cross_validation.html)

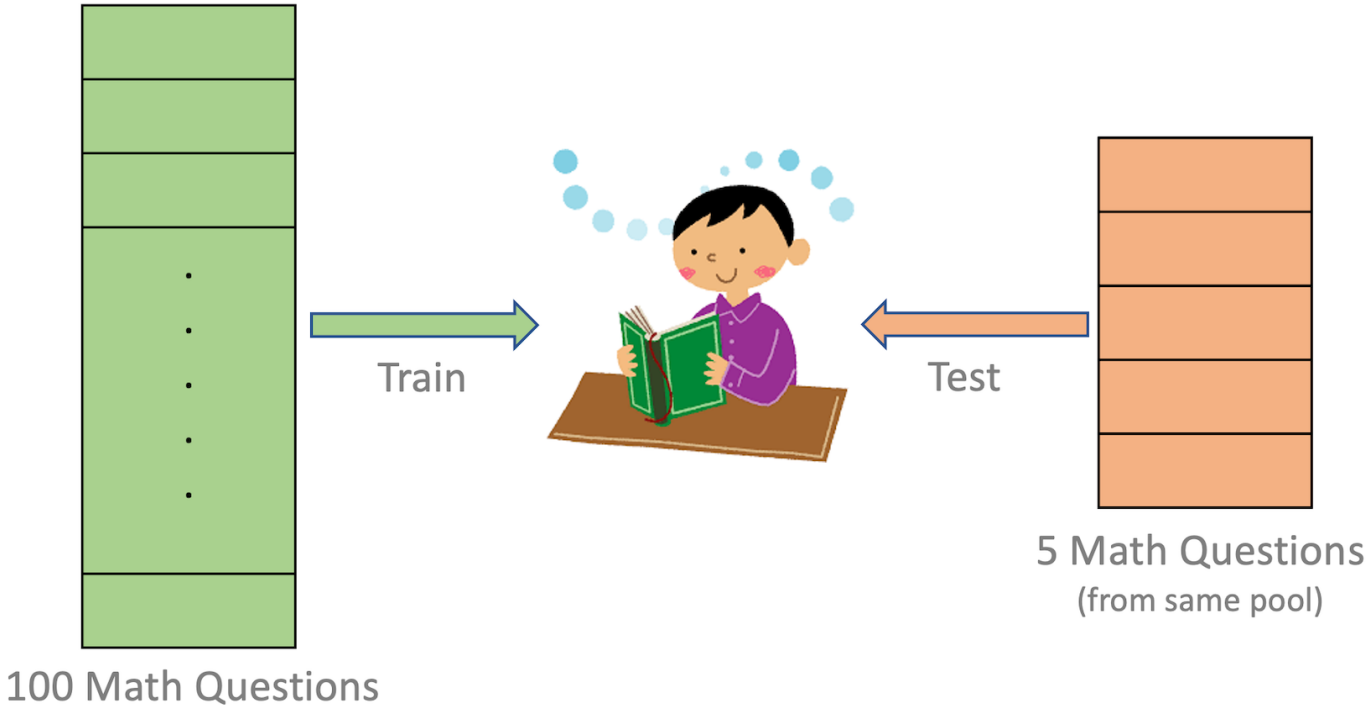
- Recap of Evaluating a Machine Learning Model
  - No Splitting of data
  - Train-Test-Split Method (Hold-out Method)

- KFold Cross Validation
- StratifiedKFold Cross Validation
- TimeSeriesSplit Cross Validation

1. Recap of Evaluating a Machine Learning Model

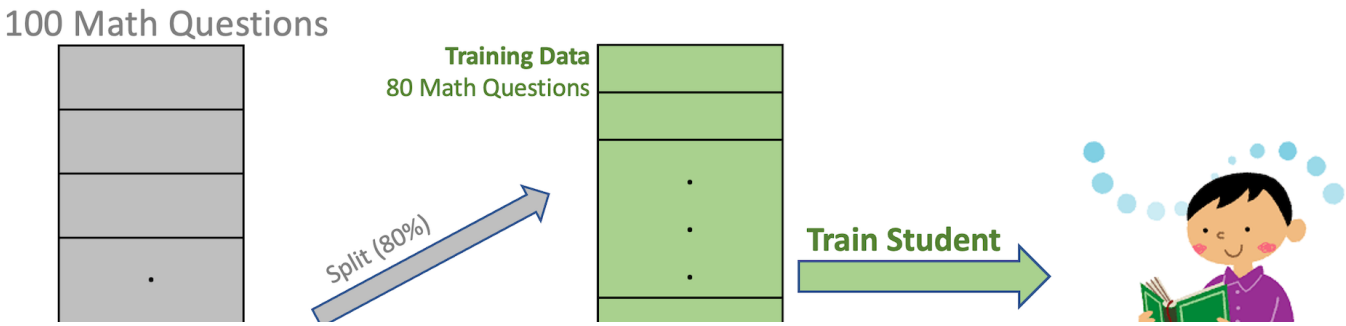
a. A Bad Approach

A terrible approach is to use all of the available data to train the model and then test the model using the same data



b. Train-Test-Split Method

- Holdout cross validation method involves removing a certain portion of data and using it as test data.
- The machine learning model is trained on the training data and then asked to predict the output on the test data



Basic Example

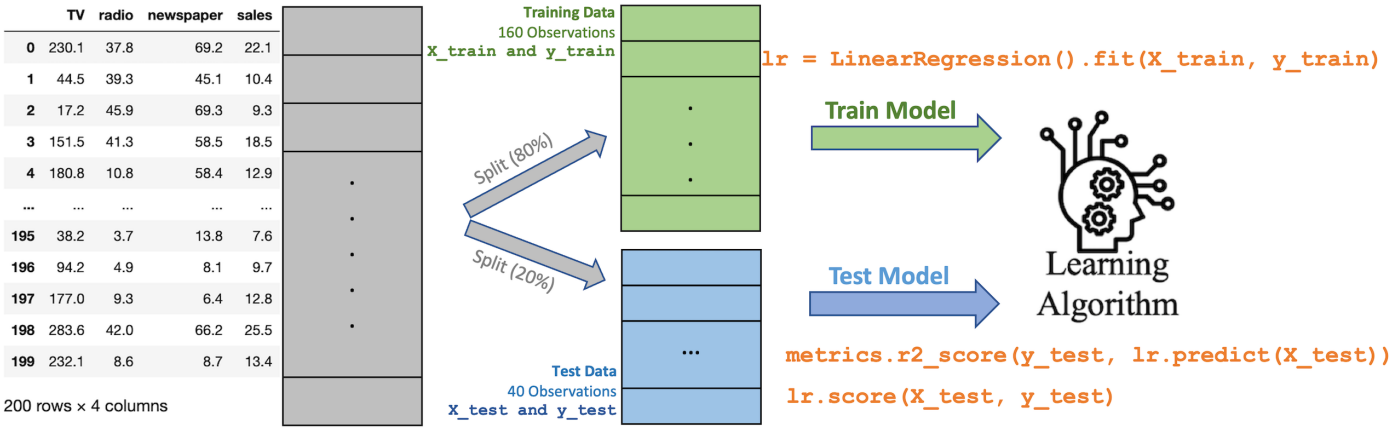
In [1]:

```
from sklearn.model_selection import train_test_split
import numpy as np
X = np.arange(1,101)
X_train, X_test = train_test_split(X, test_size=0.2, shuffle=True)
print("Available Data:\n", X)
print("\nTraining Data:\n",X_train)
print("Test Data: \n", X_test)
```

```
Available Data:
 [  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
90
  91  92  93  94  95  96  97  98  99 100]

Training Data:
 [ 55  52  56  47  51  16  74   5  29  95  25  59   3  49   8  66  31
82
  46  12  83  98  18  27  36  13  92  64  77  80  67  10  44  19  41
54
  57  43  84   2  65  42  76  97  48   4   7  70  90  88  72  23  11
32
  35  89  20  53  38  30  22  40  86  60  75  17  78  26  71 100  73
21
  68  93  96  63  14  34  58  24]
Test Data:
 [91 87 81 85 37 28 62 69 99  6 33 61 45 94 79 15 39  9  1 50]
```

Example (Advertising Dataset)




```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
```

In [2]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
df = pd.read_csv("datasets/advertising4D.csv")
X = df.drop('sales', axis=1)
y = df['sales']
# Do a train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle
# Instentiate a model and fit it to training data
lr = LinearRegression().fit(X_train, y_train)
# Evaluate
r2 = r2_score(y_test, lr.predict(X_test))
r2 = lr.score(X_test, y_test)
print("R2 Score: ", r2)
```

```
R2 Score:  0.8494021694523123
```

Limitations of Train-Test-Split:

- We are only evaluating our model on 20% of the data rather we should evaluate out model on the remaining 80% of data as well.
- Suppose you have a sample of 1000 students out of 10 million students of Pakistan. Out of 10 million students of Pakistan, the accuracy that you are getting after evaluating just 200 students might not be a true representation of the accuracy of the entire population. So even after a train-test-split, your model may overfit.
- So we want some thing more robust, let us move on to KFold Cross Validation

2. KFold Cross Validation

Instead of splitting data set into a single train-test set, the data is repeatedly split into multiple train-test sets and then the model is trained and evaluated on those multiple sets

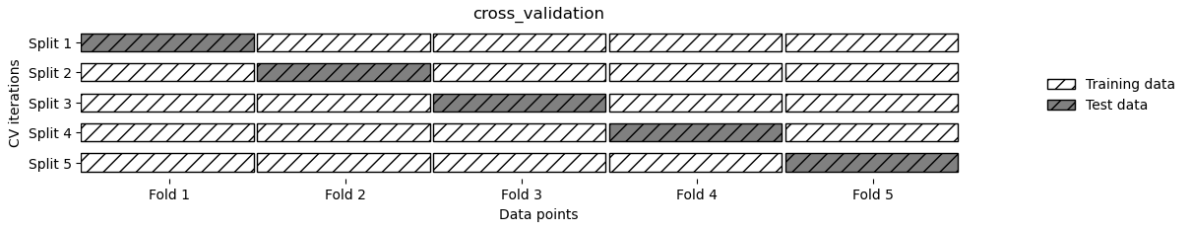
a. Overview

A Helper Library `mglearn` : https://github.com/amueller/mglearn (https://github.com/amueller/mglearn)

- The book "Introduction to Machine Learning with Python" by Andreas Mueller comes handy with a helper library, which includes some utility functions that just help to draw a pretty picture quickly or to get some interesting data.

In [3]:

```
#import sys
#!{sys.executable} -m pip install mglearn graphviz -q
import mglearn
mglearn.plots.plot_cross_validation()
```



Example: KFold Cross-Validator

- Split dataset into k consecutive folds (without shuffling by default).
- Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

In [15]:

```
from sklearn.model_selection import KFold
X=[0,1,2,3,4,5,6,7,8,9]

kf = KFold(n_splits=5, shuffle=False)  #change value of n_splits (2-10)

# split() method returns a generator object containing indices to split data int
for train_indices, test_indices in kf.split(X):
    print(train_indices, test_indices)
```

```
[2 3 4 5 6 7 8 9] [0 1]
[0 1 4 5 6 7 8 9] [2 3]
[0 1 2 3 6 7 8 9] [4 5]
[0 1 2 3 4 5 8 9] [6 7]
[0 1 2 3 4 5 6 7] [8 9]
```

b. KFold on Advertising Dataset

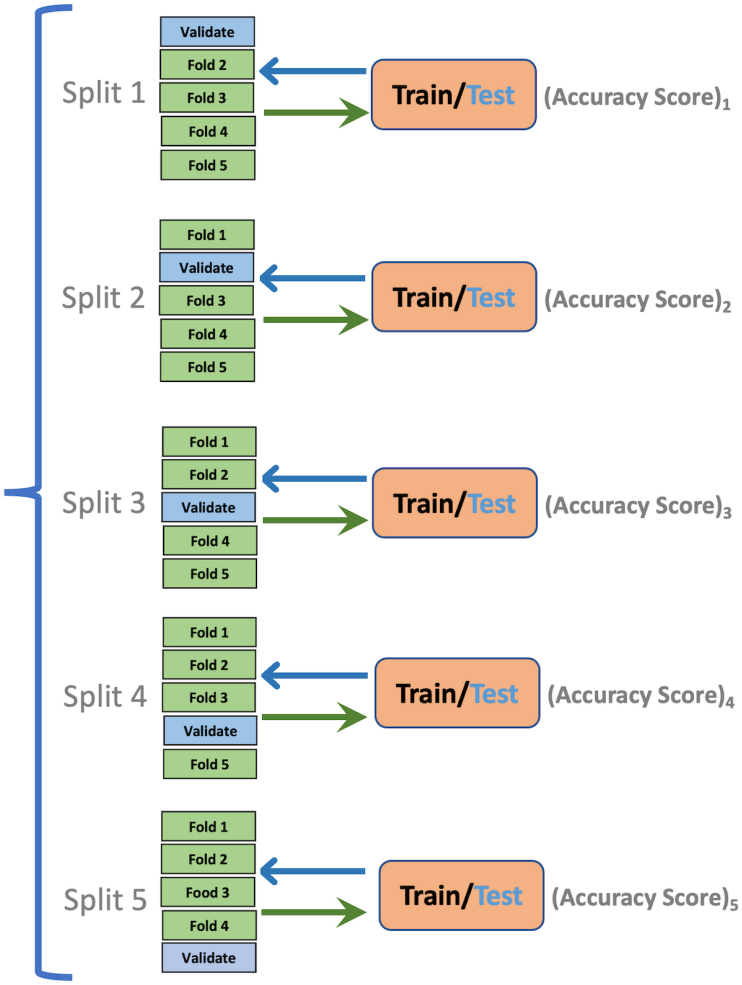
```
kf = KFold(n_splits=5, shuffle=True)
for train_i, test_i in kf.split(df):
    ...
    ...
```

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

```
lr = LinearRegression()
cross_val_score(lr, X, y, cv=5)
```

```
[0.878   0.917   0.929   0.814   0.895]
```



- **Typical Steps of KFold Cross Validation:**
  - The number of folds is defined, by default this is 5
  - The dataset is split up according to these folds, where each fold has a unique set of testing data
  - A model is trained and tested for each fold
  - Each fold returns a metric/s for it's test data
  - The mean of these metrics can then be calculated to provide a single metric for the process

Example: Do a Five Fold Split using `KFold` on Advertising Dataset

In [16]:

```
df = pd.read_csv("datasets/advertising4D.csv")
X = df.drop('sales', axis=1)
y = df['sales']
kf = KFold(n_splits=5, shuffle=False)
i=1
for train_indices, test_indices in kf.split(X, y):
    print("\nSplit {}".format(i))
    print("Test Indices: ", test_indices)
    print("Training Indices: ", train_indices)
    i=i+1
```

```
Split 1
Test Indices:  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]
Training Indices:  [ 40  41  42  43  44  45  46  47  48  49  50  51  5
2  53  54  55  56  57
  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74
75
  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92
93
  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 1
11
 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 1
29
 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 1
47
 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 1
65
 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 1
83
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199]

Split 2
Test Indices:  [40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63
 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79]
Training Indices:  [  0   1   2   3   4   5   6   7   8   9  10  11  1
2  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
35
  36  37  38  39  80  81  82  83  84  85  86  87  88  89  90  91  92
93
  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 1
11
 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 1
29
 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 1
47
 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 1
65
 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 1
83
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199]

Split 3
Test Indices:  [ 80  81  82  83  84  85  86  87  88  89  90  91  92  9
3  94  95  96  97
  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 1
15
 116 117 118 119]
Training Indices:  [  0   1   2   3   4   5   6   7   8   9  10  11  1
2  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
71
  72  73  74  75  76  77  78  79 120 121 122 123 124 125 126 127 128 1
29
```

```
 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 1
47
 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 1
65
 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 1
83
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199]

Split 4
Test Indices:  [120 121 122 123 124 125 126 127 128 129 130 131 132 13
3 134 135 136 137
 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 1
55
 156 157 158 159]
Training Indices:  [  0   1   2   3   4   5   6   7   8   9  10  11  1
2  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 1
07
 108 109 110 111 112 113 114 115 116 117 118 119 160 161 162 163 164 1
65
 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 1
83
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199]

Split 5
Test Indices:  [160 161 162 163 164 165 166 167 168 169 170 171 172 17
3 174 175 176 177
 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 1
95
 196 197 198 199]
Training Indices:  [  0   1   2   3   4   5   6   7   8   9  10  11  1
2  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 1
07
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 1
25
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 1
43
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159]
```

Calculate R2 Score for Linear Regression Model on all these five splits (Mannually)

In [21]:

```
# A user defined function, which is passed model intance along with training and
# The model is trained on the training data and the function return the R2 score
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)
```

In [18]:

```
from sklearn.linear_model import LinearRegression
df = pd.read_csv("datasets/advertising4D.csv")
X = df.drop('sales', axis=1)
y = df['sales']
kf = KFold(n_splits=5, shuffle=False)

scores_list=[]
for train_indices, test_indices in kf.split(X, y):
    X_train,X_test,y_train,y_test = X.loc[train_indices], X.loc[test_indices], y
    scores_list.append(get_score(LinearRegression(), X_train, X_test, y_train, y
scores_list
print("R2 Scores: ", scores_list)
print("Mean R2 Score: ", np.mean(scores_list))
```

```
R2 Scores:  [0.8786519804831341, 0.9176321165614463, 0.929330323579965
3, 0.8144390391722337, 0.8954782879224386]
Mean R2 Score:  0.8871063495438435
```

Calculate R2 Score for Linear Regression Model on all the five splits (`cross_val_score`)

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)

- The `cross_val_score()` method is used to evaluate a model by cross validation.

`cross_val_score(estimator, X, y=None, cv=None, scoring=None)`

- Where
  - `estimator` is the instance of machine learning model
  - `X` is an array of shape (n_samples, n_features) on which the model is to be trained
  - `y` is an array of shape (n_samples,), which is the target variable
  - `cv` can be an integer value specifying the number of folds for KFold. Default is 5-fold cross validation. You can also mention the cross-validation generator that determines the cross-validation splitting strategy
  - `scoring` default value is None, in which case it uses the model's default scoring metric. But this can be overridden in the scoring parameter by passing a single string. You can pass a string specifying the metric of your choice like 'r2', 'max_error', 'neg_root_mean_squared_error',...
- The return value is an array of scores of the estimator for each run of the cross validation

> For Details of `scoring` parameters visit: https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-

In [29]:

```
from sklearn.model_selection import cross_val_score
df = pd.read_csv("datasets/advertising4D.csv")
X = df.drop('sales', axis=1)
y = df['sales']

#kf = KFold(n_splits=5, shuffle=False)
#scores_array = cross_val_score(LinearRegression(), X, y, cv=kf)

scores_array = cross_val_score(LinearRegression(), X, y, cv=5, scoring='neg_root

print("R2 Scores: ", scores_array)
print("Mean R2 Score: ", np.mean(scores_array))
```

```
R2 Scores:  [-1.77102792 -1.55745554 -1.25905722 -2.32941088 -1.670672
08]
Mean R2 Score:  -1.7175247278732084
```

Calculate R2 Score for Linear Regression Model on all the five splits (`cross_validate`)

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html)

- The `cross_validate()` method differs from `cross_val_score()` in two ways:
  - It allows specifying multiple scoring metrics for evaluation.
  - It returns a dict containing fit-times, score-times (and optionally training scores as well as fitted estimators) in addition to the test score.

`cross_val_score(estimator, X, y=None, cv=None, scoring=None, return_train_score=True, return_estimator=True)`

- Where
  - `estimator` is the instance of machine learning model
  - `X` is an array of shape (n_samples, n_features) on which the model is to be trained
  - `y` is an array of shape (n_samples,), which is the target variable
  - `cv` can be an integer value specifying the number of folds for KFold. Default is 5-fold cross validation. You can also mention the cross-validation generator that determines the cross-validation splitting strategy
  - `scoring` default value is None, in which case it uses the model's default scoring metric. But this can be overridden in the scoring parameter by specifying a single string or list of strings of your choice like 'r2', 'max_error', 'neg_root_mean_squared_error',...
  - `return_train_score=False` , can set it to `True` to include the train scores as well
  - `return_estimator=False` , can set it to `True` to return the estimator fitted on each split
- The return value is a dictionary of arrays containing the score/time arrays for each scorer. The possible keys of the dictionary are test_score, train_score, fit_time, score_time, estimator

> For Details of `scoring` parameters visit: https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

In [31]:

```
from sklearn.model_selection import cross_validate
df = pd.read_csv("datasets/advertising4D.csv")
X = df.drop('sales', axis=1)
y = df['sales']

kf = KFold(n_splits=5, shuffle=False)
result_dict = cross_validate(LinearRegression(), X, y, cv=kf)
result_dict = cross_validate(LinearRegression(), X, y, cv=kf, return_train_score
print(result_dict)
```

```
{'fit_time': array([0.00244617, 0.00239205, 0.00956607, 0.00208807, 0.
00266814]), 'score_time': array([0.00109887, 0.0011301 , 0.00099397,
0.00140285, 0.00088382]), 'estimator': [LinearRegression(), LinearRegr
ession(), LinearRegression(), LinearRegression(), LinearRegression()],
'test_score': array([0.87865198, 0.91763212, 0.92933032, 0.81443904,
0.89547829]), 'train_score': array([0.90101302, 0.89039598, 0.8896931
6, 0.91458801, 0.89615232])}
```

In [32]:

```
1 pd.DataFrame(result_dict)
```

Out[32]:

	fit_time	score_time	estimator	test_score	train_score
0	0.002446	0.001099	LinearRegression()	0.878652	0.901013
1	0.002392	0.001130	LinearRegression()	0.917632	0.890396
2	0.009566	0.000994	LinearRegression()	0.929330	0.889693
3	0.002088	0.001403	LinearRegression()	0.814439	0.914588
4	0.002668	0.000884	LinearRegression()	0.895478	0.896152

The `scoring` argument of `cross_validate()` method:

- By default `cross_val_score()` and `cross_validate()` methods uses the chosen model's default scoring metric, but this can be overridden by specifying the scoring metric in the `scoring` parameter. For `cross_val_score()` this can be a single scoring metric, while for `cross_validate()` you can pass either a single metric or multiple as a list to the `scoring` argument.
- For Details of `scoring` parameters visit: https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)

In [33]:

```
1 from sklearn.model_selection import cross_validate
2 df = pd.read_csv("datasets/advertising4D.csv")
3 X = df.drop('sales', axis=1)
4 y = df['sales']
5 lr = LinearRegression()
6
7 kf = KFold(n_splits=5, shuffle=False)
8 result_dict = cross_validate(lr, X, y, cv=kf, scoring=['r2', 'neg_mean_absolute_
9 pd.DataFrame(result_dict)
```

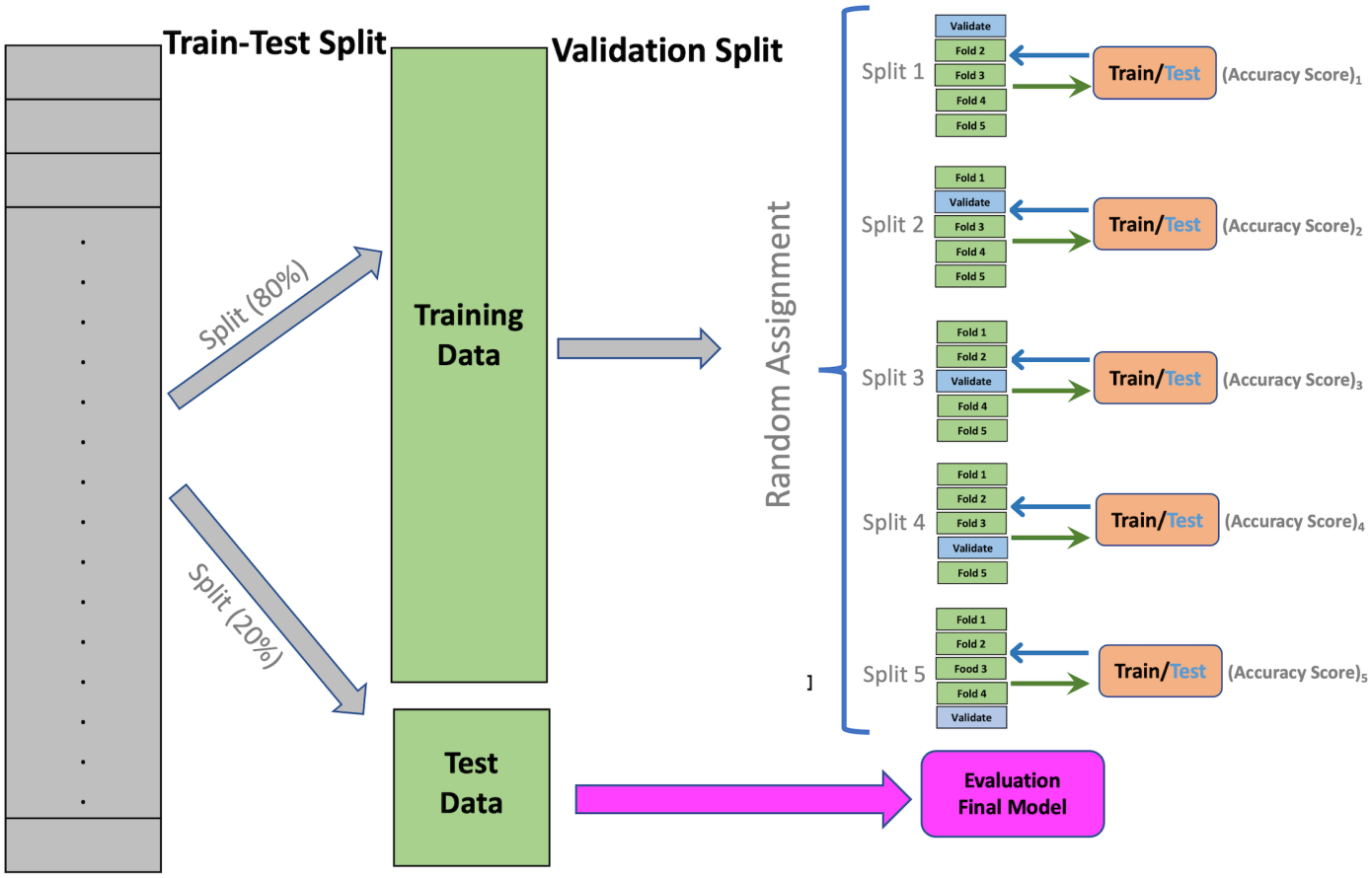
Out[33]:

	fit_time	score_time	test_r2	test_neg_mean_absolute_error
0	0.004331	0.001673	0.878652	-1.365488
1	0.002673	0.001313	0.917632	-1.292532
2	0.001930	0.001138	0.929330	-0.947893
3	0.005626	0.000941	0.814439	-1.631537
4	0.001402	0.000950	0.895478	-1.287718

In [ ]:

```
1
```

c. Extension to K-Fold Cross Validation



3. StratifiedKFold Cross Validation

a. Overview

- The standard K-Fold Cross Validation works perfectly fine for all regression tasks, however, for classification tasks it is better to use StratifiedKFold Cross Validation.
- For classification tasks, standard K-Fold Cross Validation may not work as expected for an imbalanced dataset or for a dataset having sorted class labels.
- So when we have an imbalanced data set, we need a slight change to the K-Fold cross validation technique, such that each fold contains approximately the same strata of samples of each output class as the complete.
- This variation of using a stratum in K-Fold Cross Validation is known as StratifiedKFold Cross Validation.

In [34]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
X = df.drop('target', axis=1)
y = df['target']
df
```

Out[34]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

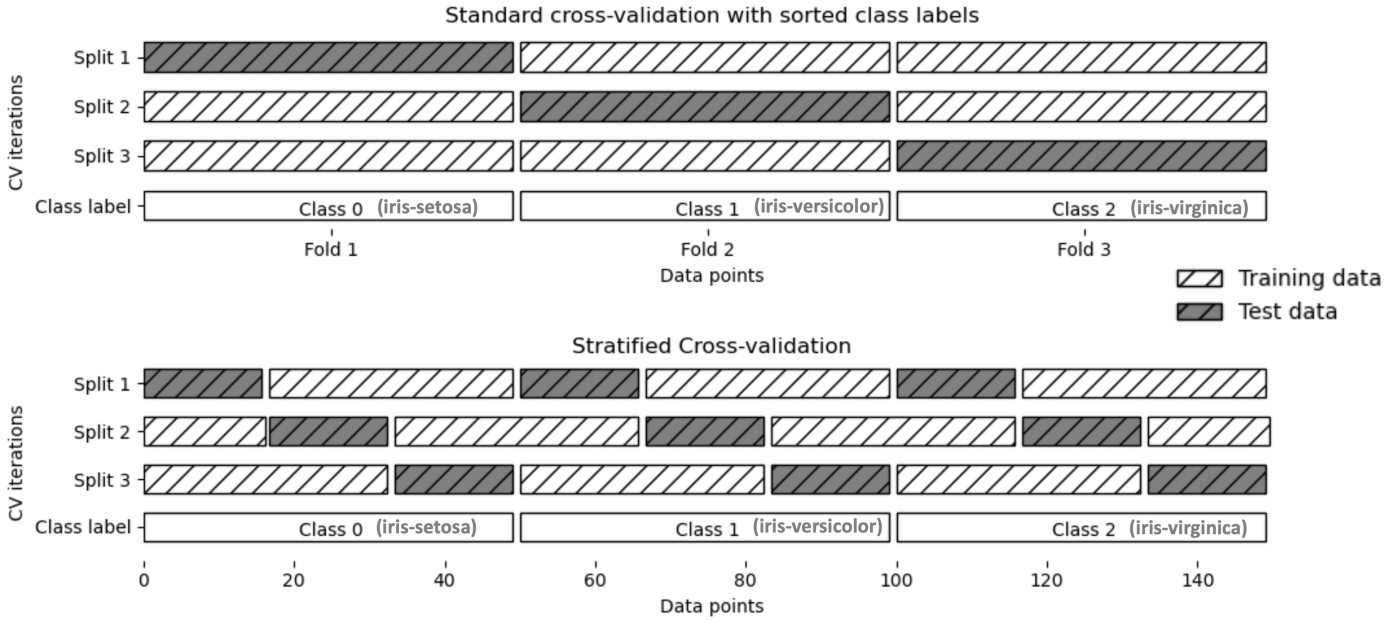
In [35]:

```
np.array(y)
```

Out[35]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
import mglearn
mglearn.plots.plot_stratified_cross_validation()
```



b. Example 1: (iris Dataset having sorted labels)

In [44]:

```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
X = df.drop('target', axis=1)
y = df['target']
```

Use `Kold` on iris Dataset (Mannually)

In [53]:

```
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)


kf = KFold(n_splits=3, shuffle=False) # set shuffle to True and see the results
scores_list=[]
for train_indices, test_indices in kf.split(X, y):
    X_train,X_test,y_train,y_test = X.loc[train_indices], X.loc[test_indices], y
    scores_list.append(get_score(LogisticRegression(solver='liblinear'), X_train

print("R2 Scores: ", scores_list)
```

```
R2 Scores:  [0.0, 0.0, 0.0]
```

Use `Kold` on iris Dataset (`cross_val_score`)

In [57]:

```
kf = KFold(n_splits=3, shuffle=False) # set shuffle to True and see the results
scores_array = cross_val_score(LogisticRegression(solver='liblinear'), X, y, cv=
print("R2 Scores: ", scores_array)
```

```
R2 Scores:  [0. 0. 0.]
```

In [55]:

```
# For regression estimator, if cv argument is set to an integer value the cross_
# KFold by default with shuffle argument set to True
scores_array = cross_val_score(LogisticRegression(solver='liblinear'), X, y, cv=
print("R2 Scores: ", scores_array)
```

```
R2 Scores:  [0.96 0.96 0.94]
```

Use `StratifiedKFold` on iris Dataset (Mannually)

In [58]:

```
skf = StratifiedKFold(n_splits=3, shuffle=False)

scores_list=[]
for train_indices, test_indices in skf.split(X, y):
    X_train,X_test,y_train,y_test = X.loc[train_indices], X.loc[test_indices], y
    scores_list.append(get_score(LogisticRegression(solver='liblinear'), X_train

print("R2 Scores: ", scores_list)
```

```
R2 Scores:  [0.96, 0.96, 0.94]
```

Use `StratifiedKFold` on iris Dataset (`cross_val_score`)

In [59]:

```
skf = StratifiedKFold(n_splits=3, shuffle=False)
scores_array = cross_val_score(LogisticRegression(solver='liblinear'), X, y, cv=

print("R2 Scores: ", scores_array)
```

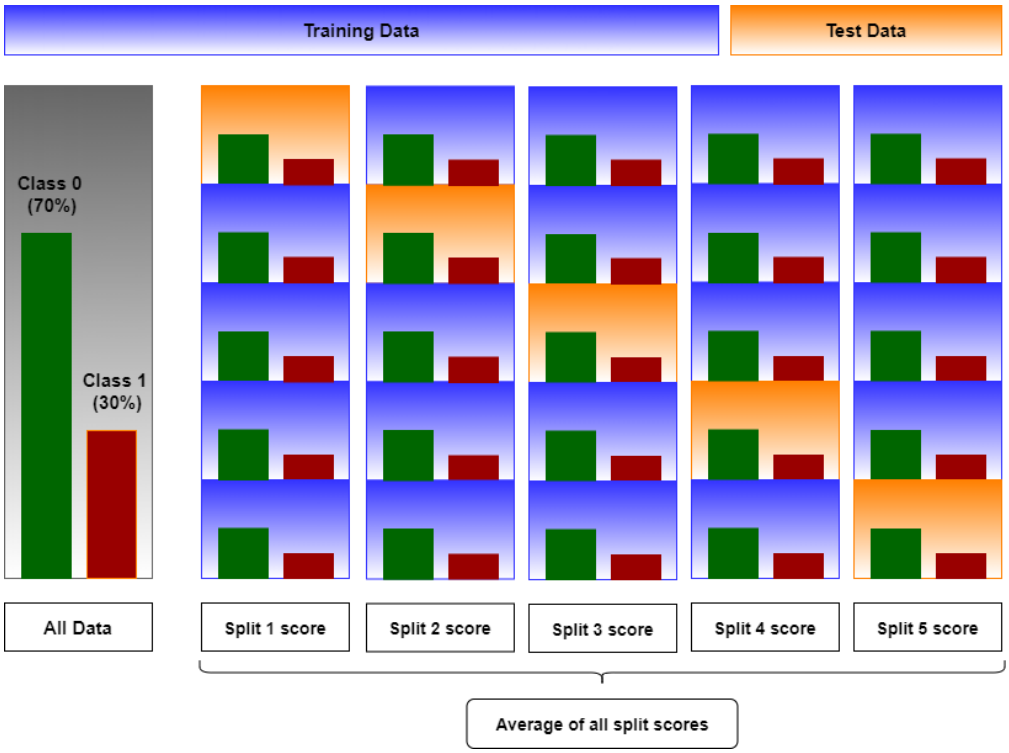
```
R2 Scores:  [0.96 0.96 0.94]
```

In [61]:

```
# For classification estimator, if cv argument is set to an integer value the cr
# StratifiedKFold by default with shuffle argument set to True
scores_array = cross_val_score(LogisticRegression(solver='liblinear'), X, y, cv=
print("R2 Scores: ", scores_array)
```

```
R2 Scores:  [0.96 0.96 0.94]
```

c. Example 2: (Synthetic Imbalanced Dataset)



Generate an Imbalanced Synthetic Dataset:

In [66]:

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000,     #number of observations
                           n_features=4,       #number of input features
                           n_classes=2,        #binary classification
                           weights=[0.95, 0.05]#95% datapoints belong to one cla
                          )
```

In [67]:

```
X
```

Out[67]:

```
array([[-0.62871392, -0.78732399,  0.66479493,  0.56622553],
       [ 0.26463575, -1.51458858,  2.12585599, -2.92695326],
       [-0.60716024, -1.41204343,  1.49131008, -0.40238178],
       ...,
       [ 1.04891116,  1.47181318, -1.31538396, -0.71412071],
       [-0.92619275, -0.68617683,  0.36205747,  1.52402749],
       [-0.70972432, -1.10427417,  1.03129639,  0.32531136]])
```

In [68]:

```
y[:100]
```

Out[68]:

```
array([0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [69]:

```
len(y[y==0])
```

Out[69]:

```
945
```

In [70]:

```
len(y[y==1])
```

Out[70]:

```
55
```

Split the Dataset using `KFold` :

In [75]:

```
kf = KFold(n_splits=8, shuffle=True, random_state=54) #perform 8 splits
for train_indices, test_indices in kf.split(X,y):
    X_train, X_test, y_train, y_test = X[train_indices], X[test_indices], y[trai
    train_0, train_1 = len(y_train[y_train==0]), len(y_train[y_train==1])
    test_0, test_1 = len(y_test[y_test==0]), len(y_test[y_test==1])
    print('Train: Class0=%d, Class1=%d,        Test: Class0=%d, Class1=%d' % (tr
```

```
Train: Class0=823, Class1=52,        Test: Class0=122, Class1=3
Train: Class0=824, Class1=51,        Test: Class0=121, Class1=4
Train: Class0=824, Class1=51,        Test: Class0=121, Class1=4
Train: Class0=828, Class1=47,        Test: Class0=117, Class1=8
Train: Class0=831, Class1=44,        Test: Class0=114, Class1=11
Train: Class0=831, Class1=44,        Test: Class0=114, Class1=11
Train: Class0=823, Class1=52,        Test: Class0=122, Class1=3
Train: Class0=831, Class1=44,        Test: Class0=114, Class1=11
```

> Using KFold the model accuracy will not be the proper reflection of how well the minority class is being predicted because we have different number of minority class observations in each split of the trining and test data sets.

Split the Dataset using `StratifiedKFold` :

In [72]:

```
skf = StratifiedKFold(n_splits=8, shuffle=True, random_state=54)
for train_indices, test_indices in skf.split(X,y):
    X_train, X_test, y_train, y_test = X[train_indices], X[test_indices], y[trai
    train_0, train_1 = len(y_train[y_train==0]), len(y_train[y_train==1])
    test_0, test_1 = len(y_test[y_test==0]), len(y_test[y_test==1])
    print('Train: Class0=%d, Class1=%d,        Test: Class0=%d, Class1=%d' % (tr
```

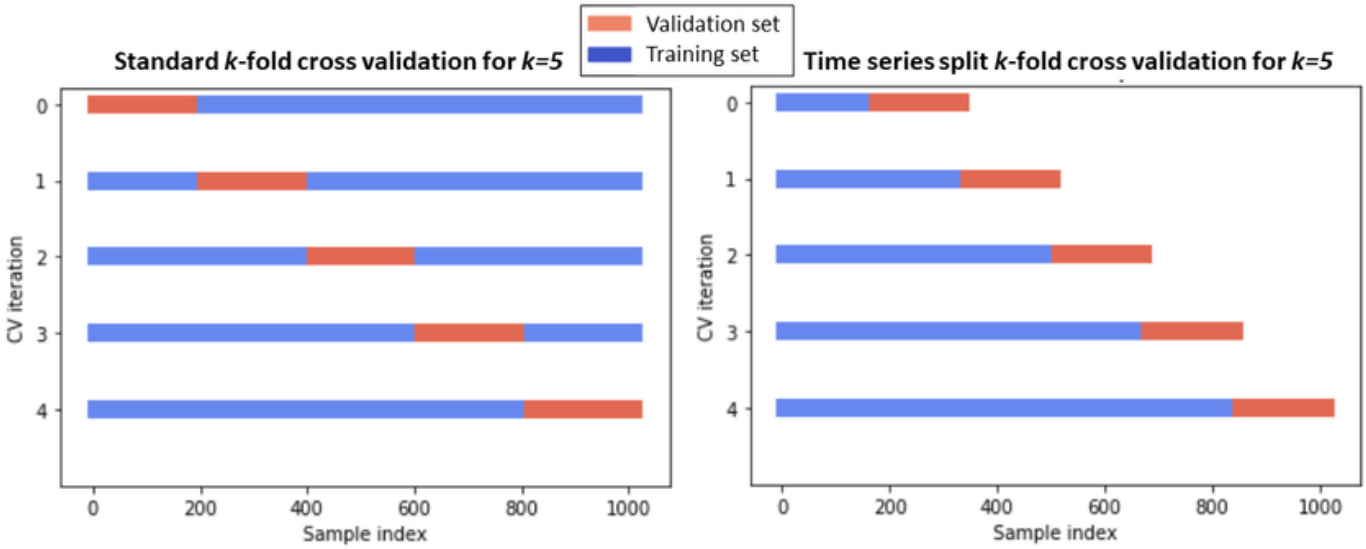
```
Train: Class0=826, Class1=49,        Test: Class0=119, Class1=6
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
Train: Class0=827, Class1=48,        Test: Class0=118, Class1=7
```

> Using StratifiedKFold the proportion between the classes are almost same in each fold as they are in the entire dataset. This ensures that one particular class is not over-present in the train or the test set

4. TimeSeriesSplit (Rolling Cross Validation)

Peaking in the future is not allowed

- Be watchfull when splitting a TimeSeries data. The techniques discussed above may not work in case of TimeSeries data.
- The best technique for splitting TimeSeries data is Rolling Cross Validation, which in Scikit-Learn is implemented in the `TimeSeriesSplit`
- Consider we have a TimeSeries dataset of say of 100 years.
- The left image use KFold to cross-validate timeseries data. Note that there is a chance that we train our model on future data and test on past data, thus breaking the golden rule in timeseries, i.e., "Peaking in the future is not allowed".
- The right image describe the use of `TimeSeriesSplit` , where we are creating the folds using forward chaining fashion by keeping the order intact.



In [80]:

```
from sklearn.model_selection import TimeSeriesSplit
X = np.array([[9,3,4], [6,1,7], [5,2,6], [0,2,1], [5,1,7], [3,9,4], [7,3,2],[9,8
tss = TimeSeriesSplit(n_splits=3)
i=1
for train_indices, test_indices in tss.split(X):
    print("Split{} : {}  {}".format(i, train_indices, test_indices))
    i=i+1
```

```
Split1 : [0 1 2 3]  [4 5]
Split2 : [0 1 2 3 4 5]  [6 7]
Split3 : [0 1 2 3 4 5 6 7]  [8 9]
```

In [81]:

```
tss = TimeSeriesSplit(n_splits=4)
i=1
for train_indices, test_indices in tss.split(X):
    print("Split{} : {}  {}".format(i, train_indices, test_indices))
    i=i+1
```

```
Split1 : [0 1]  [2 3]
Split2 : [0 1 2 3]  [4 5]
Split3 : [0 1 2 3 4 5]  [6 7]
Split4 : [0 1 2 3 4 5 6 7]  [8 9]
```

In [ ]:

```

```

Summary of Scikit-Learn's Cross Validation Iterators

- `KFold` divides all the samples in groups of samples, called folds (if , this is equivalent to the Leave One Out strategy), of equal sizes (if possible). The prediction function is learned using folds, and the fold left out is used for test.
- `RepeatedKFold` repeats K-Fold n times. It can be used when one requires to run KFold n times, producing different splits in each repetition.

- `LeaveOneOut (LOO)` is a simple cross-validation. Each learning set is created by taking all the samples except one, the test set being the sample left out. Thus, for samples, we have different training sets and different tests set. LOO is more computationally expensive than KFold cross validation.
- `LeavePOut` is very similar to LeaveOneOut as it creates all the possible training/test sets by removing samples from the complete set. Unlike LeaveOneOut and KFold, the test sets will overlap for $p > 1$.
- `ShuffleSplit` iterator will generate a user defined number of independent train / test dataset splits. Samples are first shuffled and then split into a pair of train and test sets.
- `StratifiedKFold` is a variation of k-fold which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set.
- `StratifiedShuffleSplit` is a variation of ShuffleSplit, which returns stratified splits, i.e which creates splits by preserving the same percentage for each target class as in the complete set.
- `GroupKFold` is a variation of k-fold which ensures that the same group is not represented in both testing and training sets. For example if the data is obtained from different subjects with several samples per-subject and if the model is flexible enough to learn from highly person specific features it could fail to generalize to new subjects. GroupKFold makes it possible to detect this kind of overfitting situations.
- `StratifiedGroupKFold` is a cross-validation scheme that combines both StratifiedKFold and GroupKFold. The idea is to try to preserve the distribution of classes in each split while keeping each group within a single split. That might be useful when you have an unbalanced dataset so that using just GroupKFold might produce skewed splits.
- `TimeSeriesSplit` is a variation of k-fold which returns first folds as train set and the $(K + 1)^{th}$ fold as test set. Note that unlike standard cross-validation methods, successive training sets are supersets of those that come before them. Also, it adds all surplus data to the first training partition, which is always used to train the model.