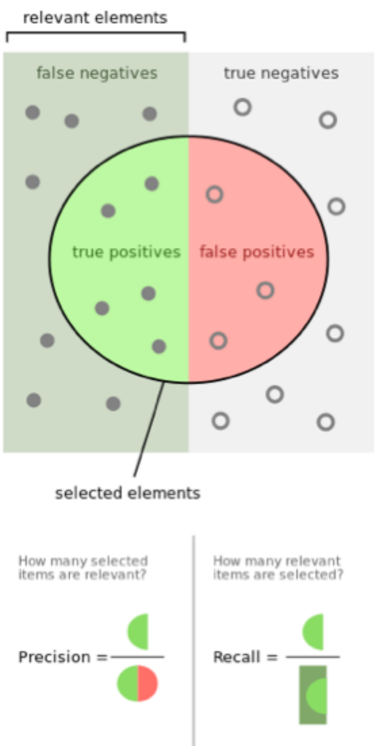
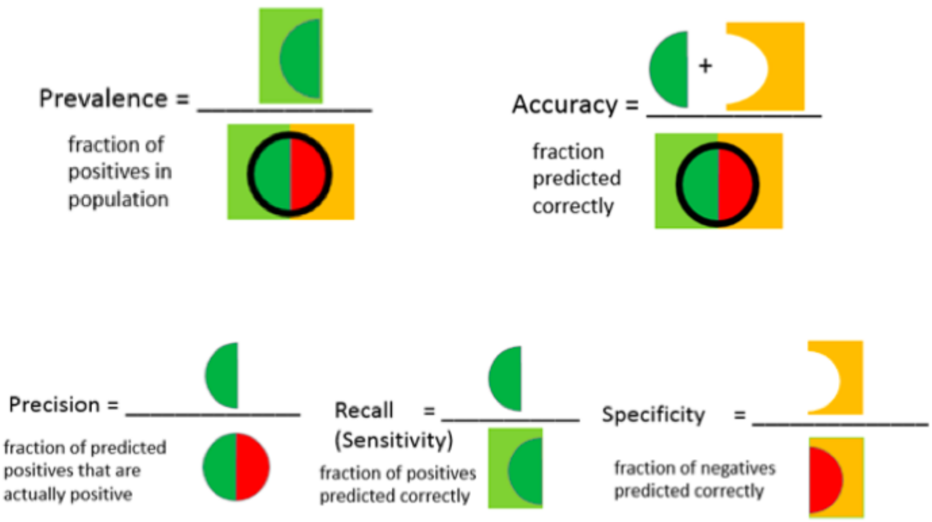
In [1]:

```
from IPython.core.interactiveshell import InteractiveShell

# This file includes the functions to clean the times and shangai datasets done in the Feat
import data_cleaning as dc

InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook")
#sns.set_context("poster")
```



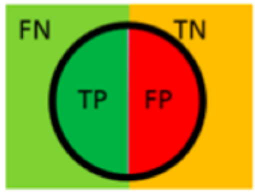


Patient	Probability	Actual
A	0.99	Hospital
B	0.97	Not Hosp
C	0.95	Hospital
Q	0.6	Not Hosp
R	0.58	Hospital
S	0.56	Not Hosp
Z	0.05	Not Hosp

Threshold

Patient	Probability	Actual
A	0.99	Hospital
B	0.97	Not Hosp
C	0.95	Hospital
Q	0.6	Not Hosp
R	0.58	Hospital
S	0.56	Not Hosp
Z	0.05	Not Hosp

Recall (Sensitivity)	2/(2+1) = 0.66	3/(3) = 1.0
Specificity	3/(3+1) = 0.75	2/(2+2) = 0.5
Accuracy	(2+3)/7 = .71	(3+2)/7 = .71
Precision	2/(2+1) = 0.66	3/(3+2) = 0.60



Performance Metrics

The choice of metrics for evaluating a model is of the utmost importance. We will tune the model, select hyperparameters and choose a particular algorithm among others, according to this metrics.

In this notebook we will focus on two of the main machine learning problems and we will discuss the metrics associated with them:

> **1) Classification problems.** We will continue to use the Pima Indias onset diabetes dataset that we have been using so far. In this dataset all attributes are numeric and its a binary classification problem.
>
> **2) Regression problems.** For regression we will use also a very traditional dataset, the Boston House Price. Again all input variables are numeric.

Obviously not all problems in machine learning can be categorized as regression or classification problems, we also have clustering, association rules, topic modeling, etc... However, classification and regression remain as the most substantial ones.

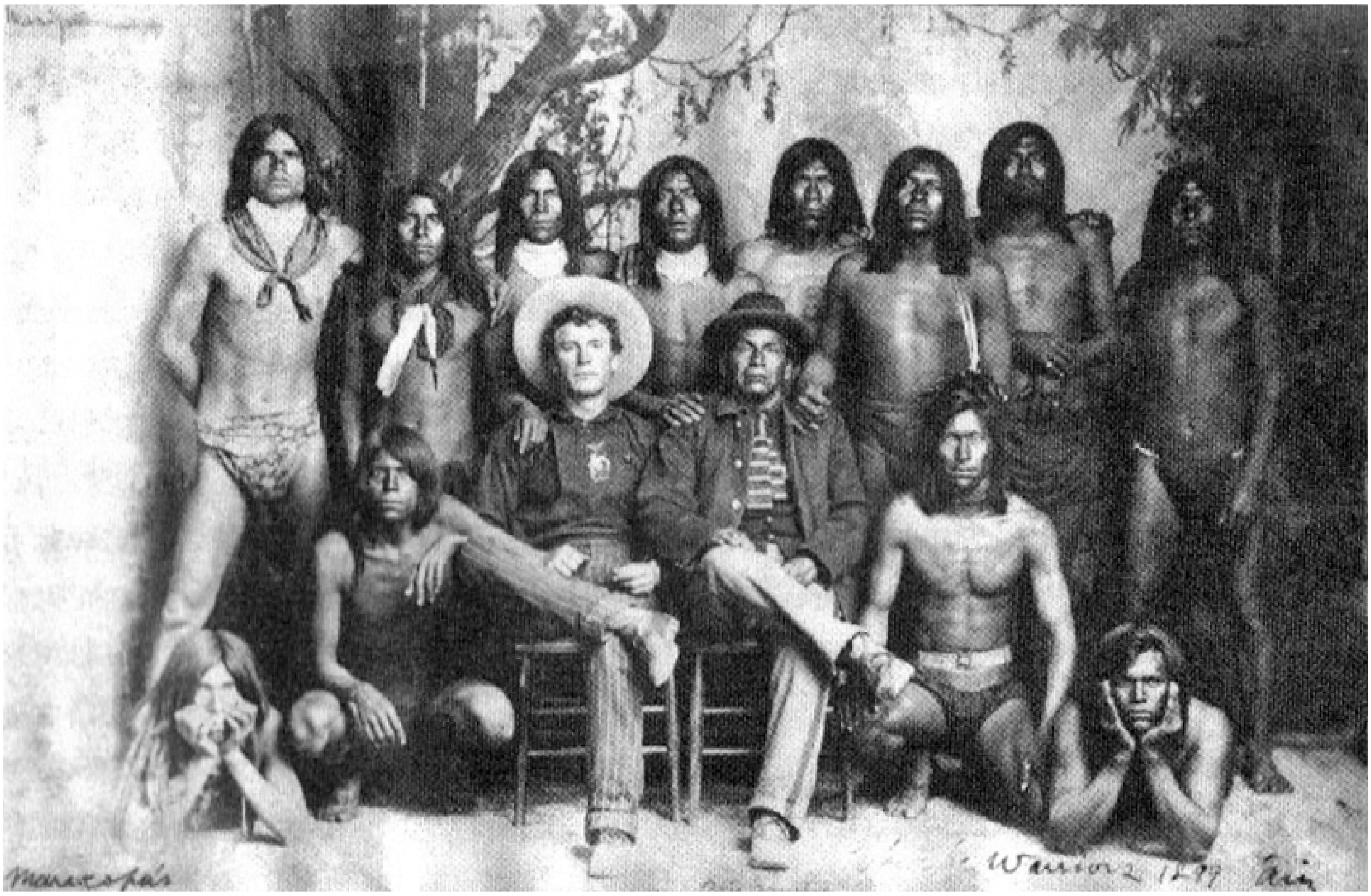
CLASSIFICATION METRICS

Classification is probably the most common problem in machine learning and many problems can be reduced to a classification problem.

Here, we will review the following metrics:

```
1) Accuracy.
2) Logarithmic loss.
3) Area under ROC curve.
4) Confusion matrix.
5) Classification Report.
```

using our already familiar Pima Indians diabetes dataset.



In this exercise we will use one of the traditional Machine Learning dataset, the Pima Indians diabetes dataset.

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content The datasets consists of several medical predictor variables and one target variable, **Outcome**. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

- Pregnancies
- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI
- DiabetesPedigreeFunction (scores de likelihood of diabetes based on family history)
- Age
- Outcome

In [2]:

```
# Load the Pima indians dataset and separate input and output components

from numpy import set_printoptions
set_printoptions(precision=3)

filename="pima-indians-diabetes.data.csv"
names=["pregnancies", "glucose", "pressure", "skin", "insulin", "bmi", "pedi", "age", "outc
p_indians=pd.read_csv(filename, names=names)
p_indians.head()

# First we separate into input and output components
array=p_indians.values
X=array[:,0:8]
Y=array[:,8]
np.set_printoptions(suppress=True)
X
pd.DataFrame(X).head()
```

Out[2]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[2]:

```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

Out[2]:

	0	1	2	3	4	5	6	7
0	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0
1	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0
2	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0

Classification Accuracy

Accuracy is the ratio of correct predictions over all predictions. It is by far the most used metrics.

However, it is only suitable when classes are balanced and errors in each class are equally important. For example whe it is equally important to missclassify a healthy person as having cancer (Type I) or a sick one as healty (Type II). As you can guess, in many cases this is not the case and accuracy is many times misused.

In [3]:

```
# Accuracy
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# KFold
splits=10
kfold=KFold(n_splits=splits, random_state=7, shuffle = True)
scoring="accuracy"

# Logistic regression
model = LogisticRegression(solver='liblinear')

# Obtain the performance measure - accuracy
results = cross_val_score(model, X, Y, scoring=scoring, cv=kfold)

print(f'Logistic regression, k-fold {splits:d} - Accuracy {results.mean()*100:.3f}% ({resul
```

```
Logistic regression, k-fold 10 - Accuracy 77.086% (5.091%)
```

In [ ]:

In [ ]:

Logarithmic Loss

Logarithmic loss (or logloss) is a performance metric for evaluating predictions of probabilities of membership to a class as a scalar between 0 and 1 that is seen as a measure of confidence.

Correct and incorrect predictions are rewarded or punished according to the confidence on the prediction. Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value.The penalty is logarithmic, offering a small penalty for small differences (0.1 or 0.2) and a large one for a large difference (0.9 or 1.0).

cross_val_score inverts (expressing the inversion with a - sign) the measure, therefore 0 is better.

In [4]:

```
# Logarithmic Loss

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

p_indians.head()

# KFold
splits=10
kfold=KFold(n_splits=splits, random_state=7, shuffle = True)
scoring="neg_log_loss"

#Logistic regression
model = LogisticRegression(solver='liblinear')

# Obtain the performance measure - accuracy
results = cross_val_score(model, X, Y, scoring=scoring, cv=kfold)

print(f'Logistic regression, k-fold {splits:d} - Logloss {results.mean():5.3f} ({results.st
```

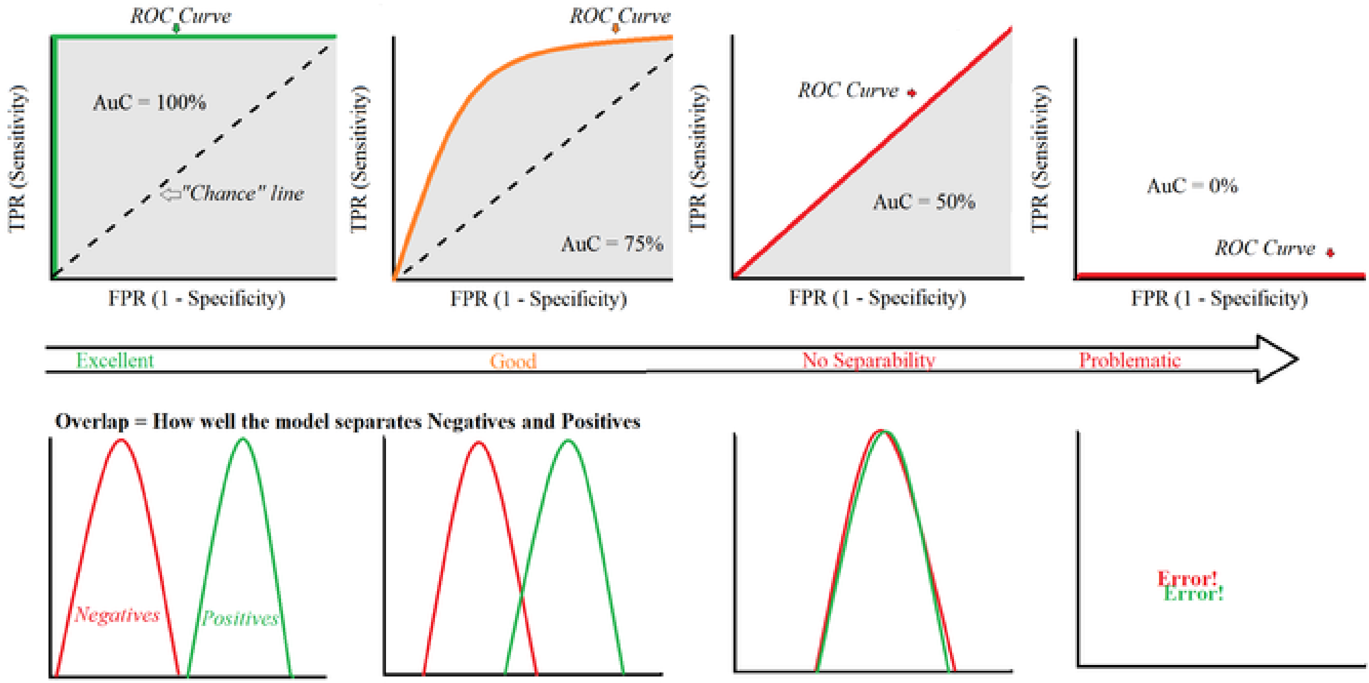
Out[4]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Logistic regression, k-fold 10 - Logloss -0.494 (0.042)
```

In [ ]:

In [ ]:



Area under ROC curve

Area under the ROC curve (AUC for short) is a metric used in binary classifications.

It represents the ability of the model to discriminate between positive and negative classes. An AUC of 0.5 represents a model that is as good as random, while an area of 1 represents a perfect model.

The area under the curve can be broken down into two metrics: sentitivity (recall) and specificity. In general any binary classification problem is a tradeoff between these two measures.

```
    - Sensitivity (True positive rate or recall). Represents the number of instances of the positive class that were correctly classified.
    - Specificity (True negative rate). Number of instances of the negative class classified correctly.
```

In [5]:

```
# Area under ROC curve

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

p_indians.head()

# KFold
splits=10
kfold=KFold(n_splits=splits, random_state=7, shuffle = True)
scoring="roc_auc"

#Logistic regression
model = LogisticRegression(solver='liblinear')

# Obtain the performance measure - accuracy
results = cross_val_score(model, X, Y, scoring=scoring, cv=kfold)

print(f'Logistic regression, k-fold {splits:d} - AUC {results.mean():5.3f} ({results.std():
```

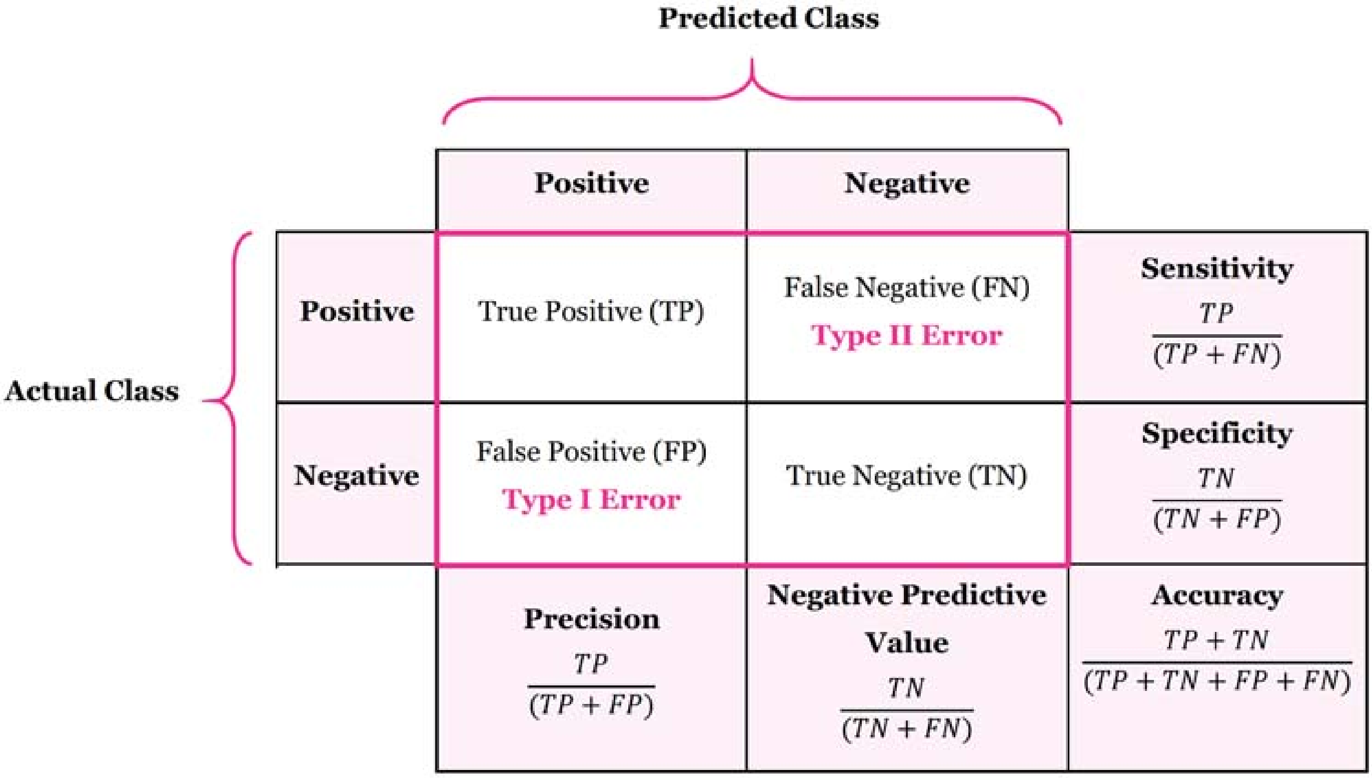
Out[5]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Logistic regression, k-fold 10 - AUC 0.825 (0.050)
```

In [ ]:

In [ ]:



Confusion Matrix

The confusion matrix is a presentation of the accuracy of the model in its four classes: True Positives, False Positives, False Negatives and True Negatives.

From it we can easily derive Precision, Recall, Specificity and Accuracy.

In [6]:

```
# Confusion Matrix

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

p_indians.head()

test_size=0.3
seed=7

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state

model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)

Y_predicted = model.predict(X_test)

c_matrix=confusion_matrix(Y_test, Y_predicted)

print("Confusion Matrix")
print(c_matrix)

print()
print(f'Accuracy {model.score(X_test, Y_test)*100:.5f}')
print(f'Accuracy check with conf. matrix {(c_matrix[0,0]+c_matrix[1,1])/c_matrix.sum()*100:
```

Out[6]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[6]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbos
e=0,
                   warm_start=False)
```

```
Confusion Matrix
[[130  17]
 [ 38  46]]

Accuracy 76.19048
Accuracy check with conf. matrix 76.19048
```

In [ ]:

In [ ]:

Classification Report

The scikit-learn library provides a report that is convenient and useful in terms of summarizing many of the measures that are commonly used together with accuracy.

For each class it presents:

> 1) Precision. $Precision = \frac{TruePositives}{TruePositives+FalsePosities}$
>
> 2) Recall (also known as Sensitivity). $Recall = \frac{TruePositives}{TruePositivies+FalseNegatives}$
>
> 3) F1 score. It's the harmonic mean of precision and recall. $F1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$, provides a good balance between precision and recall when classes are unevenly distributed. Best is 1, worst is 0.
>
> 4) Support. It's the number of elements of each class in Y_test.

The reported averages include:

```
    1) Macro average. Averaging the unweighted mean per label.
    2) Weighted average. Averaging the support-weighted mean per label.
    3) Sample average. Only for multilabel classification.
    4) Micro average. Accuracy for a binary classification. Averaging the total tr
ue positives, false negatives and false positives.
```

In [7]:

```
# Classification Report

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report

p_indians.head()

test_size=0.3
seed=7

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state

model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)

Y_predicted = model.predict(X_test)

report = classification_report(Y_test, Y_predicted, digits=5)


print(f'Accuracy {model.score(X_test, Y_test)*100:.5f}')
print()
print(report)
```

Out[7]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[7]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbos
e=0,
                   warm_start=False)
```

```
Accuracy 76.19048

              precision    recall  f1-score   support

         0.0    0.77381   0.88435   0.82540       147
         1.0    0.73016   0.54762   0.62585        84
```

```
    accuracy                          0.76190       231
   macro avg    0.75198    0.71599    0.72562       231
weighted avg    0.75794    0.76190    0.75283       231
```

In [ ]:

In [ ]:

Mission 1

We will use our predictions for top-10 and top-50 the Shanghai and Times Dataset

a) For the Shanghai dataset evaluate accuracy, logloss, AUC, confusion matrix and classification report. Briefly discuss the diferences.

b) Same for the Times ranking.

In [8]:

```
# Function to give evaluation metrics

def evaluation_metrics(x, y, kfolds):
    # Logistic regression
    model = LogisticRegression(solver='liblinear')

    # Obtain the performance measure - accuracy
    results = cross_val_score(model, x, y, scoring="accuracy", cv=kfolds)
    print(f'Logistic regression, k-fold {splits:5} - Accuracy {results.mean()*100:.3f}% ({r

    # Obtain the performance measure - logarithmic loss
    results = cross_val_score(model, x, y, scoring="neg_log_loss", cv=kfolds)
    print(f'Logistic regression, k-fold {splits:5} - Logloss {results.mean():5.3f} ({result

    # Obtain the performance measure - area under the curve
    results = cross_val_score(model, x, y, scoring="roc_auc", cv=kfolds)
    print(f'Logistic regression, k-fold {splits:5} - AUC {results.mean():5.3f} ({results.st

    # Confusion Matrix
    test_size = 0.5
    seed = 7

    X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=test_size, random_s

    model = LogisticRegression(solver = 'liblinear')
    model.fit(X_train, Y_train)

    Y_predicted = model.predict(X_test)

    c_matrix = confusion_matrix(Y_test, Y_predicted)

    print("Confusion Matrix")
    print(c_matrix)
    print()
    print(f'Accuracy {model.score(X_test, Y_test)*100:.5f}')
    print(f'Accuracy check with conf. matrix {(c_matrix[0,0]+c_matrix[1,1])/c_matrix.sum()*

    report = classification_report(Y_test, Y_predicted, digits=5)

    print(f'Accuracy {model.score(X_test, Y_test)*100:.5f}')
    print()
    print(report)
```

In [9]:

```
import data_cleaning as dc
```

In [10]:

```
# a) Predictions for top-10 Shangai Dataset

#from sklearn.utils import shuffle

shan_10 = dc.shangai_clean(10).values

shan_x10 = shan_10[:,0:6]
shan_y10 = shan_10[:,6]

# KFold
splits=3
kfold = KFold(n_splits=splits, random_state=7, shuffle = True)

evaluation_metrics(shan_x10, shan_y10, kfold)
```

```
Logistic regression, k-fold     3 - Accuracy 98.394% (0.751%)
Logistic regression, k-fold     3 - Logloss -0.136 (0.088)
Logistic regression, k-fold     3 - AUC 0.917 (0.103)
Confusion Matrix
[[246   1]
 [  0   2]]

Accuracy 99.59839
Accuracy check with conf. matrix 99.59839
Accuracy 99.59839

              precision    recall  f1-score   support

         0.0    1.00000   0.99595   0.99797       247
         1.0    0.66667   1.00000   0.80000         2

    accuracy                        0.99598       249
   macro avg    0.83333   0.99798   0.89899       249
weighted avg    0.99732   0.99598   0.99638       249
```

In [11]:

```
# a) Predictions for top-50 Shangai Dataset

#from sklearn.utils import shuffle

shan_50 = dc.shangai_clean(50).values

shan_x50 = shan_50[:,0:6]
shan_y50 = shan_50[:,6]

# KFold
splits=3
kfold=KFold(n_splits=splits, random_state=7, shuffle = True)

evaluation_metrics(shan_x50, shan_y50, kfold)
```

```
Logistic regression, k-fold     3 - Accuracy 96.988% (0.492%)
Logistic regression, k-fold     3 - Logloss -0.100 (0.021)
Logistic regression, k-fold     3 - AUC 0.981 (0.013)
Confusion Matrix
[[221   9]
 [  1  18]]

Accuracy 95.98394
Accuracy check with conf. matrix 95.98394
Accuracy 95.98394

              precision    recall  f1-score   support

         0.0    0.99550   0.96087   0.97788       230
         1.0    0.66667   0.94737   0.78261        19

    accuracy                        0.95984       249
   macro avg    0.83108   0.95412   0.88024       249
weighted avg    0.97040   0.95984   0.96298       249
```

In [12]:

```
# b) Predictions for top-10 Times Dataset

#from sklearn.utils import shuffle

times_10 = dc.times_clean(10).values

times_x10 = times_10[:,0:9]
times_y10 = times_10[:,9]

# KFold
splits=3
kfold=KFold(n_splits=splits, random_state=7, shuffle = True)

evaluation_metrics(times_x10, times_y10, kfold)
```

```
Logistic regression, k-fold     3 - Accuracy 99.145% (0.698%)
Logistic regression, k-fold     3 - Logloss -0.061 (0.040)
Logistic regression, k-fold     3 - AUC 0.994 (0.006)
Confusion Matrix
[[342   3]
 [  1   5]]

Accuracy 98.86040
Accuracy check with conf. matrix 98.86040
Accuracy 98.86040

              precision    recall  f1-score   support

         0.0    0.99708   0.99130   0.99419       345
         1.0    0.62500   0.83333   0.71429         6

    accuracy                        0.98860       351
   macro avg    0.81104   0.91232   0.85424       351
weighted avg    0.99072   0.98860   0.98940       351
```

In [13]:

```
# b) Predictions for top-50 Times Dataset

#from sklearn.utils import shuffle

times_50 = dc.times_clean(50).values

times_x50 = times_50[:,0:9]
times_y50 = times_50[:,9]

# KFold
splits=3
kfold=KFold(n_splits=splits, random_state=7, shuffle = True)

evaluation_metrics(times_x50, times_y50, kfold)
```

```
Logistic regression, k-fold     3 - Accuracy 96.866% (1.007%)
Logistic regression, k-fold     3 - Logloss -0.074 (0.023)
Logistic regression, k-fold     3 - AUC 0.988 (0.008)
Confusion Matrix
[[315  12]
 [  4  20]]

Accuracy 95.44160
Accuracy check with conf. matrix 95.44160
Accuracy 95.44160

              precision    recall  f1-score   support

         0.0    0.98746   0.96330   0.97523       327
         1.0    0.62500   0.83333   0.71429        24

    accuracy                        0.95442       351
   macro avg    0.80623   0.89832   0.84476       351
weighted avg    0.96268   0.95442   0.95739       351
```

REGRESSION METRICS

We will review the three most common regression metrics,

> 1) Mean Absolute Error (MAE).
> 2) Mean Square Error (MSE).
> 3) R^2

```
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

In [14]:

```
# Load the Boston Housing dataset and separate input and output components

from numpy import set_printoptions
set_printoptions(precision=3)

filename="HousingData.csv"
b_housing=pd.read_csv(filename)
b_housing.head()

b_housing.fillna(0,inplace=True) # we have NaN

# First we separate into input and output components

array=b_housing.values

X=array[:,0:13]
Y=array[:,13]

np.set_printoptions(suppress=True)
X
pd.DataFrame(X).head()
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST.
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	N.

Out[14]:

```
array([[  0.006,  18.   ,   2.31 , ...,  15.3  , 396.9  ,   4.98 ],
       [  0.027,   0.   ,   7.07 , ...,  17.8  , 396.9  ,   9.14 ],
       [  0.027,   0.   ,   7.07 , ...,  17.8  , 392.83 ,   4.03 ],
       ...,
       [  0.061,   0.   ,  11.93 , ...,  21.   , 396.9  ,   5.64 ],
       [  0.11 ,   0.   ,  11.93 , ...,  21.   , 393.45 ,   6.48 ],
       [  0.047,   0.   ,  11.93 , ...,  21.   , 396.9  ,   7.88 ]])
```

Out[14]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	0.00

Mean Absolute Error

The MAE (Mean Absolute Error) is the sum of the absolute differences between the actual values and the predictions.

It provides an idea of the magnitude of the error but not of its direction. A 0 indicates a perfect prediction and like logloss this metric is inverted by the cross_val_score() function.

In [15]:

```
# Mean Absolute Error

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

# KFold
kfold = KFold(n_splits=10, random_state=7, shuffle = True)

#model
model = LinearRegression()

scoring = "neg_mean_absolute_error"
res = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)

print(f'Boston Housing - Linear Regression, MAE: {res.mean():.3f} ({res.std():.3f})')
```

```
Boston Housing - Linear Regression, MAE: -3.410 (0.707)
```

In [ ]:

```

```

In [ ]:

```

```

Mean Squared Error

The idea of the MSE is the same of the MAE but we square the value in order to obtain always a positive value. Again, it provides an idea of the magnitude but not of the direction.

Many times we use the RMSE (Root Mean Squared Error) in order to convert the units back to the original units of the output variable.

Again this metric is inverted so results are increasing (scores that should be minimized are presented as negative while the ones that should be maximized as positive).

In [16]:

```
# Mean Squared Error

import math

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

# KFold
kfold = KFold(n_splits=10, random_state=7, shuffle = True)

#model
model = LinearRegression()

scoring = "neg_mean_squared_error"
res = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)

print(f'Boston Housing - Linear Regression, MSE: {res.mean():.3f} ({res.std():.3f})')
print(f'Boston Housing - Linear Regression, MSE: {math.sqrt(abs(res.mean())):.3f} ({math.sq
```

```
Boston Housing - Linear Regression, MSE: -24.758 (12.213)
Boston Housing - Linear Regression, MSE: 4.976 (3.495)
```

In [ ]:

In [ ]:

R^2

The coefficient of determination R^2 provides an indication of the goodness of the predictions.

It's a value of 0 and 1 for non-fit and perfect fit respectively. A value closer to 0 and less than 0.5 indicates a poor fit.

In [17]:

```
# R2

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

# KFold
kfold = KFold(n_splits=10, random_state=7, shuffle = True)

#model
model = LinearRegression()

scoring = "r2"
res = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)

print(f'Boston Housing - Linear Regression, R2: {res.mean():.3f} ({res.std():.3f})')
```

```
Boston Housing - Linear Regression, R2: 0.704 (0.112)
```

In [ ]:

In [ ]:

In [ ]: