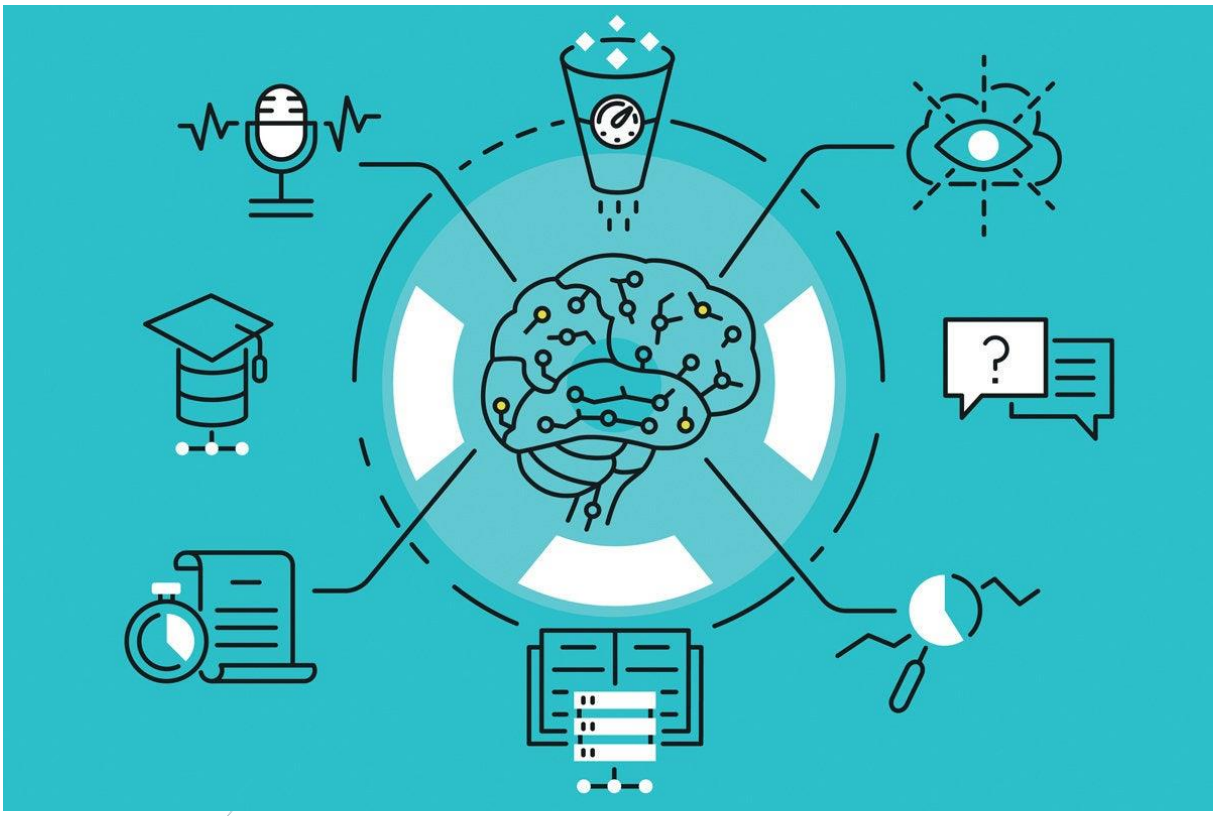
Université Abdelmalek Essaâdi

Faculté des Sciences et Techniques-Tanger

Département Génie Informatique

Machine Learning (Master SIBD)

Atelier 2 « «Classification»»



Réalisé Par :

Yossra safi chetouan

Encadre Par :

Pr . EL AACHAk LOTFI

Partie 1 (Data Visualisation et Feature Selection et Normalisation):

1. En utilisant pandas essayer d'explorer les données du Data set.

[52]:
```
data
```

[52]:

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0
...	...	...	...	...	...	...	...	...	...
762	10	101	76	48	180	32.9	0.171	63	0
763	2	122	70	27	0	36.8	0.340	27	0
764	5	121	72	23	112	26.2	0.245	30	0
765	1	126	60	0	0	30.1	0.349	47	1
766	1	93	70	31	0	30.4	0.315	23	0

767 rows × 9 columns

[54]:
```
# Afficher les premières lignes du DataFrame
print(data.head())
```

```
   6  148  72  35    0  33.6  0.627  50  1
0  1   85  66  29    0  26.6  0.351  31  0
1  8  183  64   0    0  23.3  0.672  32  1
2  1   89  66  23   94  28.1  0.167  21  0
3  0  137  40  35  168  43.1  2.288  33  1
4  5  116  74   0    0  25.6  0.201  30  0
```

[55]:
```
# Obtenir des informations générales sur les types de données et les valeurs manquantes
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 767 entries, 0 to 766
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   6       767 non-null    int64  
 1   148     767 non-null    int64  
 2   72      767 non-null    int64  
 3   35      767 non-null    int64  
 4   0       767 non-null    int64  
 5   33.6    767 non-null    float64
 6   0.627   767 non-null    float64
 7   50      767 non-null    int64  
 8   1       767 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

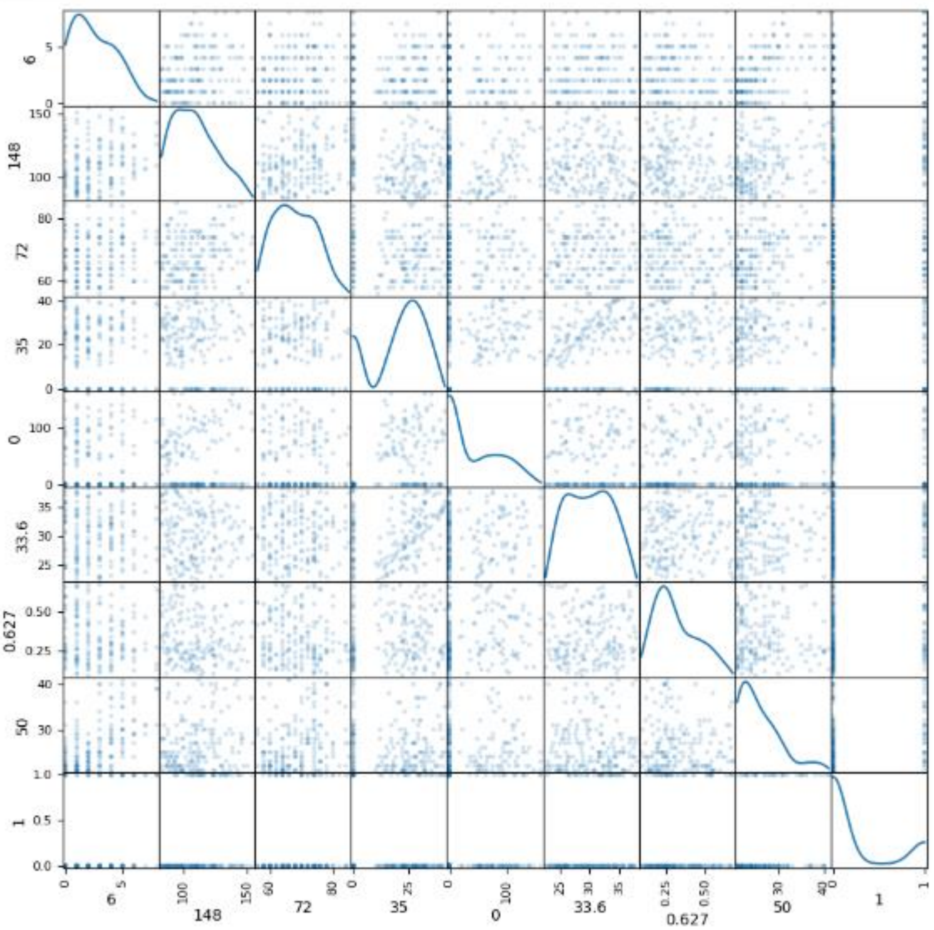
2. Afficher le résumer statistique du Data Sets avec une interprétation des résultats obtenues.

```
[56]: # Afficher des statistiques descriptives simples pour chaque colonne
print(data.describe())
```

```
                6         148          72          35           0        33.6  \
count  767.000000  767.000000  767.000000  767.000000  767.000000  767.000000
mean     3.842243  120.859192   69.101695   20.517601   79.903520   31.990482
std      3.370877   31.978468   19.368155   15.954059  115.283105    7.889091
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   27.300000
50%      3.000000  117.000000   72.000000   23.000000   32.000000   32.000000
75%      6.000000  140.000000   80.000000   32.000000  127.500000   36.600000
max     17.000000  199.000000  122.000000   99.000000  846.000000   67.100000

            0.627          50           1
count  767.000000  767.000000  767.000000
mean     0.471674   33.219035    0.348110
std      0.331497   11.752296    0.476682
min      0.078000   21.000000    0.000000
25%      0.243500   24.000000    0.000000
50%      0.371000   29.000000    0.000000
75%      0.625000   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

Afficher les nuages des points du data set selon les propriétés « Features » en utilisant matplotlib et pandas « scatter_matrix ».



3. Appliquer les 4 méthodes de Features selection « Univariate Selection, PCA, Recursive Feature Elimination et Feature Importance ».

[66]:
```
y=data[['1']]
x=data[['6','148','72','35','0','33.6','0.627','50']]
```

[67]:
```
#Recursive Feature Elimination (RFE)
X = data[['148', '33.6', '0.627', '50']]
y = data['1']
model = LogisticRegression(max_iter=1000)

# RFE pour sélectionner 1 caractéristique
selector = RFE(model, n_features_to_select=1)
selector.fit(X, y)

# Affichage des caractéristiques sélectionnées
selected_features = {X.columns[i]: rank for i, rank in enumerate(selector.ranking_)}
print("Importance des caractéristiques:", selected_features)
```

```
Importance des caractéristiques: {'148': 4, '33.6': 2, '0.627': 1, '50': 3}
```

[68]:
```python
#Feature Importance (FI)
from sklearn.ensemble import RandomForestRegressor

# Création du modèle Random Forest
forest = RandomForestRegressor()
forest.fit(x, y)

# Affichage de l'importance des caractéristiques
importances = forest.feature_importances_
print("Importance des caractéristiques:", {x.columns[i]: imp for i, imp in enumerate(importances)})
```

```
Importance des caractéristiques: {'6': 0.09190796116571993, '148': 0.1846933618483607, '72': 0.07759639965895541, '35': 0.08620045099301198, '0': 0.04348
511116831537, '33.6': 0.1947994388104901, '0.627': 0.15991901874416653, '50': 0.1613982576109799}
```

[69]:
```python
# Principal Component Analysis (PCA)
# Création de l'objet PCA et ajustement aux données
pca = PCA(n_components=2)  # Réduire à 2 dimensions
pca.fit(x)

# Transformation des caractéristiques
X_pca = pca.transform(x)
print("Nouvelles caractéristiques après PCA:\n", X_pca)
```

```
Nouvelles caractéristiques après PCA:
 [[-4.05860180e+01 -2.70560980e+01]
 [ 5.22599321e+01 -2.08660575e+01]
 [-4.36470359e+01  1.01556168e+01]
 [-4.37020470e+01  1.48010917e+00]
 [ 5.51844258e+01  3.79509430e+00]
 [ 9.70542853e+01 -1.02935889e+01]
 [ 1.11069744e+01 -2.04537328e+01]
 [-4.33561231e+01  3.72494237e+01]
 [-1.86405029e+01 -2.18797335e+01]
 [-4.35574660e+01  2.69807761e+01]
 [-4.06979140e+01 -1.22334446e+01]
 [ 5.90746864e+01  3.57088221e+01]
 [ 9.96693358e+01  2.59248557e+01]
 [-3.99189800e+01 -1.81982552e+01]
 [-4.11013694e+01  1.40123381e-01]
 [-4.04833081e+01 -1.21088740e+01]
 [-4.14706688e+01 -3.89437062e+00]
```

[70]:
```python
from sklearn.feature_selection import SelectKBest, chi2
bestfeatures = SelectKBest(score_func=chi2, k=4)
fit = bestfeatures.fit(x, y)
print("Scores de caractéristiques:", fit.scores_)
```

```
Scores de caractéristiques: [17.92773019 41.88773154  1.61452327  2.09760576  0.0633827   4.80018395
  0.16388581 16.84210526]
```

5. Normaliser les données des attributs qui nécessitent une normalisation.

[72]:
```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Choix du scaler
scaler = MinMaxScaler()

# Sélection des colonnes à normaliser
features_to_scale = ['148', '72', '35', '0', '33.6', '0.627', '50']
data[features_to_scale] = scaler.fit_transform(data[features_to_scale])

# Vérification des nouvelles statistiques
print(data[features_to_scale].describe())
```

```
              148         72         35          0       33.6      0.627  \
count  175.000000 175.000000 175.000000 175.000000 175.000000 175.000000
mean     0.376825   0.426995   0.474425   0.260464   0.492461   0.412043
std      0.250978   0.246552   0.325886   0.309813   0.272209   0.266392
min      0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
25%      0.166667   0.206897   0.000000   0.000000   0.248387   0.195392
50%      0.347222   0.413793   0.560976   0.000000   0.483871   0.336177
75%      0.569444   0.620690   0.731707   0.528125   0.716129   0.610068
max      1.000000   1.000000   1.000000   1.000000   1.000000   1.000000

               50
count  175.000000
mean     0.280000
std      0.266415
min      0.000000
25%      0.050000
50%      0.200000
75%      0.400000
max      1.000000
```

[73]:
```python
data
```

[73]:

	6	148	72	35	0	33.6	0.627	50	1
0	1	0.027778	0.344828	0.707317	0.0000	0.264516	0.441980	0.50	0
2	1	0.083333	0.344828	0.560976	0.5875	0.361290	0.127986	0.00	0
4	5	0.458333	0.620690	0.000000	0.0000	0.200000	0.186007	0.45	0
16	7	0.333333	0.620690	0.000000	0.0000	0.458065	0.276451	0.50	1
18	1	0.444444	0.482759	0.731707	0.6000	0.780645	0.745734	0.55	1
...	...	...	...	...	...	...	...	...	...
751	3	0.347222	0.206897	0.585366	0.0000	0.225806	0.223549	0.20	0
757	1	0.319444	0.689655	0.000000	0.0000	0.967742	0.179181	0.25	0
763	2	0.541667	0.482759	0.658537	0.0000	0.922581	0.423208	0.30	0
764	5	0.527778	0.551724	0.560976	0.7000	0.238710	0.261092	0.45	0
766	1	0.138889	0.482759	0.756098	0.0000	0.509677	0.380546	0.10	0

175 rows × 9 columns

Partie 2 (Classification choix de algorithme adéquat):

1. En utilisant l'API sklearn entraîner les modèles en utilisant ces algorithmes « KNN, Decision Tree, ANN,Naive Bayes, SVM selon les kernels suivants : Linear, polynomial et guassain».

[74]:
```
#Partie 2 (Classification choix de algorithme adéquat )
```

[75]:
```
X = data[['148', '33.6', '0.627', '50']]
y = data['1']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[76]:
```
# Entraînement des modèles
models = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'ANN': MLPClassifier(max_iter=1000),
    'Naive Bayes': GaussianNB(),
    'SVM Linear': SVC(kernel='linear', probability=True),
    'SVM Polynomial': SVC(kernel='poly', probability=True),
    'SVM Gaussian': SVC(kernel='rbf', probability=True)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} trained successfully.")
```

```
KNN trained successfully.
Decision Tree trained successfully.
ANN trained successfully.
Naive Bayes trained successfully.
SVM Linear trained successfully.
SVM Polynomial trained successfully.
SVM Gaussian trained successfully.
```

2. Sauvegarder les 5 modèles

[77]:
```
import joblib

for name, model in models.items():
    filename = f'{name.replace(" ", "_").lower()}_model.pkl'
    joblib.dump(model, filename)
```

3. Évaluer les modèles en utilisant ces métriques:
Classification Accuracy.
Logarithmic Loss.
Area Under ROC Curve.
Confusion Matrix.
Classification Report.

[78]:
```
#Évaluation des Modèles
```

[79]:
```
# Évaluation des modèles
for name, model in models.items():
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test) if hasattr(model, "predict_proba") else None
    print(f"{name} Classification Report")
    print(classification_report(y_test, y_pred))
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    if y_proba is not None:
        print(f"Log Loss: {log_loss(y_test, y_proba)}")
        print(f"ROC AUC: {roc_auc_score(y_test, y_proba[:, 1])}")
    print(f"Confusion Matrix: \n{confusion_matrix(y_test, y_pred)}\n")
```

```
KNN Classification Report
              precision    recall  f1-score   support

           0       0.93      0.90      0.92        30
           1       0.50      0.60      0.55         5

    accuracy                           0.86        35
   macro avg       0.72      0.75      0.73        35
weighted avg       0.87      0.86      0.86        35

Accuracy: 0.8571428571428571
Log Loss: 1.286540631399118
ROC AUC: 0.7933333333333333
Confusion Matrix:
[[27  3]
 [ 2  3]]
```

```
Decision Tree Classification Report
              precision    recall  f1-score   support

           0       0.92      0.77      0.84        30
           1       0.30      0.60      0.40         5

    accuracy                           0.74        35
   macro avg       0.61      0.68      0.62        35
weighted avg       0.83      0.74      0.77        35

Accuracy: 0.7428571428571429
Log Loss: 9.26836801434441
ROC AUC: 0.6833333333333332
Confusion Matrix:
[[23  7]
 [ 2  3]]
```

```
ANN Classification Report
              precision    recall  f1-score   support

           0       0.94      0.97      0.95        30
           1       0.75      0.60      0.67         5

    accuracy                           0.91        35
   macro avg       0.84      0.78      0.81        35
weighted avg       0.91      0.91      0.91        35

Accuracy: 0.9142857142857143
Log Loss: 0.297022035247771
ROC AUC: 0.8666666666666667
Confusion Matrix:
[[29  1]
 [ 2  3]]
```

```
Naive Bayes Classification Report
              precision    recall  f1-score   support

           0       0.94      0.97      0.95        30
           1       0.75      0.60      0.67         5

    accuracy                           0.91        35
   macro avg       0.84      0.78      0.81        35
weighted avg       0.91      0.91      0.91        35

Accuracy: 0.9142857142857143
Log Loss: 0.2904456243334175
ROC AUC: 0.8533333333333333
Confusion Matrix:
[[29  1]
 [ 2  3]]
```

```
SVM Linear Classification Report
              precision    recall  f1-score   support

           0       0.86      1.00      0.92        30
           1       0.00      0.00      0.00         5

    accuracy                           0.86        35
   macro avg       0.43      0.50      0.46        35
weighted avg       0.73      0.86      0.79        35

Accuracy: 0.8571428571428571
Log Loss: 0.4287053208544228
ROC AUC: 0.7666666666666666
Confusion Matrix:
[[30  0]
 [ 5  0]]
```

```
SVM Polynomial Classification Report
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        30
           1       0.60      0.60      0.60         5

    accuracy                           0.89        35
   macro avg       0.77      0.77      0.77        35
weighted avg       0.89      0.89      0.89        35

Accuracy: 0.8857142857142857
Log Loss: 0.3300339112323013
ROC AUC: 0.8266666666666667
Confusion Matrix:
[[28  2]
 [ 2  3]]
```

```
 -      --

SVM Gaussian Classification Report
              precision    recall  f1-score   support

           0       0.88      1.00      0.94        30
           1       1.00      0.20      0.33         5

    accuracy                           0.89        35
   macro avg       0.94      0.60      0.64        35
weighted avg       0.90      0.89      0.85        35

Accuracy: 0.8857142857142857
Log Loss: 0.37305740024044903
ROC AUC: 0.84
Confusion Matrix:
[[30  0]
 [ 4  1]]
```

4. Comparer la performance des 8 algorithmes en utilisant la technique Spot-checking.

```
[81]: # Spot-checking avec cross-validation
from sklearn.model_selection import train_test_split, cross_val_score
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5)
    print(f"{name} Cross-Validation Accuracy: {scores.mean()}")
```

```
KNN Cross-Validation Accuracy: 0.7857142857142856
Decision Tree Cross-Validation Accuracy: 0.7571428571428571
C:\Users\Acer\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  warnings.warn(
ANN Cross-Validation Accuracy: 0.7857142857142858
Naive Bayes Cross-Validation Accuracy: 0.7928571428571428
SVM Linear Cross-Validation Accuracy: 0.7857142857142857
SVM Polynomial Cross-Validation Accuracy: 0.7857142857142856
SVM Gaussian Cross-Validation Accuracy: 0.7714285714285714
```

5. Appliquer cette fois les trois techniques d'ensemble learning « bagging , stacking et boosting »

```
[83]: # Techniques d'ensemble Learning
from sklearn.ensemble import BaggingClassifier, StackingClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
```

```
[84]: # Bagging
bagging_model = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=10, random_state=42)
bagging_model.fit(X_train, y_train)
y_pred = bagging_model.predict(X_test)
print("Bagging Classification Report")
print(classification_report(y_test, y_pred))
```

```
Bagging Classification Report
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        30
           1       0.67      0.40      0.50         5

    accuracy                           0.89        35
   macro avg       0.79      0.68      0.72        35
weighted avg       0.87      0.89      0.87        35
```

[85]:

```python
# Stacking
estimators = [('knn', KNeighborsClassifier()), ('rf', RandomForestClassifier())]
stacking_model = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
stacking_model.fit(X_train, y_train)
y_pred = stacking_model.predict(X_test)
print("Stacking Classification Report")
print(classification_report(y_test, y_pred))
```

```
Stacking Classification Report
              precision    recall  f1-score   support

           0       0.88      1.00      0.94        30
           1       1.00      0.20      0.33         5

    accuracy                           0.89        35
   macro avg       0.94      0.60      0.64        35
weighted avg       0.90      0.89      0.85        35
```

[86]:

```python
# Boosting
boosting_model = GradientBoostingClassifier()
boosting_model.fit(X_train, y_train)
y_pred = boosting_model.predict(X_test)
print("Boosting Classification Report")
print(classification_report(y_test, y_pred))
```

```
Boosting Classification Report
              precision    recall  f1-score   support

           0       0.86      0.83      0.85        30
           1       0.17      0.20      0.18         5

    accuracy                           0.74        35
   macro avg       0.51      0.52      0.51        35
weighted avg       0.76      0.74      0.75        35
```