DATA MINING VALIDATION

- **Our Data Mining objective:**

⇨ the objective is to perform predictive analysis to predict the RATING of a Scrum master team based on certain team attributes, such as the number of projects and the team name.

CRISP-DM stands for Cross-Industry Standard Process for Data Mining. It is a widely-used methodology for guiding data mining and analytics projects. CRISP-DM provides a structured approach to the entire data mining process, from understanding the business problem to deploying solutions.

To summarize our work within the different phases of the CRISP-DM (Cross-Industry Standard Process for Data Mining) process, we organized our tasks according to these phases :

1) **Business Understanding and Data Understanding:**

- Data Import: Importing the dataset to begin the analysis.
- Initial Data Inspection: Displaying the first few rows of data to understand its structure.
- **Exploratory Data Analysis (EDA):**

- Data Information: Checking general information about the dataset.
- Statistical Description: Providing summary statistics to understand the distribution of data.
- Pairplot Visualization: Creating pair plots to visualize relationships between variables.

- **Data Cleaning:**

- Handling Missing Values and Duplicates: Removing rows with missing values and duplicate entries.
- Handling Infinite Values: Replacing infinite values with NaN (Not a Number) for further processing.

- **Correlation Analysis:**

- Calculating correlation coefficients between numerical variables to understand linear relationships.

2) Data Preparation:

- Dimensionality Reduction with PCA: Reducing the dimensionality of the dataset using Principal Component Analysis (PCA) to visualize data in a lower-dimensional space.
- Clustering with Agglomerative Hierarchical Clustering (AHC): Identifying clusters of similar data points using hierarchical clustering.
- One-Hot Encoding for Categorical Variables: Converting categorical variables ("description_equipe" and "nom_equipe" columns) into numerical representations suitable for machine learning models.

3) Modeling:

- Data Splitting: Dividing the dataset into training and testing sets for model training and evaluation.
- Regression Modeling: Training regression models (Linear Regression, Ridge Regression, Lasso Regression) to predict the target variable.
- Model Evaluation: Assessing model performance using evaluation metrics such as Mean Squared Error (MSE) and R-squared (R^2). Visualization of Model Performance: Plotting actual vs. predicted values to visualize model performance.

4) Evaluation:

- Cluster Evaluation with Confusion Matrix: Evaluating the quality of clusters using a confusion matrix.

5) Deployment :

- Exporting Cleaned Data: Exporting the cleaned dataset for further analysis or deployment.
- Our approach in the Data Mining phase is quite systematic and well-structured. Here is a detailed analysis of your methodologies in this phase:

1) Data importation

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix

# Importation des données
data = pd.read_csv('C:\\Users\\USER\\Downloads\\equipes_cleaan.csv')
```

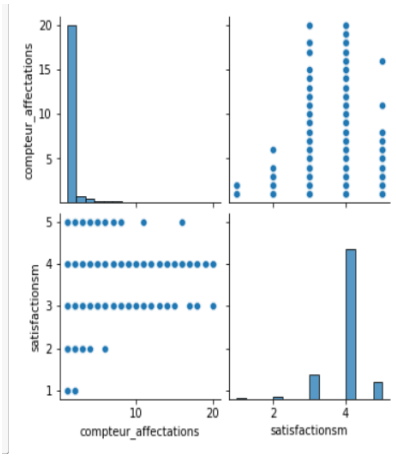
Data importation

2) Exploratory Data Analysis (EDA):

- We conducted an exploratory data analysis to understand the structure and characteristics of the data. We displayed the first few rows of the data to get an overview of the variables. We used info() to obtain information about the data types and missing values. We used describe() to get descriptive statistics of the numerical data. We used pairplot() to visualize the relationships between different variables.

```
# Analyse exploratoire des données (EDA)
print(data.info())
print(data.describe())
sns.pairplot(data)
plt.show()
```

⇨ **Results :**




```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21837 entries, 0 to 21836
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   nom_equipe            21837 non-null  object
 1   compteur_affectations  21837 non-null  int64
 2   description_equipe    21837 non-null  object
 3   satisfactionsm        21837 non-null  int64
dtypes: int64(2), object(2)
memory usage: 682.5+ KB
None
       compteur_affectations  satisfactionsm
count           21837.000000    21837.000000
mean                1.493154        3.926913
std                 1.579722        0.544475
min                 1.000000        1.000000
25%                 1.000000        4.000000
50%                 1.000000        4.000000
75%                 1.000000        4.000000
max                20.000000        5.000000
```

Exploratory Data Analysis

3) Data Cleaning:

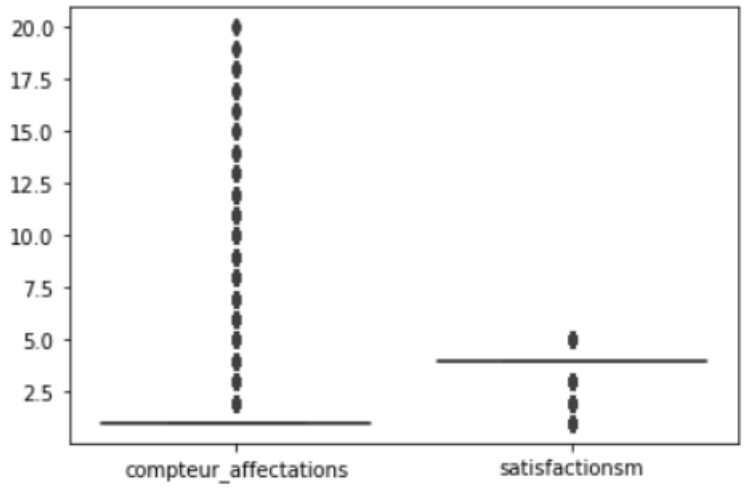
- We removed rows containing missing values or duplicates.
- We replaced infinite values with NaN to facilitate further processing.

```
# Nettoyage des données
data.dropna(inplace=True)
data.drop_duplicates(inplace=True)
```

```
# Remplacer les valeurs infinies par NaN
data.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
# Visualisation des données nettoyées
sns.boxplot(data=data)
plt.show()
```

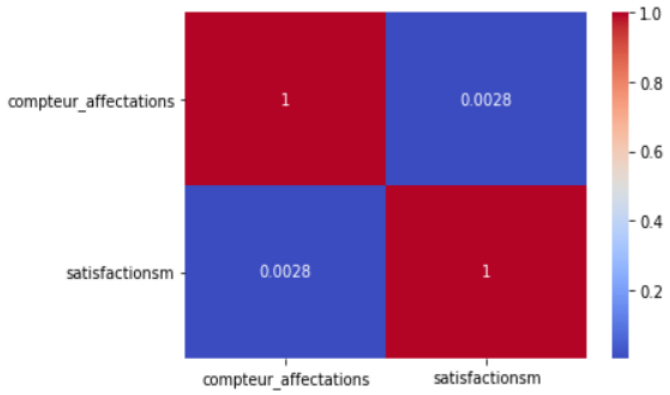
⇨ Result :



4) Correlation Analysis:

- We calculated the correlation matrix to understand the linear relationships between variables.

```
# Analyse de corrélation
numeric_data = data.select_dtypes(include=[np.number])  # Select only numeric columns
corr_matrix = numeric_data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```



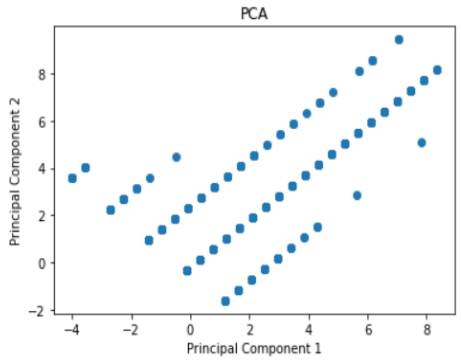
5) Dimensionality Reduction with PCA:

- We utilized Principal Component Analysis (PCA) to reduce the dimensionality of the data and visualize it in a reduced-dimensional space.

```
# Réduction de dimensionnalité avec PCA
numeric_data = data.select_dtypes(include=[np.number])  # Sélectionner uniquement les colonnes numériques
scaler = StandardScaler()
X_scaled = scaler.fit_transform(numeric_data)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA')
plt.show()
```



6) Clustering with AHC:

- We employed the agglomerative hierarchical clustering (AHC) algorithm to group similar data into clusters.

```
# Clustering avec Agglomerative Hierarchical Clustering (CAH)
n_clusters = 3
hc = AgglomerativeClustering(n_clusters=n_clusters)
cluster_labels = hc.fit_predict(X_scaled)

data['Cluster'] = cluster_labels
```

```
# Évaluation des clusters avec matrice de confusion
conf_matrix = confusion_matrix(data['satisfactionsm'], data['Cluster'])
print(conf_matrix)
```

```
[[    0     0     0     0     0     0]
 [    0     0    74     0     0     0]
 [    0     1   319     0     0     0]
 [    0    58  2615     0     0     0]
 [15508  1040     0     0     0     0]
 [ 1949     2     0     0     0     0]]
```

7) Regression Models:

- You trained several regression models (linear regression, ridge regression, lasso regression) to predict the target variable (satisfactionsm) using other variables as features.

Model Evaluation:

- You evaluated the models' performance by calculating the mean squared error (MSE) and coefficient of determination (R^2). Additionally, you visualized the models' performance by comparing actual values to predicted values using graphs.

```python
# Modèles de régression
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso()
}

# Entraînement et évaluation des modèles
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"Model: {name}")
    print("Mean Squared Error:", mse)
    print("R-squared (R2) Score:", r2)
    print()
    # Plot actual vs. predicted values for each model
plt.figure(figsize=(15, 5))

# Linear Regression
plt.subplot(1, 3, 1)
plt.scatter(y_test, models['Linear Regression'].predict(X_test), color='blue')
plt.title('Linear Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Ridge Regression
plt.subplot(1, 3, 2)
plt.scatter(y_test, models['Ridge Regression'].predict(X_test), color='green')
plt.title('Ridge Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

# Lasso Regression
plt.subplot(1, 3, 3)
plt.scatter(y_test, models['Lasso Regression'].predict(X_test), color='red')
plt.title('Lasso Regression')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')

plt.tight_layout()
plt.show()
```

⇨ Result :

```
Model: Linear Regression
Mean Squared Error: 3.849713136739491e+20
R-squared (R2) Score: -1.2280116844838453e+21

Model: Ridge Regression
Mean Squared Error: 0.08653194004883484
R-squared (R2) Score: 0.7239736320950307

Model: Lasso Regression
Mean Squared Error: 0.31377540896277956
R-squared (R2) Score: -0.000905404698118506
```