Rapport du Projet :

Réalisé par :

Coulibaly Bourama

Gilles Gneme

Assia Darhouane

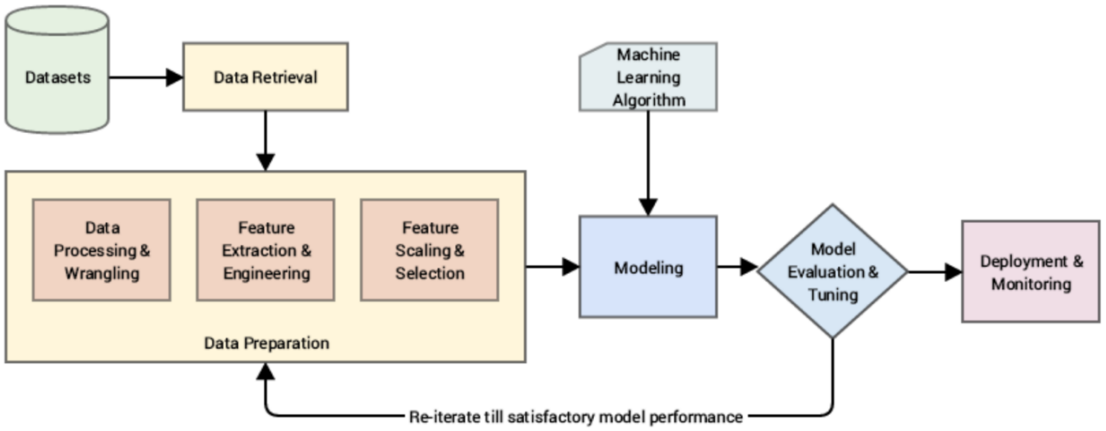
Oumaima Ghali

Tami Alain Aboukary Jude

Présentation du projet :

Ce projet a été concentré sur la classification du breast cancer. Le breast cancer est un type de cancer qui se forme dans les cellules du sein. Il peut survenir chez les hommes et les femmes, mais il est beaucoup plus fréquent chez les femmes. Le cancer du sein est le deuxième cancer le plus fréquent chez la femme après le cancer de la peau.

Pour bien mener à terme notre projet, nous avons suivi le pipeline suivant. Mais pour notre cas on s'est arrêté au niveau de l'évaluation du modèle.



1. *Data Retrieval :*

En apprentissage automatique, la récupération de données fait référence au processus d'acquisition de données à partir de diverses sources, telles que des bases de données, des API ou du web scraping, afin de les utiliser pour l'entraînement et le test des modèles d'apprentissage automatique.

Pour notre cas, on a récupéré le dataset sur Kaggle.

En ce qui concerne les étapes de Data Processing, Feature Extraction, Feature Scaling and selection, nous avons suivi les étapes suivantes.

- Nettoyage des données : Cette étape consiste à traiter les données manquantes ou incomplètes, qui peuvent être soit supprimées (lignes ou colonnes) soit remplies avec des valeurs estimées.
- Normalisation des données : Cette étape consiste à mettre à l'échelle les données dans une plage spécifique, comme entre 0 et 1 ou -1 et 1. C'est important pour garantir que toutes les fonctionnalités oSélection des caractéristiques : Cette étape consiste à ne sélectionner que les caractéristiques les plus pertinentes pour le modèle d'apprentissage automatique, ce qui peut aider à réduire le surajustement et améliorer la précision du modèle.
- Sélection des caractéristiques : Cette étape consiste à ne sélectionner que les caractéristiques les plus pertinentes pour le modèle d'apprentissage automatique, ce qui peut aider à réduire le surajustement et améliorer la précision du modèle.

```
data = pd.read_csv('C:/Users/HP/Desktop/Projet_academique/breast-cancer-dataset/breast-cancer.csv')
df = pd.DataFrame(data)
```

Basic Data Analysis

```
#Checking dataset informations
data.info()
```

```
#Checking dataset column and row numbers
data.shape
```

]: (569, 32)

```
# Binary classification
data['diagnosis'] = [1 if i == 'M' else 0 for i in data['diagnosis']]
```

```
data['diagnosis'].unique()
```

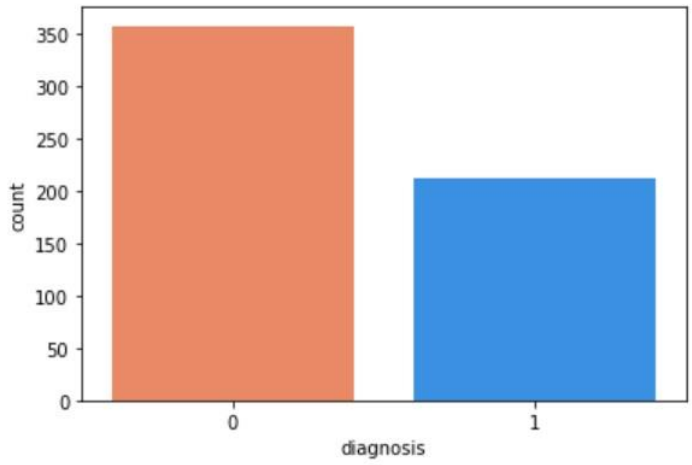
]: array([1, 0], dtype=int64)

```
data["diagnosis"].value_counts()
```

```
#checking statistical analysis of numerical values
data.describe().T
```

```
custom_palette = sn.color_palette(['#FF7F50', '#1E90FF'])

sn.countplot(x='diagnosis' , data=data, palette=custom_palette)
```

<AxesSubplot:xlabel='diagnosis', ylabel='count'>



```
data.isna().sum()
```

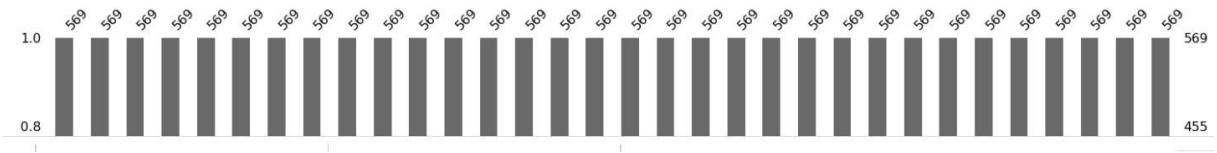
```
data.head().T
```

```
data.drop("id", axis = 1, inplace=True)
```

```
data.head().T
```

```
# Missing values
#Missingno is a Python library for visualizing missing data in datasets
#Utilisation de Missingno pour diagnostiquer la rareté des données
fig = plt.figure(figsize=(15,8))
msno.bar(data)
```

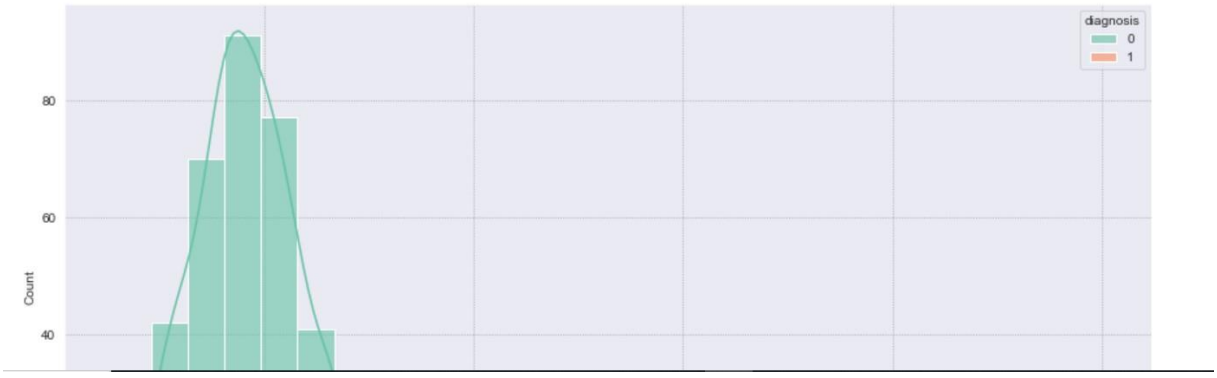
```
<AxesSubplot:>
```




```
for i in data.columns:
    fig = px.histogram(data,
                       x=i,
                       marginal='box',
                       text_auto=True,
                       color_discrete_sequence  = ['palevioletred'],
                       template='simple_white',
                       title=i.upper() + ' HISTOGRAM')

    fig.update_layout(xaxis_title=i,yaxis_title="Count", bargap=0.1)

    fig.show()
```

```
sn.set_style("darkgrid", {"grid.color": ".6", "grid.linestyle": ":"})

fig = plt.figure(figsize=(15,8))
sn.histplot(data=data,x='area_mean',
            hue='diagnosis',
            kde=True,
            palette='Set2',
            alpha=0.6)
```

```
<AxesSubplot:xlabel='area_mean', ylabel='Count'>
```



```
fig = plt.figure(figsize=(15,8))
sn.histplot(data=data,x='radius_mean',
            hue='diagnosis',
            kde=True,
            palette='Set2',
            alpha=0.6)
```

```
<AxesSubplot:xlabel='radius_mean', ylabel='Count'>
```


```
fig = px.histogram(data,
                   x="area_worst",
                   color_discrete_sequence=['palevioletred'],
                   marginal='box',
                   text_auto=True,
                   hover_data=["diagnosis"],
                   nbins=10,
                   template='simple_white',
                     labels={"area_worst": "Worst Area"},
                   title="Worst Area Distribution")
fig.update_layout(xaxis_title="worst area",yaxis_title="Count")
fig.update_layout(bargap=0.1)
fig.show()
```

Model et Training

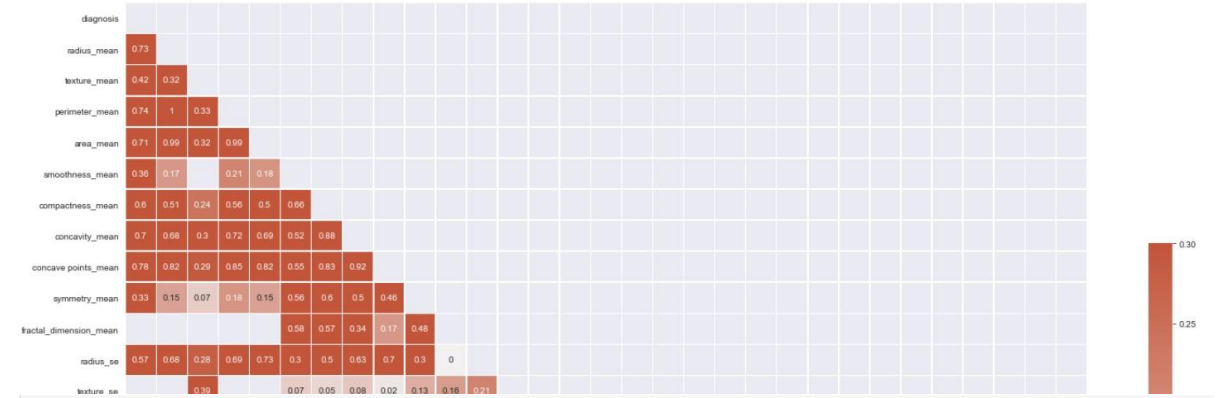
```
import pandas as pd
import numpy as np
import sklearn as sk
```

```
data = pd.read_csv("./cleaned_data.csv")
```

```
data.head()
```

[10]:

```python
#Correlation Heatmap showing only those values that have a positive correlation
corr = data.corr()
corr = np.around(corr[corr > 0.0],2)                #filters out any negative correlation
mask = np.triu(np.ones_like(corr, dtype=bool))      #Gets rid of the other triange in the heatmap
f, ax = plt.subplots(figsize=(25,20))
cmap = sn.diverging_palette(220, 20, as_cmap=True)
sn.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, annot=True,square=True, linewidths=.5, cbar_kws={'shrink': .5})
plt.show()
```




```python
fig = px.scatter(data,
                 x="compactness_mean",
                 size="compactness_mean",
                 color="compactness_mean",
                 labels={"compactness_mean": "Mean Compactness"},
                 hover_data=["diagnosis"],
                 template='seaborn',
                 title="Cancer Compactness Mean")
fig.update_layout(xaxis_title="Mean Compactness",yaxis_title=" ")
fig.update_layout({'plot_bgcolor': 'rgba(0, 0, 0, 0)','paper_bgcolor': 'rgba(0, 0, 0, 0)'},)
fig.show()
```

2. Model Evaluation and Tuning :

L'entraînement d'un modèle consiste à entraîner un modèle d'apprentissage automatique sur un ensemble de données étiquetées afin de faire des prédictions sur de nouvelles données non vues auparavant. Pour notre cas nous avons utilisé la régression logistique.

Voici les étapes qu'on a suivi au cours de notre projet :

- *Prétraitement des données : Avant d'entraîner un modèle, les données doivent être nettoyées, prétraitées et divisées en ensembles d'entraînement et de test.*
- *Choix d'un modèle : Choisissez un algorithme d'apprentissage automatique approprié pour la tâche et les données, puis instanciez le modèle. Ici, comme déjà mentionné c'est la régression logistique.*
- *Ajustement du modèle : Ajustez le modèle aux données d'entraînement en appelant la méthode fit(), qui ajuste les paramètres internes du modèle pour minimiser l'erreur entre les sorties prédites et les sorties réelles.*
- *Évaluation du modèle : Après l'entraînement du modèle, il est important d'évaluer ses performances sur un ensemble de test séparé. Utilisez la méthode predict() pour faire des prédictions sur l'ensemble de test, puis comparez ces prédictions aux étiquettes réelles pour calculer des métriques telles que la précision, le rappel et le score F1.*
- *Ajustement des hyperparamètres : Pour optimiser les performances du modèle, vous devrez peut-être ajuster les hyperparamètres (comme le taux d'apprentissage, le paramètre de régularisation ou le nombre de couches cachées). Cela peut être fait en testant différentes combinaisons d'hyperparamètres et en sélectionnant les meilleurs.*

```
def initialisation(X):
    W = np.random.randn(X.shape[1], 1)
    #b = np.random.randn(1)
    return W
theta = initialisation(X_train)
```

```
def model(X, W):
    Z = np.dot(X,W)
    A = 1 / (1 + np.exp(-Z))
    return A

#model(X_train, theta)
```

```
def cost_function(X, y, theta):
    epsilon = 1e-15
    m = len(y)
    h = model(X, theta)
    J = -1/m * np.sum(y * np.log(h + epsilon) + (1 - y) * np.log(1 - h + epsilon))
    return J
cost_function(X_train.values, y_train.values, theta)
```

```
9878.09004894445
```

```
def gradients(X, y, theta):
    A = model(X, theta)
    dW = 1 / len(y) * np.dot(X.T, A - y)
    #db = 1 / len(y) * np.sum(A - y)
    return dW

dw = gradients(X_train.values, y_train.values, theta)
dw.shape
```

```
(30, 455)
```

```
np.mean(dw, axis=1).shape
theta.shape
```

```
(30, 1)
```

```
def Update(dW, W, learning_rate):
    W = W - learning_rate * dW
    #W = W - learning_rate * np.mean(dW, axis=1).reshape(len(dW), 1)
    #b = b - learning_rate * db
    return W

new_w = Update(dw, theta, 0.01)
new_w.shape
```

```
(30, 455)
```

```
def logistic_regression(X, y, learning_rate = 0.01, n_iterations = 100):
    # initialisation de W
    W = initialisation(X)

    Loss = []
    for i in range(n_iterations):
        #A = model(X, W)
        for j in range(X.shape[0]):
            Xx = X[j, :].reshape(1, X.shape[1])
            yy = y[j].reshape(1, 1)
            Loss.append(cost_function(Xx, yy, W))
            dW = gradients(Xx, yy, W)
            W = Update(dW, W, learning_rate)
        #Loss.append(cost_function(X, y, W))
        #dW = gradients(X, y, W)
        #W = Update(dW, W, learning_rate)
    return (W, Loss)

output = logistic_regression(X_train.values, y_train.values)
```

```
def check(x):
    if x:
        return 1
    return 0

def predict(X, W):
    A = model(X, W)
    x = A >= 0.5
    xx = [check(i) for i in x]
    return xx

preds = predict(X_test.values, output[0])
```

```
C:\Users\gille\AppData\Local\Temp/ipykernel_2404/218105309.py:3: RuntimeWarning: overflow encountered in exp
  A = 1 / (1 + np.exp(-Z))
```

```
accuracy_score(y_test.values, preds)
```

```
: 0.9385964912280702
```

```
weights = pd.DataFrame(output[0])
```

```
weights.to_csv('./weights_without_optimisation.csv', index=False)
```