RAPPORT DU PROJET
“Prédiction du cancer du sein”

Préparé par :

- Boutaoui Nada
- Madadi Mounia
- Mitiche Sarah Marya
- Touati Yanis
- Zerrouk Ikram

15/05/2023

2CS - Machine Learning

Table de matières

I. Introduction

Dans le cadre de ce projet il nous a été demandé de développer un modèle de machine learning, ce qui nous permet d'acquérir de nouvelles compétences techniques comme apprendre et maîtriser des bibliothèques et des frameworks , d'innover et d'exploiter les données de manière efficace.

Et comme sujet, on a choisi de travailler sur des prédictions du cancer du sein, car un tel modèle peut avoir de très grand impact dans le domaine médical, par exemple il sera possible de détecter le cancer du sein à un stade précoce, ce qui augmente les chances de succès du traitement, il peut aussi contribuer à optimiser l'utilisation des ressources médicales, en identifiant les cas présentant un risque élevé de cancer du sein ainsi les médecins peuvent prioriser les patients pour des examens plus approfondis, un tel modèle peut même être utilisés pour la recherche et le développement de nouvelles méthodes de diagnostic et de traitement, mais évidemment le modèle qu'on va construire ne sera qu'un simple exemple des modèle de prédiction de cancer du sein car pour arriver à des résultats fiables et précis il est nécessaire d'acquérir des données de très haute qualité ainsi que des compétences technique très élevé.

Pour ce projet, nous avons travaillé avec un Dataset étiqueté qu'on a trouvé en ligne, et qui contenait des résultats de différents tests sur des personnes différentes ainsi que le résultat de leur diagnostic.

II. Exploration des données

Évidemment, la première étape dans la construction de notre modèle est l'exploration et le prétraitement des données car ces étapes nous permettent de prendre des décisions éclairées sur la modélisation, de découvrir des informations précieuses et de maximiser les performances du modèle.

1) Prétraitement des données :

Le prétraitement des données est important car il peut avoir un impact significatif sur les performances de notre modèle final.

En ce qui concerne notre Dataset, la plupart des données étaient déjà prêtes pour commencer l'entraînement, donc il n'y avait pas énormément de transformations à faire sur notre Dataset.

Voici les étapes par lesquelles on est passé pour la préparation de nos données :

- On a commencé par voir un petit aperçu sur notre Dataset :

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetr
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	(
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	(
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	(
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	(
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	(
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	(
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	(

- Ensuite, on a vérifié le nombre de valeurs nulles par colonne :

```
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Comme présenté ci dessus, mis à part une seule colonne dans laquelle toutes les valeurs étaient nulles, toutes les autres colonnes ne contenaient aucune valeur nulle. Il suffisait donc de supprimer la colonne **“unnamed”** :

```
df.drop(columns=['Unnamed: 32','id'],inplace=True)
```

- On devait ensuite vérifier s’il y avait des lignes dupliquées :

```
df.duplicated().sum()
```

```
0
```

Et comme il est affiché, il n’y avait aucune ligne dupliquée, ce qui est parfait.

- Finalement, on a vérifié qu' il n’y avait pas des valeurs hors des résultats attendus (c’est à dire **“B”** ou **“M”**) dans notre variable cible :

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

```
array(['M', 'B'], dtype=object)
```

- Une fois sûr, on applique un petit codage sur nos valeur pour faciliter le traitement de nos données :

```
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
```

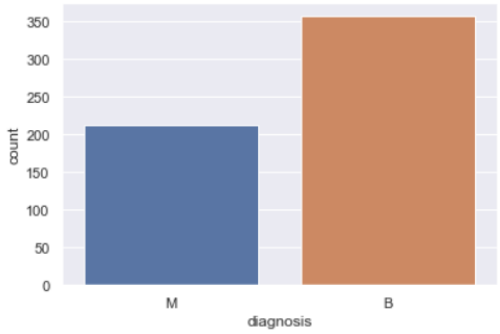
Globalement, c’est ça, on a fait le tour sur tous les prétraitements qu’il faut pour assurer la production des modèles performants et efficaces qui convergent plus rapidement et fournissent de meilleurs résultats. Cela aide également à réduire la complexité et à faciliter l'interprétation des résultats.

2) Visualisation des données :

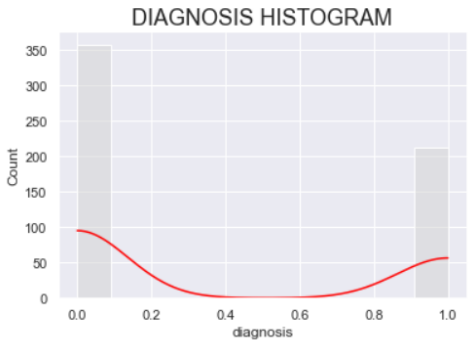
Afin de bien comprendre notre Dataset, ainsi que les attributs qui le composent, la relation entre eux, leur effet sur notre variable cible et la distribution des résultats des patients, nous avons affiché quelques visualisations dont les suivantes :

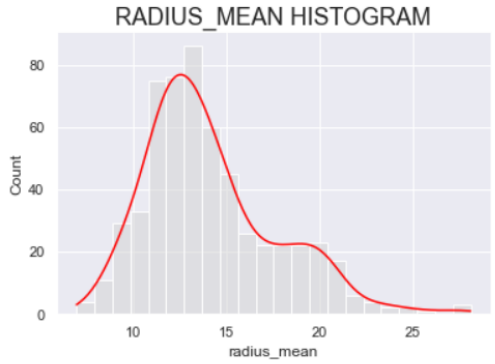
- On a d'abord vérifié quelle était la distribution des résultats du diagnostic qu'on a dans notre Dataset :

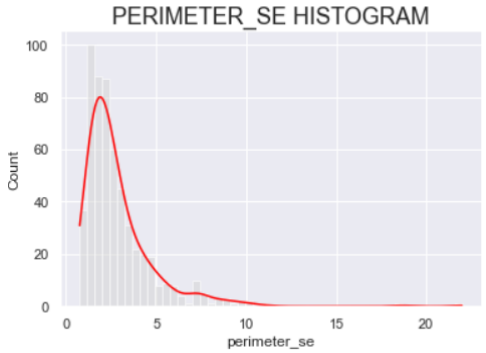
```
sns.countplot(x=df.diagnosis);
```



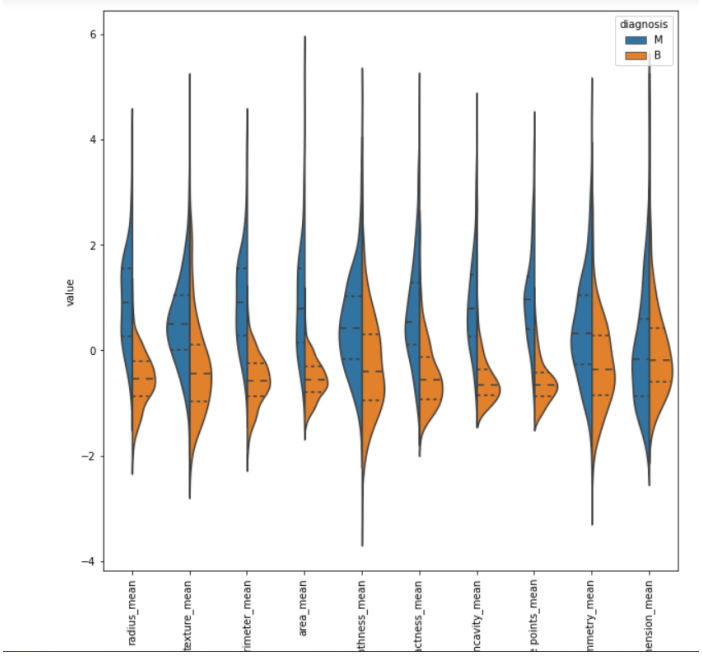
- On a aussi analysé la distribution des valeurs de toutes colonnes qu'on a :







- Ensuite une analyse de la distribution des résultats du diagnostic selon les 10 premières colonnes a été effectuée :

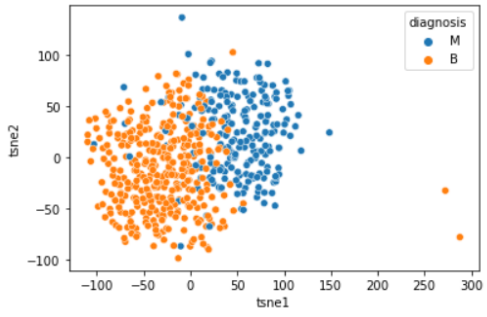


A côté de ces visualisations, on a aussi utilisé deux techniques qui sont couramment très utilisées en Machine Learning surtout pour voir s'il nous faut une réduction de la dimensionnalité :

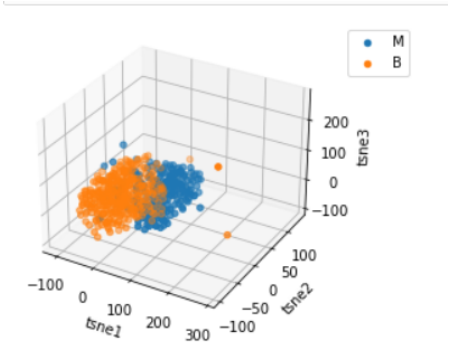
1. T-SNE (t-Distributed Stochastic Neighbor Embedding) :

Une technique de réduction de dimensionnalité non linéaire utilisée pour visualiser les données de haute dimension en deux ou trois dimensions.La méthode t-SNE utilise une approche probabiliste pour préserver les relations de similarité entre les points dans les données d'origine. Avec cette méthode on a analysé les visualisation suivante :

Une visualisation 2D des données :



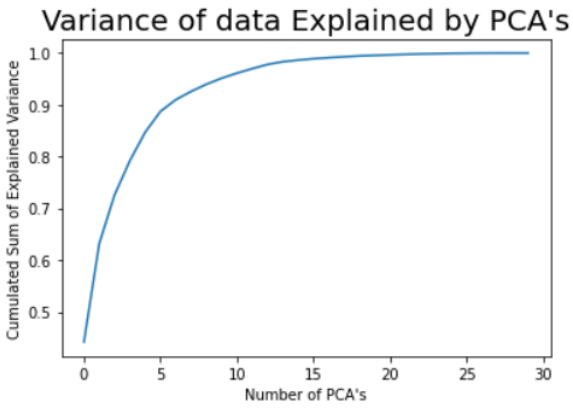
Une visualisation 3D des données :



2. PCA (Principal Component Analysis) :

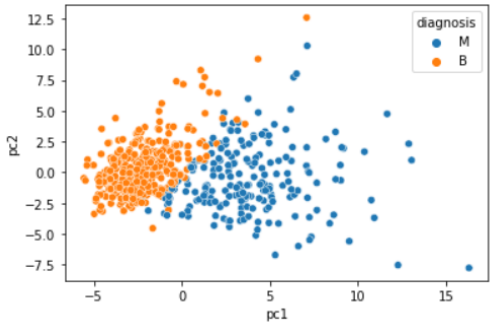
Une méthode linéaire qui cherche à transformer les variables d'origine en un nouvel ensemble de variables non corrélées appelées composantes principales. Les composantes principales sont ordonnées en fonction de la quantité de variance qu'elles capturent dans les données. par cette méthode on a obtenu les visualisations suivantes :

On peut clairement observé sur la figure suivante comment 90% de la variances des données est expliqué grâce à seulement 7 colonnes :

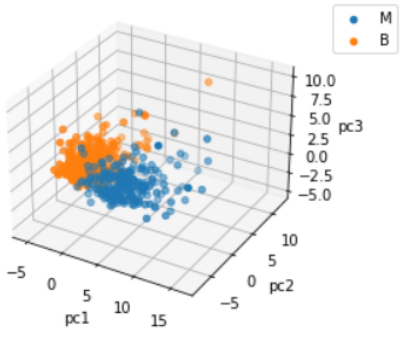


7 number of principle components explains 90.0 % of the data's variance

Une visualisation 2D des données :



Une visualisation 3D des données :



III. Modélisation

1. Logistic Regression :

A)Feature Selection :

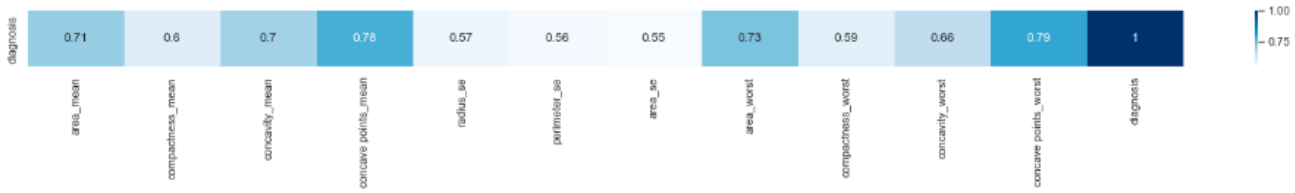
La régression linéaire avec sélection de caractéristiques est une méthode qui vise à sélectionner un sous-ensemble de caractéristiques pertinentes pour améliorer la performance d'un modèle de régression linéaire. Cela peut aider à réduire la complexité du modèle, à améliorer l'interprétabilité et à éviter le surajustement.

Pour notre cas, on a effectué une sélection de caractéristiques à l'aide de la méthode **ANOVA F-value** (analyse de variance), avec calcul des scores et des p-values de chaque fonctionnalité. Les 15 meilleures fonctionnalités sélectionnées en fonction des scores sont les suivantes :

```
Index(['radius_mean', 'perimeter_mean', 'area_mean', 'compactness_mean',
       'concavity_mean', 'concave points_mean', 'radius_se', 'perimeter_se',
       'area_se', 'radius_worst', 'perimeter_worst', 'area_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst'],
      dtype='object')
```

Après choix des meilleures features, on devait créer notre nouveau Dataframe qui sera constitué des features sélectionnées + la variable cible “diagnosis” pour qu'on puisse identifier les relations de corrélation entre les fonctionnalités et de détecter d'éventuelles dépendances linéaires entre elles. Cette dernière étude nous a fait laisser tomber quelques features, à savoir : **'radius_mean'**, **'perimeter_mean**', **'radius_worst'** et **'perimeter_worst**'.

Finalement, et en utilisant une heatmap, on a pu visualiser cette fois ci la corrélation entre la variable cible **"diagnosis"** et les autres variables du Dataframe comme le montre cette figure :



Après avoir fait toutes les modifications qu'il faut, on passe aux étapes suivantes :

1- Data Splitting : ou nous divisons les données en un ensemble d'entraînement (X_train, y_train) et un ensemble de test (X_test, y_test) à l'aide de la fonction **train_test_split.**

2- Feature Scaling : ou nous effectuons une mise à l'échelle des caractéristiques à l'aide de la classe **StandardScaler**, **fit_transform** sur l'ensemble d'entraînement, et **transform** sur l'ensemble de test pour garantir que les deux ensembles de données sont mises à l'échelle de la même manière.

3- Ajustement des hyperparamètres : nous utilisons **GridSearchCV,** ça donne ces résultats :

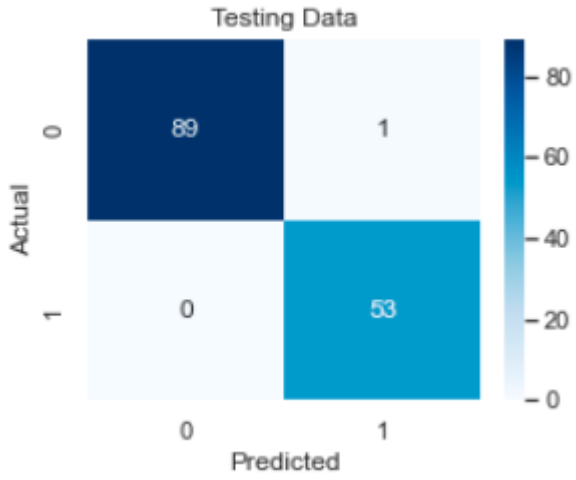
```
{'C': 100, 'penalty': 'l1'}
```

Une fois que toutes ces étapes sont achevées, on peut utiliser ces ensembles de données pour entraîner et évaluer notre modèle de régression linéaire (**Feature Selection**).

Pour l'évaluation, on peut utiliser plusieurs méthodes dont on a présentées :

1)Training Report et sa matrice de confusion :

2)Testing Report et sa matrice de confusion :



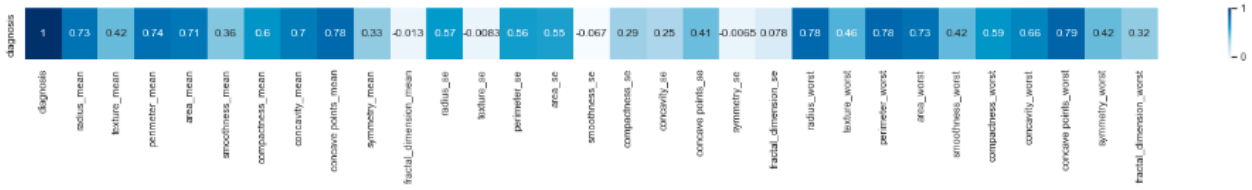
3)Precision-Recall Curve et ROC Curve : une mesure de la performance d'un modèle de classification binaire en termes de sa capacité à discriminer entre les classes positif et négatif. Le résultat obtenu :

```
Testing ROC-AUC Score:  0.9985324947589099
```

B) All Features :

La même chose que la régression linéaire avec sélection des caractéristiques, sauf que cette fois ci, on prend toutes les features que l'on dispose.

En utilisant une heatmap, on peut visualiser la corrélation entre la variable cible **"diagnosis"** et les autres variables comme le montre cette figure :



De la même manière, on enchaîne avec les étapes du Data Splitting & Feature Scaling, entraîner le modèle et enfin l'évaluer :

1)Training Report et sa matrice de confusion :

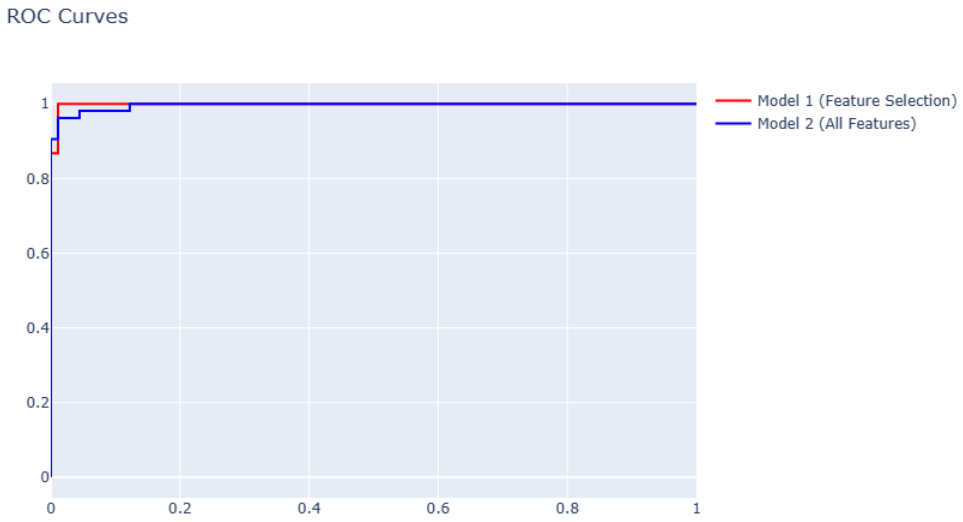
2)Testing Report et sa matrice de confusion :

3)Precision-Recall Curve et ROC Curve : le résultat obtenu :

```
Testing ROC-AUC Score:  0.9962264150943396
```

C) Comparaison :

Pour comparer les performances de deux modèles : Model 1 (**Feature Selection**) et Model 2 (**All Features**), on a opté pour les tracés des courbes **ROC** (Receiver Operating Characteristic). PS: Les valeurs des taux de faux positifs et des taux de vrais positifs sont passées comme arguments.



Comme on peut le voir sur le tracé, et comme c'est déjà mentionné, les performances sont satisfaisantes pour les deux modèles avec une légère différence.

Le classificateur avec sélection des features a atteint un taux de vrai positive==1 bien avant le classificateur sans sélection. Ce qui prouve que notre sélection a été réussie et ceci est grâce à l'ajustement des paramètres du modèle choisi.

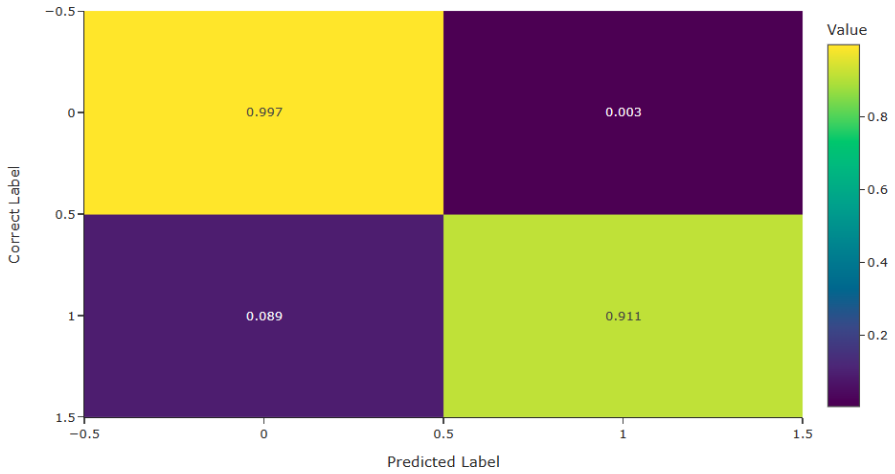
2. LDA (Linear Discriminant Analysis) :

L'analyse discriminante linéaire (LDA) est une technique de classification supervisée utilisée pour trouver une projection linéaire des données qui maximise la séparation entre les classes. Tout d'abord, on crée une instance de la classe **LinearDiscriminantAnalysis.**

Une fois que le modèle LDA est ajusté aux données d'entraînement, on peut l'utiliser pour effectuer des prédictions sur les données de validation. En utilisant la projection des données dans l'espace discriminant, le classificateur LDA attribue des étiquettes de classe aux nouvelles instances en se basant sur la proximité de chaque instance aux centroïdes des classes.

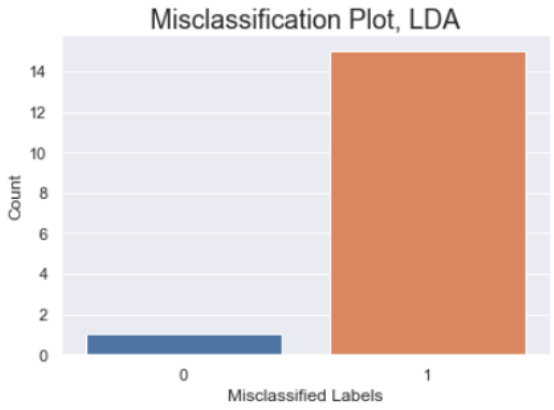
Le tracé de la matrice de confusion suivant permet de visualiser les performances du modèle LDA en termes de classification correcte et incorrecte des échantillons pour chaque classe.

On a utilisé la bibliothèque **Plotly** pour l'afficher sous forme d'une heatmap :



Pour évaluer les performances de ce modèle, on a choisi deux métriques :

- Le graphique à barres horizontales qu'est utilisé pour visualiser les étiquettes de classe mal classées par le modèle et cela en utilisant la bibliothèque **Seaborn :**



- Afficher le nombre de cas mal classés par le modèle LDA, ainsi que le taux d'erreur correspondant :

```
16 cases out of 455 cases are being misclassified by the LDA which gives us an error rate of 3.516
```

Une erreur de classification de 3.516% peut généralement être considérée comme relativement faible. Cela signifie que la LDA parvient à classer correctement la grande majorité des cas dans le jeu de données d'entraînement.

3. KNN (K Nearest Neighbors) :

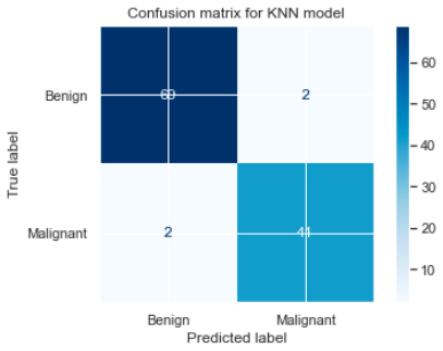
Dans ce qui suit, on va utiliser une méthode bien connue pour la classification supervisée appelée **k-NN**. On effectue également une sélection des hyperparamètres et une validation croisée pour évaluer la performance du modèle.

À l'aide de la fonction **KNeighborsClassifier** on crée une instance du modèle KNN. Ici, le modèle utilisera les quatre voisins les plus proches pour prédire la classe d'un nouvel exemple.

Ensuite, on a effectué une recherche par grille en utilisant la fonction **GridSearchCV** pour trouver les meilleurs hyperparamètres pour le modèle KNN. Voici les résultats obtenus ainsi que la précision de ce modèle :

```
The best parameters for KNN are: "{'algorithm': 'auto', 'n_neighbors': 13, 'weights': 'uniform'}"
This model can explain the dataset with an accuracy of 93.32 %
```

On peut aussi afficher la matrice de confusion pour ce modèle à l'aide de la fonction **DisplayConfusionMatrix** :



Ces résultats suggèrent que le modèle KNN avec les meilleurs hyperparamètres trouvés présente une performance solide dans la classification des données.

Enfin, le modèle est évalué en utilisant la cross validation avec la fonction **cross_val_score** de la bibliothèque **model_selection**. Ici, le modèle est exécuté avec les hyper paramètres par défaut et est validé avec une validation croisée à 10 plis.

Le score moyen obtenu est de :

```
Score (KNN default CV): 0.922713
```

Cette valeur élevée indique une performance globalement bonne du modèle avec les hyper paramètres par défaut. Cela suggère que le modèle KNN est capable de bien généraliser et de maintenir une précision élevée sur des données inconnues.

4. Decision Tree Classifier :

Le classifieur d'arbre de décision est un modèle d'apprentissage automatique qui construit un arbre basé sur les caractéristiques des données pour effectuer des prédictions de classification. Pour se faire, un classifieur d'arbre de décision est créé avec la classe **DecisionTreeClassifier.**

Ensuite, une recherche des meilleurs hyperparamètres en utilisant la méthode **GridSearchCV** a été effectuée, ce qui permet d'optimiser les performances du modèle en ajustant les hyperparamètres pour trouver la meilleure configuration de l'arbre de décision.

Une fois la recherche terminée, le modèle est ajusté à l'ensemble d'entraînement (X_train et y_train) avec les meilleurs hyperparamètres trouvés présentés dans la figure suivante :

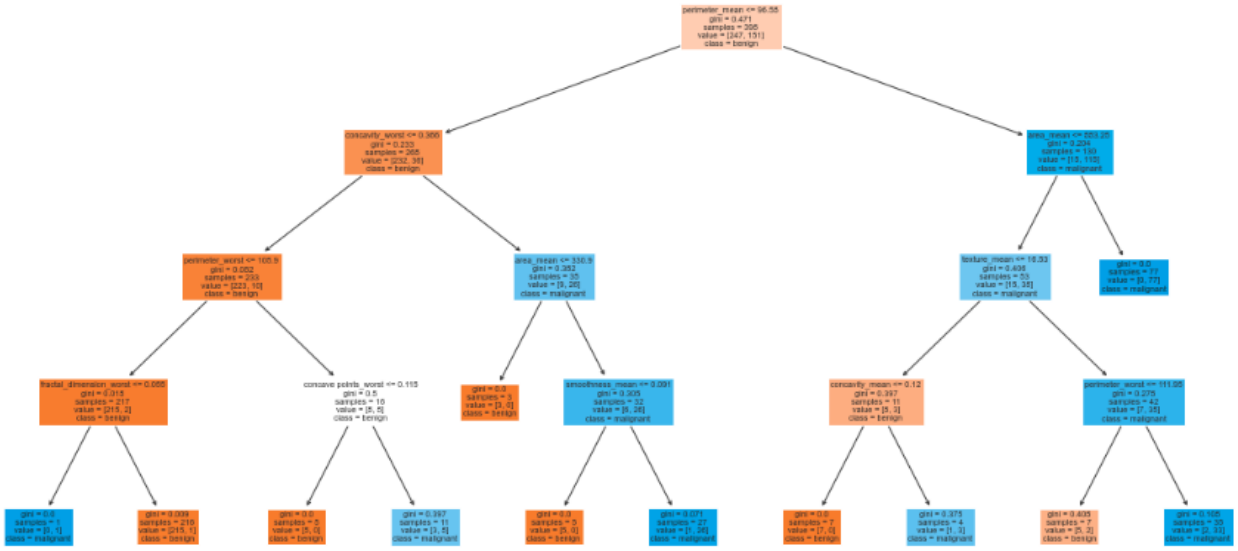
```
{'criterion': 'gini', 'max_depth': 4, 'max_features': 4}
```

Le modèle résultant a été ensuite utilisé pour faire des prédictions sur l'ensemble de validation. Les résultats des métriques de performance pour ce modèle s'affichent comme suit :

```
Accuracy Score :  0.9532163742690059
Recall Score (how much of malignant tumours were predicted correctly) :  0.9508196721311475
Precision Score (how much of tumours, which were predicted as 'malignant', were actually 'malignant'):  0.9206349206349206
```

Ces scores indiquent que le modèle d'arbre de décision a de bonnes performances en termes d'exactitude globale, de rappel pour la détection des tumeurs malignes et de précision dans la prédiction des tumeurs malignes.

Pour bien comprendre la structure de l'arbre de décision et les règles de décision prises par le modèle pour classer les données. On l'a visualisé graphiquement à l'aide de la fonction **plot_tree** de la bibliothèque scikit-learn (sklearn). Le résultat est le suivant :



IV. Conclusion

Dans le cadre de ce projet de Machine Learning, nous avons exploré et analysé différentes méthodes de prédiction du cancer du sein. Notre objectif était de se familiariser avec les différents modèles existants et tester leur performance en utilisant une variété de métriques.

Nous avons tout d'abord effectué une collecte de données et une analyse exploratoire approfondie pour comprendre les caractéristiques des tumeurs et leur relation avec le diagnostic de cancer du sein. Ensuite, nous avons procédé à un prétraitement des données, incluant la normalisation et la gestion des valeurs manquantes, ou on a vérifié que notre Dataset est prêt pour la principale étape, à savoir : La Modélisation.

Nous avons également réalisé des visualisations des données pour mieux comprendre la nature des caractéristiques et leur relation avec le diagnostic du cancer du sein. Comme mentionné un peu plus haut, nous avons utilisé des graphiques tels que des histogrammes, des diagrammes en boîte, des graphiques de dispersion et des matrices de corrélation (pour l'**ACP** et le **T-SNE**) pour visualiser les distributions, les tendances et les relations entre les variables.

Ensuite, nous avons testé plusieurs modèles de machine learning, dont la **Régression Logistique** en deux variantes : **Feature Selection** & **All Features** ou on a conclu avec une petite comparaison entre eux. Ensuite, on a testé d'autres modèles, à savoir la **LDA**, le **KNN** et l'**Arbre de décision**. Chaque modèle a été évalué en utilisant des métriques telles que l'exactitude, la sensibilité, la précision, la courbe ROC-AUC,... Les résultats ont montré que tous les modèles ont atteint de bons niveaux de performance, avec des scores d'exactitude supérieurs à 90%.

En fin de compte, ce projet a démontré l'importance de l'exploration des données et de la sélection de fonctionnalités pour la création de modèles de classification précis. Il a également souligné l'importance de l'évaluation rigoureuse des performances des modèles à l'aide de différentes métriques, afin d'obtenir une image complète de leur efficacité.

Il est important de noter que malgré les performances prometteuses des modèles testés, il existe des limites et des aspects à prendre en compte. Les performances peuvent varier en fonction de la qualité des données et de la représentativité de l'échantillon.