Process EDA (step by step + codes)


```
# si pas déjà fait :

pip install statannotations
pip install statannot
```

```
from ydata_profiling import ProfileReport
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from statannot import add_stat_annotation
from spicy.stats import ttest_ind
import random
from random import sample
```


Commenter les observations.

3.1 .head()

3.2 .tail()

3.3 .sample()

3.4 .info()

Dataset Cancer Breast : dernière colonne à supprimer.

3.5 .nunique()

```
def count_unique_values(dataframe):
    unique_counts_df = dataframe.nunique().to_frame()
    unique_counts_df.reset_index(inplace=True)
    unique_counts_df.rename(columns={'index': 'column', 0: 'n
    return unique_counts_df
```

3.6 .isna().sum()

```
def count_isna(df):
    isna_df= df.isna().sum().to_frame()
    isna_df.reset_index(inplace = True)
    isna_df.rename(columns={'index':'column', 0: 'nb_isna'},
    isna_df= isna_df.loc[(isna_df != 0).all(axis=1)]
    return isna_df
```

3.7 .duplicates()

```
def find_duplicates(df):
    duplicates_mask = df.duplicated(keep=False)
    duplicates_df = df[duplicates_mask]
    return duplicates_df
```

3.8 analyse des zéros

```
def count_iszero(df):
    df_iszero = pd.DataFrame(df[df == 0].count())
    df_iszero.rename(columns={0: 'nb_iszero'}, inplace = True
    df_iszero = df_iszero.loc[~(df_iszero == 0).all(axis=1)]
    return df_iszero
```

3.9 .describe()

3.10 .value_counts() sur la variable “target” pour voir combien on a de patients sains et malades

4. Traitement

4.1 Valeurs aberrantes

ex: CKD: \t et valeurs multipliées par 10 à corriger

```
# Fonction pour trouver les erreurs de saisies (tabulations
def find_tabs_and_spaces(dataset):
    issues_found = []

    for index, row in dataset.iterrows():
        for column in dataset.columns:
            cell_value = row[column]
            if isinstance(cell_value, str):
                if '\t' in cell_value:
                    issues_found.append(f'Ligne {index}, Co
                if ' ' in cell_value:
                    issues_found.append(f'Ligne {index}, Co

    return issues_found

# Résolution
def remplacement(df):
    columns = df.select_dtypes(include='object').columns
    for column in columns:
        df[column] = df[column].apply(lambda x: x.replace('
    return df

df_ckdc['dm'] = df_ckdc['dm'].replace({' yes': 'yes'})
```

```
# Détection des Outliers

def detect_outliers(dataframe, threshold=1.5):
```

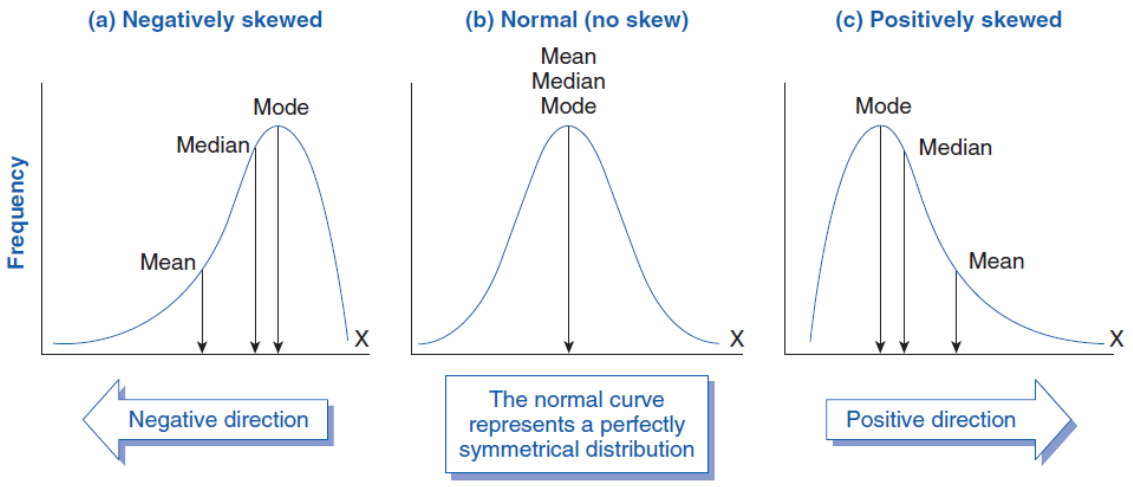
```
    for col in dataframe.columns:
        if dataframe[col].dtype in ['int64', 'float64']:
            # Calcul de l'IQR
            Q1 = dataframe[col].quantile(0.25)
            Q3 = dataframe[col].quantile(0.75)
            IQR = Q3 - Q1

            # Détection des outliers
            lower_bound = Q1 - threshold * IQR
            upper_bound = Q3 + threshold * IQR

            col_outliers = dataframe[(dataframe[col] < lowe

            # Tri des outliers par ordre croissant
            sorted_outliers = col_outliers.sort_values()

            print(f"Column: {col}")
            print(f"Number of Outliers: {len(sorted_outlier
            print(f"Outliers: {sorted_outliers.values}\n{'=
```

4.2 Stocker les colonnes numériques et catégorielles dans 2 variables distinctes.

Attention, les variables catégorielles peuvent ressembler à du numérique (0, 1, 2...)

```
# Définir un seuil pour le nombre de valeurs uniques pour d
threshold_unique_values = 10  # Ajustez ce seuil en fonctio

# Identifier les variables catégorielles (nombre de valeurs
categorical_vars = [col for col in df.columns if df[col].nu

# Identifier les variables numériques continues (nombre de
numeric_continuous_vars = [col for col in df.columns if col
```

4.3 Si Nan: .fillna() avec la médiane ou la moyenne selon la skweness (asymétrie)

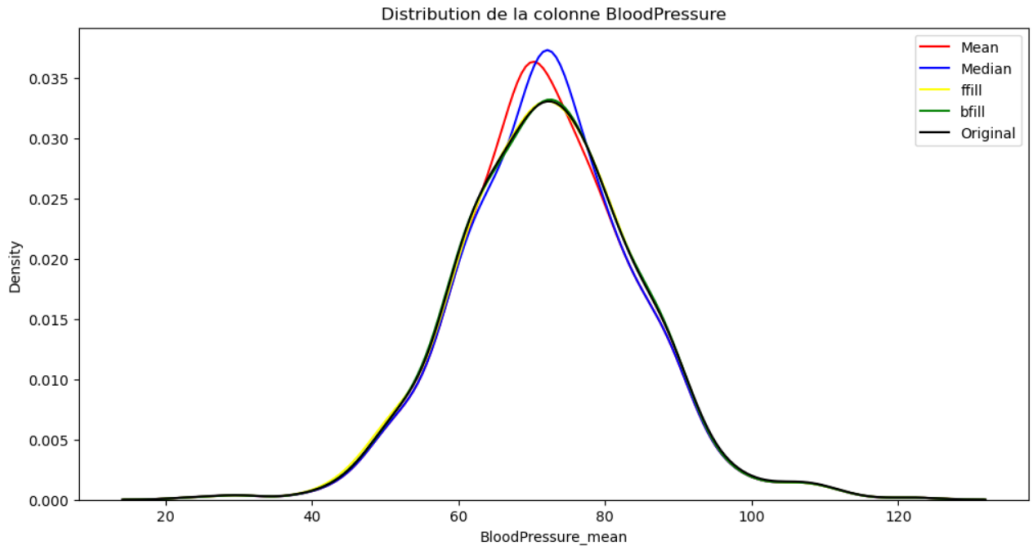


Nan dans variables numériques :

- Right skewed/left skewed : on remplace par la moyenne ou la médiane selon ce qui donne la meilleure skewness (par classe de patients).

  https://www.scribbr.com/frequently-asked-questions/whats-the-best-measure-of-central-tendency-to-use/

  - Cas particulier dataset CKD : si sur une ligne, tu as plus de 3 Nan, demander au client pour suppression de la ligne = le client valide la suppression.
  - CKD : plus de 50 valeurs manquantes pour une variable = on fait une random imputation
  - Moins de 50 Nan : remplacer par la moyenne ou la médiane.




```
def impute_and_plot_all_columns(data):
    # Copie du DataFrame pour éviter de modifier l'original
    data_imputed = data.copy()

    # Parcourir toutes les colonnes avec des valeurs manqua
    for col in data.columns[data.isna().any()]:
        # Calculer la moyenne et la médiane de la colonne
        col_mean = data[col].mean()
        col_median = data[col].median()

        # Imputer les valeurs manquantes avec la moyenne, l
        data_imputed[f'{col}_mean'] = data[col].fillna(col_
        data_imputed[f'{col}_median'] = data[col].fillna(co
        data_imputed[f'{col}_ffill'] = data[col].fillna(met
        data_imputed[f'{col}_bfill'] = data[col].fillna(met

        # Tracer les distributions résultantes
        plt.figure(figsize=(12, 6))
        sns.kdeplot(data_imputed[f'{col}_mean'], color='red
        sns.kdeplot(data_imputed[f'{col}_median'], color='b
        sns.kdeplot(data_imputed[f'{col}_ffill'], color='ye
        sns.kdeplot(data_imputed[f'{col}_bfill'], color='gr
        sns.kdeplot(data[col], color='black', label='Origir
        plt.title(f'Distribution de la colonne {col}')
```

```
        plt.legend()
        plt.show()
```

Après visualisation, on définit comment remplir :

- Si on veut remplacer les Nan par la médiane (distinction sur classes malade / sain) :

```
def remplacer_nan_par_mediane_selon_etat(dataset, colonn
    # Séparer les lignes malades et saines
    malades = dataset[dataset[colonne_categorie] == 'mal
    sains = dataset[dataset[colonne_categorie] == 'sain'

    # Boucler sur les colonnes numériques
    for colonne in dataset.columns:
        # Vérifier si la colonne est numérique (moins de
        if dataset[colonne].nunique() <= 10 and pd.api.t
            # Calculer la médiane pour les patients mala
            mediane_malades = malades[colonne].median()
            mediane_sains = sains[colonne].median()

            # Remplacer les NaN par la médiane correspon
            dataset.loc[dataset[colonne_categorie] == 'm
                dataset.loc[dataset[colonne_categorie] =

            dataset.loc[dataset[colonne_categorie] == 's
                dataset.loc[dataset[colonne_categorie] =

    return dataset
```

- Si on veut remplacer les Nan par la moyenne (distinction sur classes malade / sain)

```
def remplacer_nan_par_moyenne_selon_etat(dataset, colonn
    # Séparer les lignes malades et saines
    malades = dataset[dataset[colonne_categorie] == 'mal
    sains = dataset[dataset[colonne_categorie] == 'sain'

    # Boucler sur les colonnes numériques
```

```
    for colonne in dataset.columns:
        # Vérifier si la colonne est numérique (moins de
        if dataset[colonne].nunique() <= 10 and pd.api.t
            # Calculer la moyenne pour les patients mala
            moyenne_malades = malades[colonne].mean()
            moyenne_sains = sains[colonne].mean()

            # Remplacer les NaN par la moyenne correspon
            dataset.loc[dataset[colonne_categorie] == 'm
                dataset.loc[dataset[colonne_categorie] =

            dataset.loc[dataset[colonne_categorie] == 's
                dataset.loc[dataset[colonne_categorie] =

    return dataset
```

- Si on veut remplacer les Nan avec le ffill ou bfill (pas de distinction entre patients malades et sains)

```
def fillna_with_ffill(df):
    for col in df.columns:
        if col in columns_with_less_than_50_nan:
            df_col1 = df[col].fillna(method='ffill')
            if df_col.isna().sum() != 0:
                df[col].fillna(method='mean', inplace=Tr
                print(f"Le ffill n'a pas pu remplir enti
            else:
                df[col].fillna(method='ffill', inplace=T
                print(f"Le ffill a pu remplir entièremen
```

Nan dans variables non-numériques/catégorielles :

- Utiliser le mode sampling (qui nécessite préalablement d'effectuer une factorisation) quand on a peu de valeurs Nan à traiter (moins de 50).

```

```

- Faire du random sampling quand il y a beaucoup de valeurs Nan à traiter (plus de 50).

```
def traiter_nan_cat(dataset):
    # Récupérer la liste des colonnes avec moins de 3 va
    colonnes_cat = dataset.columns[dataset.nunique() < 3

    # Boucler sur les colonnes catégorielles avec NaN
    for colonne in colonnes_cat:
        # Vérifier si la colonne a des NaN
        if dataset[colonne].isna().any():
            # Vérifier le nombre de NaN dans la colonne
            nb_nan = dataset[colonne].isna().sum()

            # Remplacer NaN par le mode si le nombre de
            if nb_nan < 50:
                mode_colonne = dataset[colonne].mode()[0
                dataset[colonne].fillna(mode_colonne, in
            else:
                # Remplacer NaN par random sampling si l
                valeurs_existants = dataset[colonne].dro
                valeurs_random = np.random.choice(valeur
                dataset[colonne].loc[dataset[colonne].is

    return dataset
```

```
# Random Sampling de Gwen

df_ckdc = df_ckdc.apply(lambda x: np.where(x.isnull() &
```

- Variables “redbloodcells” et “pus_cell” : faire du random sampling.
- Les autres categorical columns : utiliser le mode.
- Une variable n’est considérée comme catégorielle **dans le dataset CKD que si il y a 2 catégories** (pas plus).

4.4 Doublons

Supprimer les doublons le cas échéant.

```
df.drop_duplicates(inplace = True)
```

4.5 Zéro

- Si variables catégorielles : c'est normal.
- Si 0 est une valeur normale pour cette variable : OK.
- Si 0 est une valeur manquante : cf process de gestion des Nan.
    - Cas particulier dataset Diabète :
        - "SkinThickness" : choisir en fonction de la courbe puis on entrainera les modèles de ML avec et sans cette variable pour voir si on la conserve ou non.
        - "BMI" : utiliser la médiane.
        - "Glucose" : choisir en fonction de la courbe.
        - "BloodPressure" : choisir en fonction de la courbe.
        - "Insulin" : choisir en fonction de la courbe.

```
'''def pour afficher les courbes de distribution
selon la méthode de remplacement des 0 choisie'''

def impute_and_plot_columns_with_0(data):
    data_imputed = data.copy()
    for col in data.columns:
         # Calculer la moyenne et la médiane de la colonne
        col_mean = data_imputed[col].mean()
        col_median = data_imputed[col].median()

        data_imputed[col].replace(0,  np.nan, inplace = Tru


                # Imputer les valeurs manquantes avec la mo
        data_imputed[f'{col}_mean'] = data_imputed[col].fil
        data_imputed[f'{col}_median'] = data_imputed[col].f
        data_imputed[f'{col}_ffill'] = data_imputed[col].fi
        data_imputed[f'{col}_bfill'] = data_imputed[col].fi

                # Tracer les distributions résultantes
        plt.figure(figsize=(12, 6))
        sns.kdeplot(data_imputed[f'{col}_mean'], color='red
```

```
        sns.kdeplot(data_imputed[f'{col}_median'], color='b
        sns.kdeplot(data_imputed[f'{col}_ffill'], color='ye
        sns.kdeplot(data_imputed[f'{col}_bfill'], color='gr
        sns.kdeplot(data_imputed[col], color='black', label
        plt.title(f'Distribution de la colonne {col}')
        plt.legend()
        plt.show()
```

```
# fonction pour remplacer les zéros avec un forward fill
def fill_0_ffill(df):
    df.replace(0,  np.nan, inplace = True)
    df.fillna(method='ffill', inplace=True)


fill_0_ffill(df_diabetes['Glucose'])
```

```
# fonction pour remplacer les zéros avec un backward fill
def fill_0_bfill(df):
    df.replace(0,  np.nan, inplace = True)
    df.fillna(method='bfill', inplace=True)
```

```
# fonction pour remplacer les zéros avec la moyenne
# (avec distinction entre les classes malades et sains)

def fill_0_mean(dataset, colonne_a_modifier, colonne_catego
     # Séparer les lignes malades et saines
    classe0 = dataset[dataset[colonne_categorie] == 0]
    classe1 = dataset[dataset[colonne_categorie] == 1]
    moyenne_classe0 = classe0[colonne_a_modifier].mean()
    moyenne_classe1 = classe1[colonne_a_modifier].mean()

    dataset[colonne_a_modifier].replace(0,  np.nan, inplace
    dataset[colonne_a_modifier].fillna(method='bfill', inpl
    # Remplacer les NaN par la moyenne correspondante en fo
    dataset.loc[dataset[colonne_categorie] == 'classe0', co
                dataset.loc[dataset[colonne_categorie] == '
```

```
    dataset.loc[dataset[colonne_categorie] == 'classe1', cc
                dataset.loc[dataset[colonne_categorie] == '
    return dataset


fill_0_mean(df_diabetes, 'BMI', 'Outcome')
```

- Cas particuliers dataset Cancer Breast : 13 lignes avec toutes les valeurs concavité à 0 = lignes à supprimer.

```
df_cancerbreast.loc[df_cancerbreast['concavity_mean']==0]
rows_to_suppress = df_cancerbreast.loc[df_cancerbreast['cor
index_to_suppress = rows_to_suppress.index.tolist()
df_cancerbreast.drop(index_to_suppress, inplace= True)
```

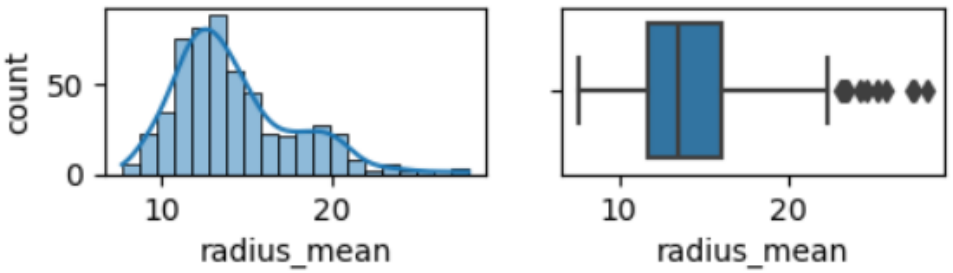
5. Analyse statistique

Maintenant que le dataset est “propre”, je propose un nouveau .describe()

5.1 Analyse univariée

5.1.1. Sur l’ensemble des colonnes :

- Histogramme + boxplot pour les variables numériques.
- Histogramme seulement pour les variables catégorielles.




```
def histo_moustaches_numerical(dataframe):
    num_df = dataframe.select_dtypes(include=np.number)
    for col in num_df:
```

```
        plt.figure(figsize = (5, 1))
        plt.subplot(1, 2, 1)
        sns.histplot(num_df, x= num_df[col], kde=True)
        # plt.title(col)
        plt.ylabel('count')
        plt.subplot(1, 2, 2)
        sns.boxplot(x=num_df[col])
        #plt.title(col)
        plt.show()
```

```
# Permet de faire la disction de la distribution sur tous l
# Permet de voir si une classe influence la distribution de

def histo_by_target(dataframe, target_col='target'):
    num_df = dataframe.select_dtypes(include=np.number)

    for col in num_df:
        plt.figure(figsize=(15, 5))

        # Histogramme pour tous les patients
        plt.subplot(1, 3, 1)
        sns.histplot(dataframe, x=col, kde=True, element="s
        plt.title(f'All Patients - {col}')
        plt.xlabel(col)
        plt.ylabel('Count')

        # Histogramme pour tous les patients malades (targe
        plt.subplot(1, 3, 2)
        sns.histplot(dataframe[dataframe[target_col] == 1],
        plt.title(f'Traget 1 - {col}')
        plt.xlabel(col)
        plt.ylabel('Count')

        # Histogramme pour tous les patients sains (target
        plt.subplot(1, 3, 3)
        sns.histplot(dataframe[dataframe[target_col] == 0],
        plt.title(f'Target 0 - {col}')
        plt.xlabel(col)
```

```
        plt.ylabel('Count')

        plt.tight_layout()
        plt.show()

histo_by_target(df_heart_disease, target_col='target')
```

5.1.2. Zoom sur les colonnes qui seraient à creuser (outliers) avec des graphiques interactifs

```
# à retravailler pour que ça fonctionne avec une colonne er

def histo_moustaches_numerical(dataframe):
    num_df = dataframe.select_dtypes(include=np.number)
    for col in num_df:
        fig = make_subplots(rows=1, cols=2,subplot_titles=(

        fig.add_trace(go.Histogram(x= num_df[col],name= "")
                      row=1, col=1)
        fig.update_layout(showlegend= False)
        fig.add_trace(go.Box(x=num_df[col],name= ""),
                      row=1, col=2)

        fig.update_layout(height=300, width=800)
        fig.show()
```

5.1.3. Ecarts interquartiles

```
# à utiliser avec les valeurs numériques uniquement: df[num

def calculate_iqr(dataframe):

    q1 = dataframe.quantile(0.25)
    q3 = dataframe.quantile(0.75)
    iqr_values = q3 - q1
```

```
print(iqr_values)
```

5.2 Analyse bivariée

5.2.1 Boxplots pour chaque colonne avec en “hue” le diagnostic (patients malade ou non)

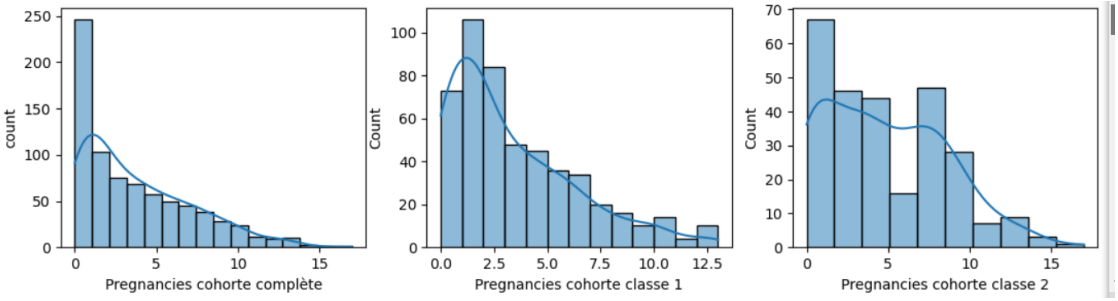
```
'''AVEC TESTS STATISTIQUES'''
'''conversion en str du diagnostic indispensable.
On le fait en dbt de fonction et on repasse en int en fi

def boxplot_for_each_column(dataframe, column_diagnostic
    dataframe[column_diagnostic] = dataframe[column_diag
    df_y= dataframe.drop([column_diagnostic], axis=1)
    df_x= dataframe[column_diagnostic]
    plt.figure(figsize=(15, 8))
    for column in df_y.columns:
        plt.subplot(1,len(df_y.columns), df_y.columns.ge
        ax = sns.boxplot(y=df_y[column], x= df_x)
        add_stat_annotation(ax, data=dataframe, x= colum
                    box_pairs=[("0", "1")],
                    test='Kruskal', text_format='full',
        plt.title(column)
    plt.tight_layout()
    plt.show()
    dataframe[column_diagnostic] = dataframe[column_diag
```

- Pour les valeurs non-numériques, il y 2 solutions. Soit on fait deux histogramme côte à côte pour mettre en lumière les différences entre patients sains et malades, soit on fait un histogramme empilé avec les patients malades et sains = on teste les 2 solution et on conserve celle qui offre le plus de visibilité.

5.2.2 Focus sur les colonnes pour lesquelles on ne voit pas la différence entre patients sains et patients malades = créer une définition de fonction.

```
# CODE RETOUCHE POUR AVOIR LA DISTRIBUTION SUIVANT LES M
```

```
def histo_numerical(dataframe, colonne_diagnostic, sain,
    for col in dataframe.columns:
        plt.figure(figsize=(13, 3))

        # Cohorte complète
        plt.subplot(1, 3, 1)
        sns.histplot(data=dataframe, x=dataframe[col], k
        plt.ylabel('count')
        plt.xlabel(f'{col} - Cohorte complète')

        # Patients sains
        plt.subplot(1, 3, 2)
        sns.histplot(data=dataframe[dataframe[colonne_di
        plt.xlabel(f'{col} - Cohorte {sain}')

        # Patients malades
        plt.subplot(1, 3, 3)
        sns.histplot(data=dataframe[dataframe[colonne_di
        plt.xlabel(f'{col} - Cohorte {malade}')

        plt.show()
```



5.3. Analyse multivariée

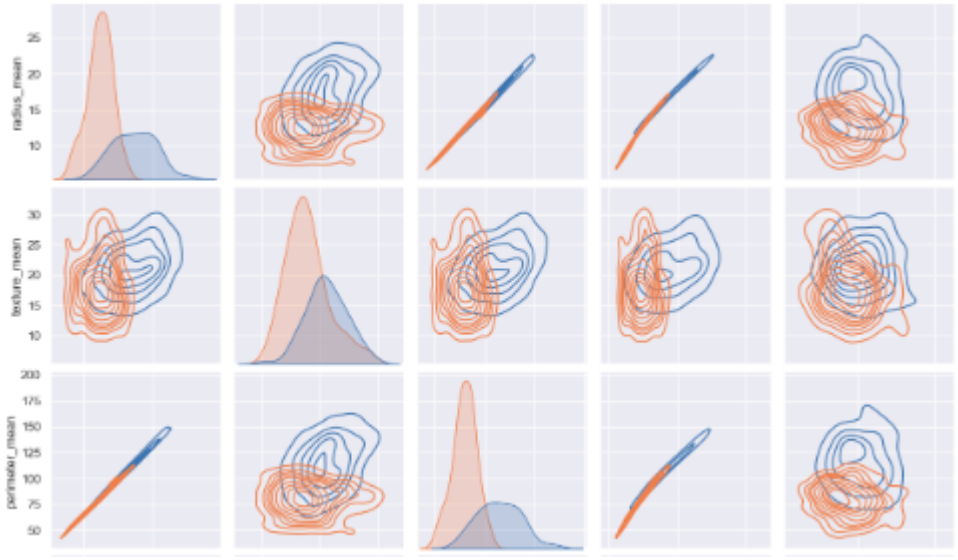
5.3.1 Pairplots

```
# def de Félix. ne fonctionne pas chez Caro
def plot_pairplot_numeric(dataframe):
    # Sélectionner les colonnes numériques continues
    numeric_columns = dataframe.select_dtypes(include=['
```

```
    # Filtrer les colonnes qui ne sont pas des colonnes
    numeric_continuous_columns = [col for col in numeric

    # Créer une sous-DataFrame avec les colonnes numériq
    subdata_numeric = dataframe[numeric_continuous_colum

    # Tracer le pairplot avec seaborn
    sns.pairplot(subdata_numeric)

    # Afficher le pairplot
    plt.show()

# def de Caroline
'''import seaborn as sns
import matplotlib.pyplot as plt
import random
from random import sample'''

def pairplot(data, diagnostic):
    df_pairplot = data.sample(frac= 0.4, random_state =
    pairplot = sns.pairplot(df_pairplot, hue = diagnosti
    return pairplot

pairplot(df_cancerbreast[colonnes_num], "diagnosis_facto
```

- Cas particulier Cancer Breast: on fait les graphiques multivariés sur les 3 groupes mais on ne garde qu'un groupe pour la programmation des modèles de ML (spoiler alert : on va utiliser les moyennes).
- Pairplot (autre version, Ju)

```
def pairplot_2(data, diagnostic):
        df_pairplot = data.sample(frac= 0.4, random_s
        pairplot = sns.pairplot(df_pairplot, hue = di
        return plt.show()
pairplot_2(df_bcancer_mean, "diagnosis")
```



5.3.2 Heatmaps

```
def plot_triangle_corr_heatmap(dataframe):
    # Calculer la matrice de corrélation
    corr_matrix = dataframe.corr()

    # Créer un masque pour le triangle supérieur
    mask = np.triu(np.ones_like(corr_matrix))

    # Tracer la heatmap avec seaborn
    plt.figure(figsize=(20, 8))
    heatmap = sns.heatmap(corr_matrix, mask=mask, vmin=-
                          annot=True, cmap='BrBG', ann
    heatmap.set_title('Triangle Correlation Heatmap', fo

    # Afficher la heatmap
    plt.show()
```

6. Conclusions de l’exploration des données

CKD

Suite à l'analyse de la matrice de corrélation, plusieurs conclusions peuvent être tirées :

Les variables ne présentent pas de fortes corrélations entre elles, à l'exception d'une corrélation notable de 0.70 entre l'hémoglobine et le Packed Cell Volume.

En ce qui concerne le diagnostic (classification_facto), trois variables semblent être corrélées négativement : sg avec une corrélation de -0.71, hemo avec -0.65, et pcv avec -0.66.

En considération de ces résultats, la décision a été prise de conserver toutes les variables pour l'application de modèles de machine learning. La faible corrélation entre les variables suggère qu'elles fournissent des informations distinctes, tandis que la corrélation négative avec le diagnostic pourrait indiquer un potentiel de prédiction significatif. Cette approche permettra d'explorer le rôle de chaque variable dans le modèle sans exclure des informations potentiellement pertinentes.

CANCERBREAST

Le périmètre, l'aire et le rayon sont très corrélés entre eux.
Proposition: Entrainer les ML en n'en conservant qu'un des trois. réponse du client ?

Chacune des variables est disponible dans 3 groupes: (mean/ standard error et worst)
Proposition: Ne conserver que les mean pour le ML. réponse du client ?

Pour chacune des variables dans les 3 groupes (mean/ standard error et worst), la valeur p ajustée est généralement inférieure à 0,05, sauf pour le groupe standard error.

La valeur p est supérieure à 0.05 pour la fractale dimension. Nous pensons tester le ML avec et sans cette variable.

DIABETE

L'âge et le nombre de grossesses semblent corrélés. Cette corrélation est sans rapport avec le diabète.

L'indice de masse corporelle et la SkinThickness sont également positivement corrélés. On peut envisager d'entrainer des modèles de Machine Learning avec seulement une de ces 2 variables et avec les 2 pour voir celui qui donne les meilleurs résultats.

MALADIES CARDIAQUES

Nous pouvons voir que les variables sont peu corrélées entre elles. En effet, la faible corrélation entre les variables peut être bénéfique pour plusieurs raisons.

Évitement de la redondance : Lorsque les variables sont fortement corrélées, elles apportent souvent des informations similaires au modèle. Cela peut conduire à une redondance dans les prédictions, ce qui n'ajoute pas nécessairement de la valeur et peut même introduire du bruit.

Réduction des biais : Des corrélations élevées entre les variables peuvent entraîner des biais dans le modèle, car il peut être difficile de discerner l'effet spécifique de chaque variable sur la variable cible. Une faible corrélation peut aider à isoler les effets individuels des variables indépendantes.

Interprétabilité accrue : Les modèles avec des variables faiblement corrélées peuvent souvent être plus faciles à interpréter, car il est plus simple d'attribuer des effets spécifiques à chaque variable sans qu'elles se chevauchent excessivement.

Stabilité des prédictions : Les modèles basés sur des variables faiblement corrélées peuvent être plus stables et généralisables à de nouvelles données, car ils ne sont pas aussi sensibles aux variations spécifiques dans une seule variable.

MALADIES DU FOIE

Étant donné une forte corrélation entre les variables 'Direct_Bilirubin', 'Aspartate_Aminotransferase', 'Total_Protiens' et 'Albumin', il est recommandé de les retirer du jeu de données lors de l'entraînement des modèles de machine learning. Cela peut contribuer à améliorer la performance du modèle en évitant la redondance et en favorisant une meilleure interprétabilité des résultats

7. **Export du dataset nettoyé**

```
df_diabetes.to_csv(r'C:\Users\wilders\Documents\Clinical Data
```