Welcome to the module 7b coding part: Making a graph for Models comparison!

This notebook was created at San Francisco State University (SFSU) for the Promoting INclusivity and Computing (PINC) and gSTAR programs by Dr. Pleuni Pennings (SFSU biology professor), Lucy Moctezuma Tan (California State University, East Bay CSUEB master student) and Lorena Benitez-Rivera (SFSU master student). All members of the COde to understand Drug resistance Evolution (CODE) lab in 2023.

The code is based on a project by Faye Orcales, Jameel Ali, Meris Johnson-Hagler, Kristiene Recto, Lucy Moctezuma Tan (all CODE lab members at SFSU), and in turn inspired by work by Danesh Moradigaravand and coauthors Moradivaravand et al.

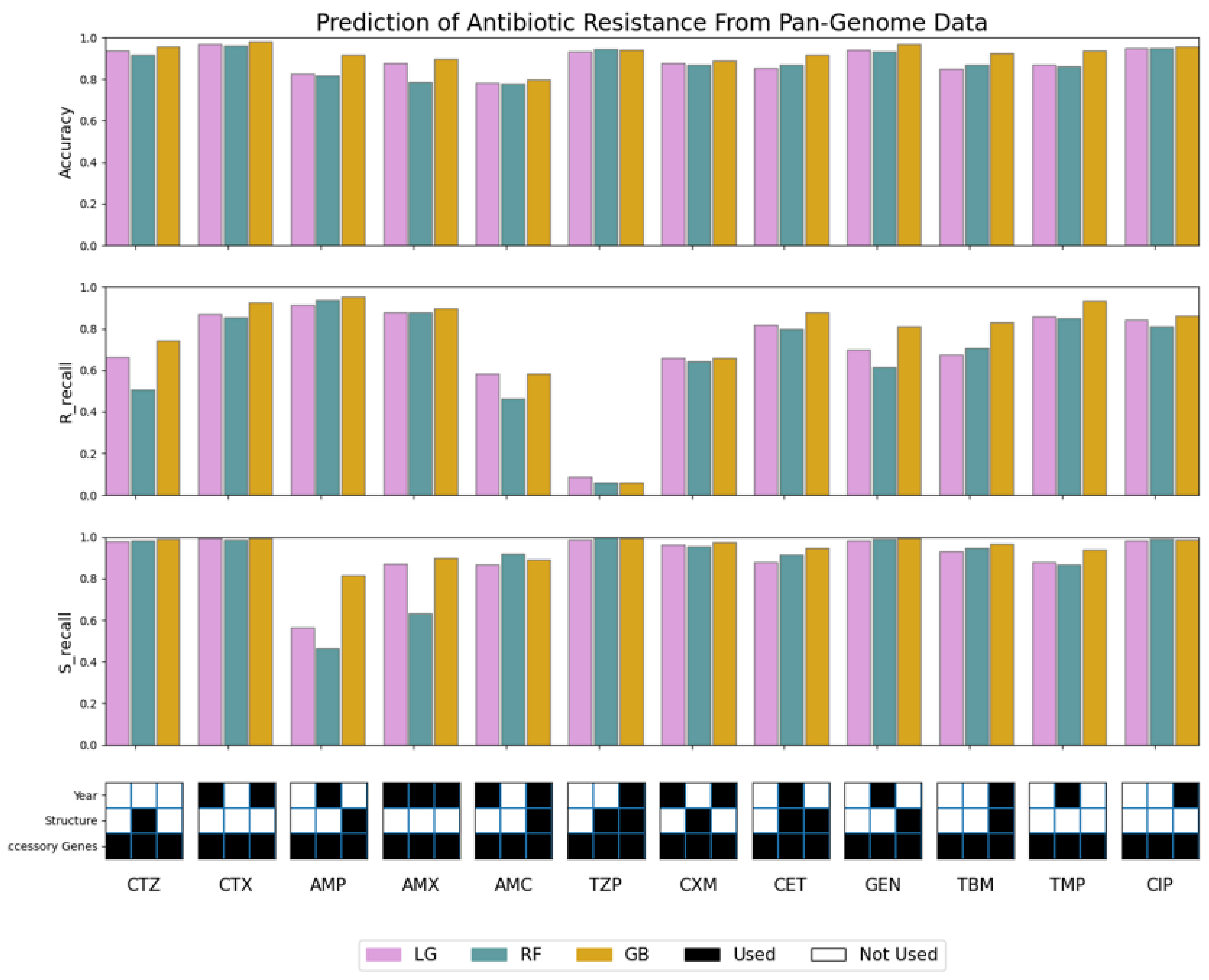
OBJECTIVE OF THIS NOTEBOOK:

Recreating graphs of Accuracies and Recall

In this notebook we are going to create a graph that will let us compare the performance for all the three models used in notebooks 6b-7a:

1. **Gradient-Boosted Trees**
2. **Random Forest**
3. **Logistic Regression**

We will also be able to check what combinations were best for each drug. Below is an example of the graph we want to achieve with this code:



Step 1) Importing packages needed for visualization

NOTE: Please allow access to your google drive when prompted, this will let you create and store the files in your drive to be accessed.

```
# Data Wrangling Imports
import pandas as pd
import numpy as np

# Data visualization Imports
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec
from matplotlib.patches import Patch

# File Manipulation Imports
import os
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

Step 2) Loading each models scores and join them into a single dictionary

```
import os

# Define the directory
filepath = '/content/drive/MyDrive/EColi_ML_CSV_files/'

# Initialize a dictionary to store model scores
Model_Scores = {}

# Verify and load each file safely
try:
    if os.path.exists(filepath + "LG_metrics_df.csv"):
        LG_metrics = pd.read_csv(filepath + "LG_metrics_df.csv")
        Model_Scores["Logistic_Regression"] = LG_metrics
    else:
        print("Error: LG_metrics_df.csv not found!")

    if os.path.exists(filepath + "RF_metrics_df.csv"):
        RF_metrics = pd.read_csv(filepath + "RF_metrics_df.csv")
        Model_Scores["Random_Forest"] = RF_metrics
    else:
        print("Error: RF_metrics_df.csv not found!")

    if os.path.exists(filepath + "GB_metrics_df.csv"):
        GB_metrics = pd.read_csv(filepath + "GB_metrics_df.csv")
        Model_Scores["Gradient_Boosted_Trees"] = GB_metrics
    else:
        print("Error: GB_metrics_df.csv not found!")
except Exception as e:
    print(f"An error occurred: {e}")

# Check loaded data
for model, metrics in Model_Scores.items():
    print(f"{model} Metrics:")
    print(metrics.head())
```

```
Error: RF_metrics_df.csv not found!
Logistic_Regression Metrics:
  Drug_combo  Accuracy  R_recall  S_recall
0      CTZ_G  0.935837  0.662921  0.980000
1      CTZ_S  0.843505  0.426966  0.910909
2     CTZ_GY  0.932707  0.651685  0.978182
3     CTZ_GS  0.841941  0.438202  0.907273
4     CTZ_SY  0.830986  0.415730  0.898182
Gradient_Boosted_Trees Metrics:
  Drug_combo  Accuracy  R_recall  S_recall
0      CTZ_G  0.956182  0.764045  0.987273
1      CTZ_S  0.893584  0.449438  0.965455
2     CTZ_GS  0.951487  0.741573  0.985455
3    CTZ_GYS  0.949922  0.741573  0.983636
4      CTX_G  0.982085  0.926230  0.995935
```

Step 3) Selecting only the bestscores for each drug from each model

```
# Extracting list of combos from dictionary created
combo_list = list(Model_Scores['Gradient_Boosted_Trees']["Drug_combo"].str.split("_", expand= True)[1].unique())
combo_list
```

```
['G', 'S', 'GS', 'GYS']
```

```
# Extracting list of drugs from dictionary created
drug_list = list(Model_Scores['Gradient_Boosted_Trees']["Drug_combo"].str[:3].unique())
drug_list
```

```
['CTZ',
 'CTX',
 'AMP',
 'AMX',
 'AMC',
 'TZP',
 'CXM',
 'CET',
 'GEN',
 'TBM',
 'TMP',
 'CIP']
```

a) Creating a function to scan the best metrics for each drug

```
# creating function that will take the best scores from each model
def Best_metrics(model,df):
    print("Selecting Best Scores for model: ",model)
    bestcores_dic = {}
    for drug in drug_list:
        data = df.loc[df["Drug_combo"].str.startswith(drug)]
        max_acc = max(data["Accuracy"])
        drug_combo = data["Drug_combo"][data["Accuracy"] == max_acc].unique()[0]
        S_rec = float(data[data["Drug_combo"] == drug_combo]["S_recall"])
        R_rec = float(data[data["Drug_combo"] == drug_combo]["R_recall"])
        bestcores_dic[drug_combo] = [max_acc, R_rec, S_rec]
    bestscores_df = pd.DataFrame.from_dict(bestcores_dic, orient ='index',columns=["Accuracy", "R_recall", "S_recall"]).reset_index()
    bestscores_df = bestscores_df.rename(columns = {'index':'Drug_combo'})
    return bestscores_df
```

b) Using the function for the results from our Logistic Regression model.

```
# Implementation of function Best_metrics() example with Logistic Regression scores
LG_best_metrics = Best_metrics("Logistic_Regression", Model_Scores["Logistic_Regression"])
LG_best_metrics
```

```
Selecting Best Scores for model:  Logistic_Regression
<ipython-input-10-027028a77d23>:9: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeEr
  S_rec = float(data[data["Drug_combo"] == drug_combo]["S_recall"])
<ipython-input-10-027028a77d23>:10: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeE
  R_rec = float(data[data["Drug_combo"] == drug_combo]["R_recall"])
```

	Drug_combo	Accuracy	R_recall	S_recall
0	CTZ_G	0.935837	0.662921	0.980000
1	CTX_GY	0.972313	0.885246	0.993902
2	AMP_GY	0.827338	0.917874	0.563380
3	AMX_G	0.870166	0.875000	0.862319
4	AMC_G	0.780180	0.576471	0.870130
5	TZP_G	0.931408	0.090909	0.984645
6	CXM_GY	0.877934	0.664804	0.960870
7	CET_G	0.852518	0.817391	0.877301
8	GEN_G	0.934272	0.686869	0.979630
9	TBM_G	0.848921	0.674157	0.931217
10	TMP_G	0.870504	0.857143	0.880503
11	CIP_G	0.951487	0.842466	0.983773

Step 4) Converting combo strings into separate columns

Combo strings for **years of isolation (Y), gene absence/presence (G) and the population structure (S)** data (features).

The function below will allow us to plot the squares below our bar graphs so that we can tell which feature combinations were used.

```
# Creating a function that takes the combo strings from Drug_combo column and turns them into separate columns
def make_GYS(bestscores_df):
    # Create 3 new columns one for each type of features (Year of isolation, gene absence or presence, and population structu
    bestscores_df["G"] = " "
    bestscores_df["Y"] = " "
    bestscores_df["S"] = " "

    # Read the combo part of Drug_combo
    split_c = bestscores_df["Drug_combo"].str.split("_", expand=True)
    i=0
    while i < len(split_c[1]):
        split_each_c = [x for x in split_c[1][i]]
        for g in split_each_c:
            if "G" in split_each_c:
                bestscores_df.at[i,"G"] = 1
            else:
                bestscores_df.at[i,"G"] = 0
        for y in split_each_c:
            if "Y" in split_each_c:
                bestscores_df.at[i,"Y"] = 1
            else:
                bestscores_df.at[i,"Y"] = 0
        for s in split_each_c:
            if "S" in split_each_c:
                bestscores_df.at[i,"S"] = 1
            else:
                bestscores_df.at[i,"S"] = 0
        i += 1
    bestscores_df["Drug_combo"] = bestscores_df["Drug_combo"].map(lambda x: x.rstrip('_GYS'))
    bestscores_df.rename(columns={"Drug_combo": "Drug"}, inplace = True)

    return bestscores_df
```

Below is an implementation of the function we just created, using our Logistic Regression results.

```
# Implementing function make_GYS()
GYS_LG_best_metrics = make_GYS(LG_best_metrics)
GYS_LG_best_metrics
```

	Drug	Accuracy	R_recall	S_recall	G	Y	S
0	CTZ	0.935837	0.662921	0.980000	1	0	0
1	CTX	0.972313	0.885246	0.993902	1	1	0
2	AMP	0.827338	0.917874	0.563380	1	1	0
3	AMX	0.870166	0.875000	0.862319	1	0	0
4	AMC	0.780180	0.576471	0.870130	1	0	0
5	TZP	0.931408	0.090909	0.984645	1	0	0
6	CXM	0.877934	0.664804	0.960870	1	1	0
7	CET	0.852518	0.817391	0.877301	1	0	0
8	GEN	0.934272	0.686869	0.979630	1	0	0
9	TBM	0.848921	0.674157	0.931217	1	0	0
10	TMP	0.870504	0.857143	0.880503	1	0	0
11	CIP	0.951487	0.842466	0.983773	1	0	0

Step 5) Extracting best metrics and coding for GYS into columns for all models

We have 3 models so the code below will implement the function we created in the previous step for all 3 models results.

```
# Getting the best metrics for all the models
Best_metrics_models = {}
for model, df in Model_Scores.items():
    # select the best scores obtained from each model
    Model_best_metrics = Best_metrics(model, df)

    # Code GYS data in 0 "for absence" and 1 "for presence" into 3 columns
    GYS_coded_best = make_GYS(Model_best_metrics)
    print(GYS_coded_best)
    # Save new dataframe in a dictionary with best metrics selected
    Best_metrics_models[model] = GYS_coded_best
```

```
Selecting Best Scores for model:  Logistic_Regression
   Drug  Accuracy  R_recall  S_recall  G  Y  S
0   CTZ  0.935837  0.662921  0.980000  1  0  0
1   CTX  0.972313  0.885246  0.993902  1  1  0
2   AMP  0.827338  0.917874  0.563380  1  1  0
3   AMX  0.870166  0.875000  0.862319  1  0  0
4   AMC  0.780180  0.576471  0.870130  1  0  0
5   TZP  0.931408  0.090909  0.984645  1  0  0
6   CXM  0.877934  0.664804  0.960870  1  1  0
7   CET  0.852518  0.817391  0.877301  1  0  0
8   GEN  0.934272  0.686869  0.979630  1  0  0
9   TBM  0.848921  0.674157  0.931217  1  0  0
10  TMP  0.870504  0.857143  0.880503  1  0  0
11  CIP  0.951487  0.842466  0.983773  1  0  0
Selecting Best Scores for model:  Gradient_Boosted_Trees
   Drug  Accuracy  R_recall  S_recall  G  Y  S
0   CTZ  0.956182  0.764045  0.987273  1  0  0
1   CTX  0.982085  0.926230  0.995935  1  0  0
2   AMP  0.913669  0.956522  0.788732  1  1  1
3   AMX  0.886740  0.892857  0.876812  1  0  0
4   AMC  0.805405  0.576471  0.906494  1  1  1
5   TZP  0.936823  0.060606  0.992322  1  1  1
6   CXM  0.887324  0.659218  0.976087  1  0  1
7   CET  0.920863  0.869565  0.957055  1  1  1
8   GEN  0.965571  0.808081  0.994444  1  0  0
9   TBM  0.928058  0.831461  0.973545  1  0  1
10  TMP  0.935252  0.941176  0.930818  1  1  1
11  CIP  0.953052  0.849315  0.983773  1  0  0
<ipython-input-10-027028a77d23>:9: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeError in the f
  S_rec = float(data[data["Drug_combo"] == drug_combo]["S_recall"])
<ipython-input-10-027028a77d23>:10: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeError in the
  R_rec = float(data[data["Drug_combo"] == drug_combo]["R_recall"])
<ipython-input-10-027028a77d23>:9: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeError in the f
  S_rec = float(data[data["Drug_combo"] == drug_combo]["S_recall"])
<ipython-input-10-027028a77d23>:10: FutureWarning: Calling float on a single element Series is deprecated and will raise a TypeError in the
  R_rec = float(data[data["Drug_combo"] == drug_combo]["R_recall"])
```

Step 6) Creating a function to plots bar charts for each metric

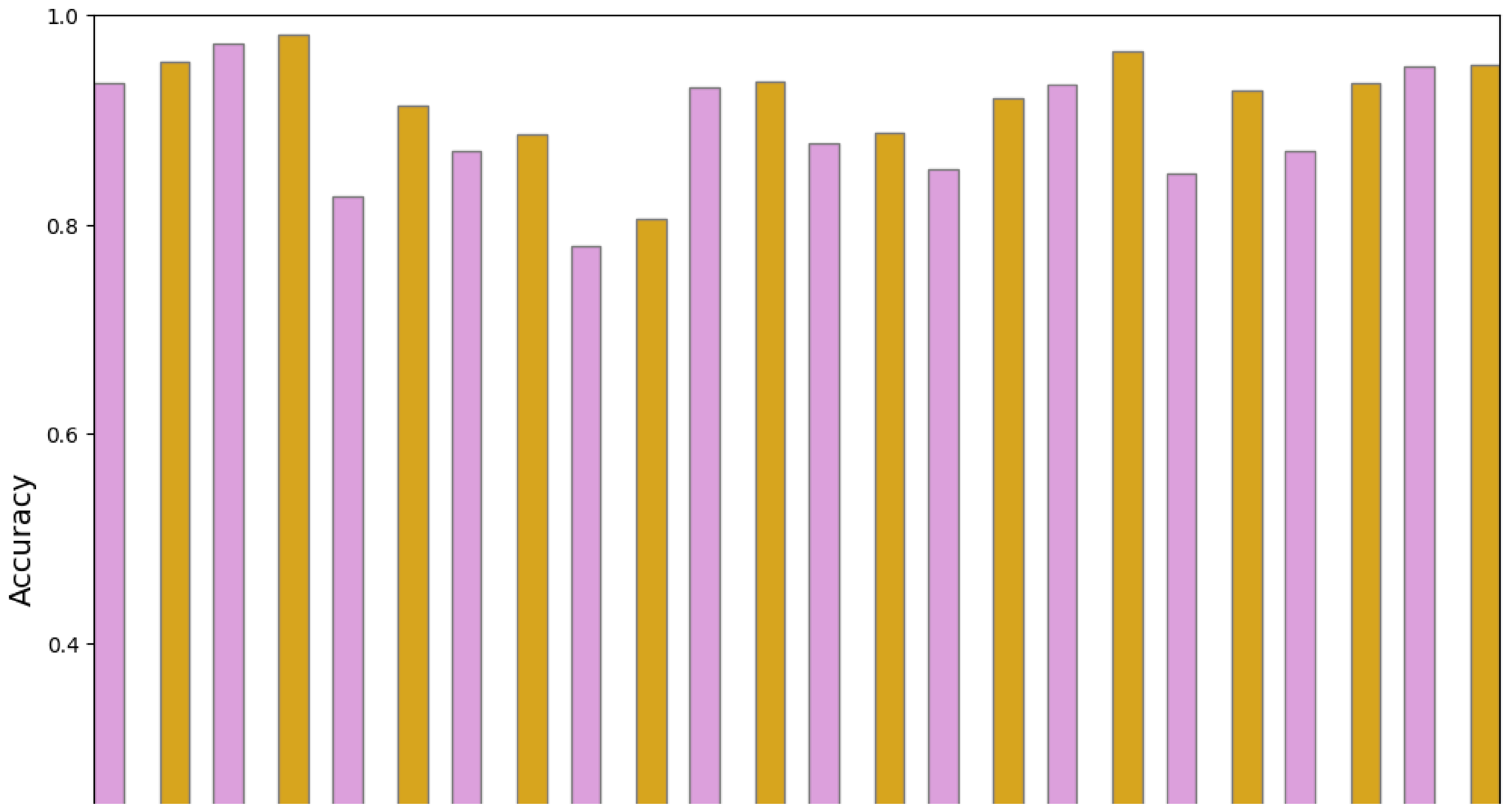
The function we create below will allow us to graph the bar charts for each of our antibiotic drugs.

```
def barplot(metric_col, subplot_axis, label_show = True):
    for model, df in Best_metrics_models.items():
        X_axis = np.arange(len(drug_list))
        X_labels = drug_list

        Y_axis = np.arange(0,1.2,0.2)

        subplot_axis.set_ylabel(metric_col, fontsize = 14)
        subplot_axis.set_ylim(bottom=0, top=1)

        subplot_axis.set_xticklabels(X_labels, fontsize = 15)
        subplot_axis.set_xticks(X_axis)
        subplot_axis.margins(x=0)

        if label_show == False:
            subplot_axis.tick_params(left = True, right = False , labelleft = True , labelbottom = False, bottom = True)

        if model == "Logistic_Regression":
            X_axis = X_axis - 0.2
            color = "plum"
            label = "LG"
            subplot_axis.bar(X_axis, list(df[metric_col]), width =.25, align = 'center', color = color, label = label, edgecolor="gr
        elif model == "Random_Forest":
            X_axis = X_axis + 0.075
            color = "cadetblue"
            label = "RF"
            subplot_axis.bar(X_axis, list(df[metric_col]), width =.25, align = 'center', color = color, label = label, edgecolor="gr
        elif model == "Gradient_Boosted_Trees":
            X_axis = X_axis + 0.35
            color = "goldenrod"
            label = "GB"
            subplot_axis.bar(X_axis, list(df[metric_col]), width =.25, align = 'center', color = color, label = label, edgecolor="gr
    return
```

Below we implement this function for the best results obtained in our Logistic Regression Model. In this case we will be plotting the "Accuracy"

```
# implementing function barplot() for one metric "Accuracy"
figure, subs = plt.subplots(1, 1,figsize = (12,9))
acc_plot = barplot("Accuracy",subs,label_show=True)
```

```
<ipython-input-15-092e439ac933>:11: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after s
  subplot_axis.set_xticklabels(X_labels, fontsize = 15)
```

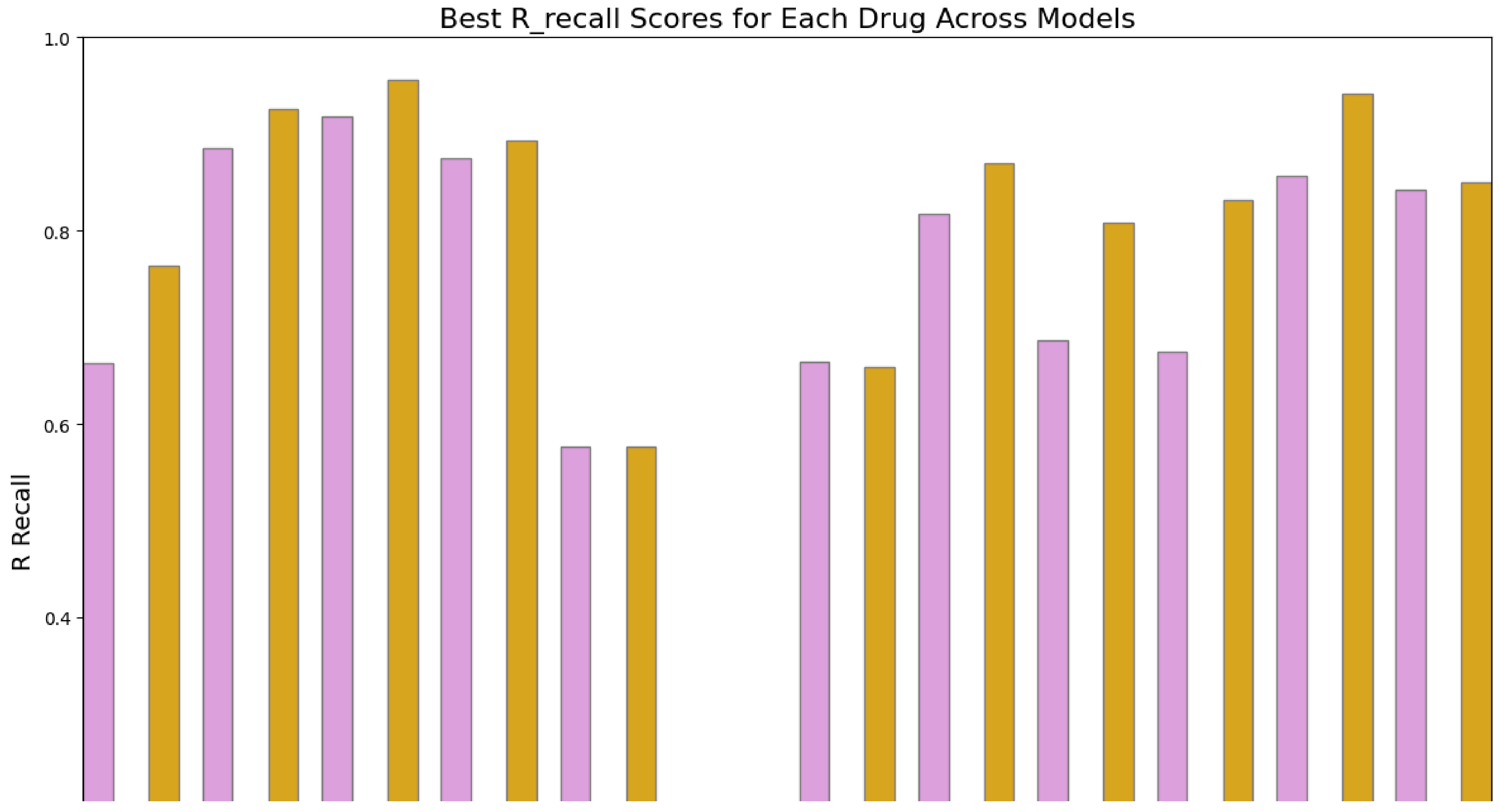


Task 1:

- Create bar charts using the function above for the best Resistance (R) recall scores.

```
# Import required libraries
import matplotlib.pyplot as plt

# Create a bar chart for the best R_recall scores
figure, axis = plt.subplots(1, 1, figsize=(12, 9))
barplot("R_recall", axis, label_show=True)

# Set title and labels for clarity
axis.set_title("Best R_recall Scores for Each Drug Across Models", fontsize=16)
axis.set_ylabel("R Recall", fontsize=14)
axis.set_xlabel("Drugs", fontsize=14)
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility

# Display the plot
plt.tight_layout()
plt.show()
```

```
<ipython-input-15-092e439ac933>:11: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after s
  subplot_axis.set_xticklabels(X_labels, fontsize = 15)
```



Step 7) Creating a function to plot GYS grids for each drug

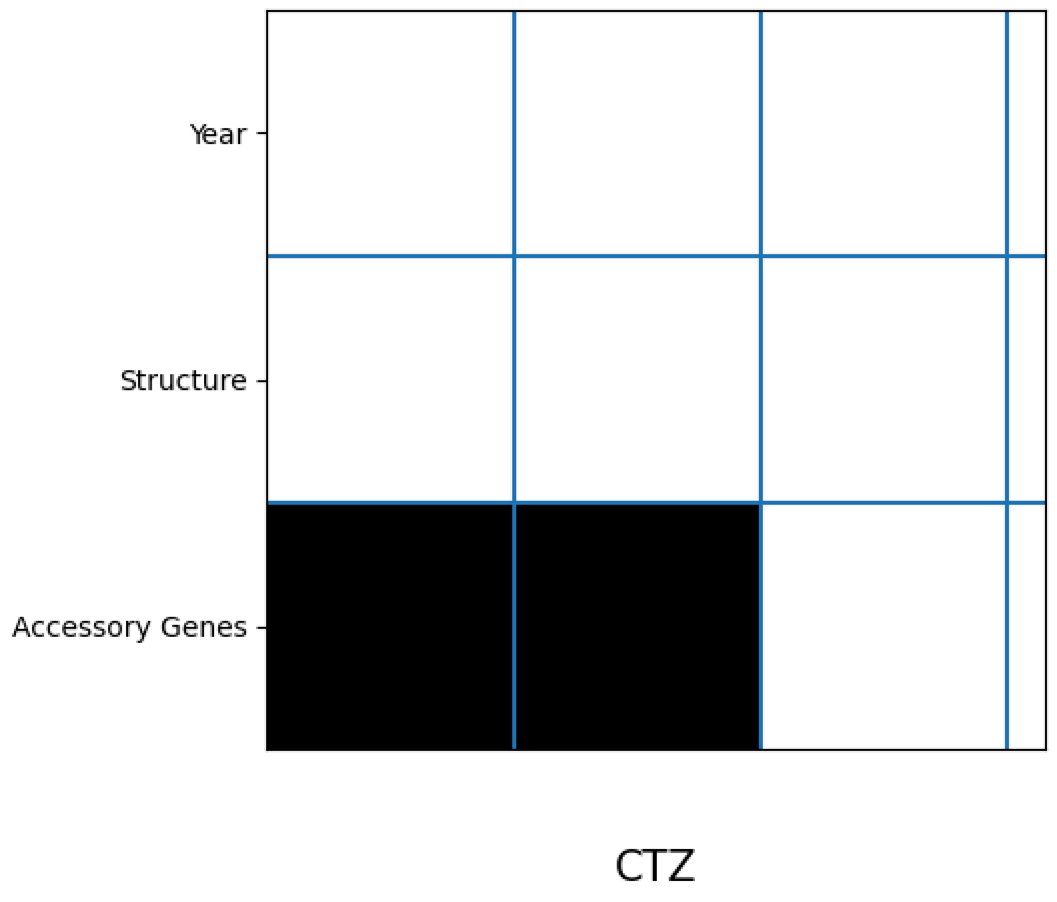
The function below will let us introduce the graphics below each drug, to show us what is the best combination of features.

```
# function that creates a graph for each of the drugs
def GYS_gridplot(drug,subplot_axis, label_show = True, title_pos = -0.2):
    # Create 3 new lists one for each type of features (Year, Population structure and Gene presence)
    Y_list = [] # storing whether it used or not year data
    S_list = [] # storing whether it used or not population structure data
    G_list = [] # storing whether it used or not accessory gene data

    # Fill up corresponding lists from drug results from each model
    for model, df in Best_metrics_models.items():
        for drug_name in df["Drug"]:
            if drug_name == drug:
                Y_list.append(int(df["Y"][df["Drug"]==drug]))
                S_list.append(int(df["S"][df["Drug"]==drug]))
                G_list.append(int(df["G"][df["Drug"]==drug]))

    Drug_GYS_list = [Y_list, S_list, G_list]

    plt.yticks(np.arange(3), ["Year", "Structure", "Accessory Genes"])
    if label_show == False:
            subplot_axis.tick_params(left = False, labelleft = False)

    orig_map = plt.cm.get_cmap('gray')
    reversed_map = orig_map.reversed()
    subplot_axis.imshow(Drug_GYS_list, cmap = reversed_map)

    subplot_axis.axvline(x=0.5)
    subplot_axis.axvline(x=1.5)
    subplot_axis.axvline(x=2.5)
    subplot_axis.axhline(y=0.5)
    subplot_axis.axhline(y=1.5)
    subplot_axis.set_title(drug, fontsize= 15, y=title_pos)
    subplot_axis.tick_params(
        axis = 'x',
        which = 'both',
        bottom = False,
        top = False,
        labelbottom = False)
    return [Y_list, S_list, G_list]
```

The example implementation below shows us the best results obtained from the drug "CTZ" for each of the 3 models. In this case the order would be: **Logistic Regression, Random Forest and Gradient-Boosted Trees**.

```
# implementing function GYS_gridplot() for one drug "CTZ"
figure, subs = plt.subplots()
CTZ_gridplot = GYS_gridplot("CTZ",subs, label_show=True)
```

```
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
```



Step 8) Create the final composite graph (Bargraphs + GYS gridplots)

Below we create our composite plot (bar charts for each measurement + gridplots) for each of the drugs.

```
# Code to create bargraphs

fig = plt.figure(figsize = (20,15), constrained_layout=False)

gs1 = fig.add_gridspec(nrows=4, ncols=12, left=0.05, right=0.6, wspace=0.07)
# Accuracy barcharts for all models
acc_axis = fig.add_subplot(gs1[0, :])
acc_axis.set_title('Prediction of Antibiotic Resistance From Pan-Genome Data', fontsize = 20)
acc_plot = barplot("Accuracy",acc_axis, label_show = False)

# R_recall barcharts for all models
R_rec_axis = fig.add_subplot(gs1[1, :])
R_rec_plot = barplot('R_recall',R_rec_axis, label_show = False)

# S_recall barcharts for all models
S_rec_axis = fig.add_subplot(gs1[2, :])
S_rec_plot = barplot('S_recall',S_rec_axis, label_show = False)

# GYS gridplot charts for each drug
gs2 = fig.add_gridspec(nrows=1, ncols=12, top=0.50,bottom=0,left=0.05, right=0.6, wspace=0.2)
i=0
while (i< len(drug_list)):
    for drug in drug_list:
        if i == 0:
            Drug_grid = fig.add_subplot(gs2[-1, i])
            drug_GYS = GYS_gridplot(drug, Drug_grid, label_show=True, title_pos=-0.5)
            i+=1
        else:
            Drug_grid = fig.add_subplot(gs2[-1, i])
            drug_GYS = GYS_gridplot(drug, Drug_grid, label_show=False, title_pos=-0.5)
            i+=1

legend_elements = [Patch(facecolor='plum', edgecolor='plum', label='LG'),
                                      Patch(facecolor='cadetblue', edgecolor='cadetblue', label='RF'),
                                      Patch(facecolor='goldenrod',  edgecolor='goldenrod',  label='GB'),
                                      Patch(facecolor='black',  edgecolor="black",  label='Used'),
                                      Patch(facecolor='white',  edgecolor="black",label='Not  Used')]

plt.legend(handles=legend_elements,loc  =  'lower  center',  bbox_to_anchor=(-6.1,  -1),  prop  ={'size':  15},  borderaxespad=-2,  ncol  =  len(
```

```
<ipython-input-15-092e439ac933>:11: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after s
  subplot_axis.set_xticklabels(X_labels, fontsize = 15)
<ipython-input-15-092e439ac933>:11: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after s
  subplot_axis.set_xticklabels(X_labels, fontsize = 15)
<ipython-input-15-092e439ac933>:11: UserWarning: set_ticklabels() should only be used with a fixed number of ticks, i.e. after s
  subplot_axis.set_xticklabels(X_labels, fontsize = 15)
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:14: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  G_list.append(int(df["G"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wil
  orig_map = plt.cm.get_cmap('gray')
<ipython-input-18-7b624cc67cd1>:12: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  Y_list.append(int(df["Y"][df["Drug"]==drug]))
<ipython-input-18-7b624cc67cd1>:13: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeErr
  S_list.append(int(df["S"][df["Drug"]==drug]))
```

Task 2:

- Look at the results from our final graph, do you think there might be ways to improve our results?
- Consider some of the following: would you use different models? Different hyperparameters? Collect more data? Use different parts of the data?

```
  G list.append(int(df["G"][df["Drug"]==drug]))
```

Task 2 answer: Yes, there are several ways to improve the results based on the final graph:

1. **Different Models:** Exploring other models such as Neural Networks or Support Vector Machines might yield better predictions for complex feature interactions.
2. **Hyperparameter Optimization:** Fine-tuning parameters like the number of estimators for Gradient-Boosted Trees and Logistic Regression's regularization could help improve performance.

3. **Data Collection:** Increasing the size of the dataset or including more diverse samples could improve the generalization and robustness of the models.
4. **Feature Selection/Engineering:** Investigating feature importance and engineering new features (e.g., interaction terms or reducing redundancy) might enhance the models' ability to capture critical patterns.

```
<ipython-input-18-7b624cc67cd1>:22: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and wi
```

🙂 Congratulations, you are done with this module! Now you know how to check if a model performs better for a specific dataset you need to work on for your projects.