Two-Day Hands-On Workshop on Machine Learning Approaches in Bacterial Genomics

23rd and 24th August 2025

WORKSHOP HANDOUTS

Coordinator -Vigneshwar Ramakrishnan

Convener - Subbiah Thamotharan

Co-convener - Suma Mohan S

Organizing Committee:

Dr. Sathish Kumar M
Vignesh K.
Annesha Chakraborty.
Srimathy R.
Akilandeswari R.
Sasikala R.
Logalakshmi T.
R. Archana.

Organized by

DBT-Bioinformatics Center

School of Chemical & Biotechnology,

SASTRA Deemed University

Thanjavur, Tamil Nadu

Two-Day Hands-on Workshop on Machine Learning Approaches in Bacterial Genomics
23rd and 24th August 2025
Program Schedule

Day 1: 23rd August 2025 **Venue: Bioinformatics Lab, JVC II Floor**		**Resource Person**
8:45 am – 9.00 am	**Registration**	**R. Archana**
9:00 am – 9.15 am	**Inauguration**	
9.15 am – 10 am	**Decoding Microbial Genomics: AI technologies for AMR and beyond**	**Dr. S. Suma Mohan**
10:00 – 10:45 am	Introduction to Google Colab and basics of Python	**R. Akilandeswari & R. Srimathy**
10.45 am – 11.00 am	Tea Break	
11:00 am – 11:30 am	**Lecture:** Introduction to Machine learning approaches	**T. Logalakshmi**
11:30 am –12:15 pm	**Hands-on:** Data Collection and Curation (BV-BRC and ABRicate) using Galaxy	**Annesha Chakraborty & R. Srimathy**
12:15 pm – 12:45 pm	**Hands-on:** Data Collection and Curation (BV-BRC and ABRicate) using Galaxy	**Annesha Chakraborty & R. Srimathy**
12:45 pm – 2:00 pm	**Lunch**	
2:00 pm – 2:45 pm	**Lecture:** Model Building, Feature selection & Evaluation metrics (Gene Selection)	**R. Sasikala**
2:45 pm – 3:45 pm	**Hands-On:** Data Preparation and Model Building and hyperparameter Optimisation (LR, Decision Tree, Random Forest, SVM and Xgboost) - for Phenotype Prediction)	**Dr. M. Sathish Kumar & Akilandeswari**
3:45 pm – 4:00 pm	Tea break	
4:00 pm – 5:00 pm	**Hands-On:** ML Models: Results Interpretation & Discussion	**R. Sasikala & R. Srimathy**

Day 2: 24th August 2025 Venue: Bioinformatics Lab, JVC II Floor		Resource Person
9:30 am – 10:00 am	**Lecture:** Introduction to Bacterial Genome Variants	**T. Logalakshmi**
10:00 am – 10:30 am	**Lecture:** Variant Calling and Annotation (snippy)	**R. Sasikala & R. Akilandeswari**
10:30 am – 10:45 am	Tea Break	
10:45 am – 11:45 am	**Hands-On:** Variant Calling and Annotation (snippy) – using Galaxy	**R. Sasikala & R. Srimathy**
11:45 am – 12:45 pm	**Hands-On:** Data Preparation and Feature Selection	**R. Srimathy & R. Akilandeswari**
12:45 pm – 2:00 pm	Lunch Break	
2:00 pm – 2:45 pm	**Hands-On:** Model Building, and Optimisation (Phenotype prediction based on SNP)	**Dr. M. Sathish Kumar & R. Akilandeswari**
2:45 pm – 3:45 pm	**Hands-On:** Model Interpretation	**Dr. M. Sathish Kumar & R. Srimathy**
3:45 pm – 4:00 pm	Tea Break	
4:00 pm – 4:30 pm	Quiz and Discussions	**Annesha Chakraborty**
4:30 pm – 5:00 pm	**Valedictory session**	

1. Basics of Python

Assignment and data type checking

```
[1] # Assignment and datatype check
    x = "Hello World"
    y = 10
    z = 3.5
    fruits = ("apple", "banana", "cherry")

    print(type(x))
    print(type(y))
    print(type(z))
    print(type(fruits))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'tuple'>
```

Assignment and manipulation of the List

```
[2] # Create a list of 5 numbers
    numbers = [10, 20, 30, 40, 50]

    # 1. Print the first and last element
    print("First:", numbers[0])
    print("Last:", numbers[-1])

    # 2. Add a new number 60
    numbers.append(60)
    print("After append:", numbers)

    # 3. Remove 30 from the list
    numbers.remove(30)
    print("After removing 30:", numbers)

    # 4. Find the sum of all numbers
    print("Sum:", sum(numbers))
```

```
First: 10
Last: 50
After append: [10, 20, 30, 40, 50, 60]
After removing 30: [10, 20, 40, 50, 60]
Sum: 180
```

Creating and manipulating Dictionary

```
[3] # Create a dictionary of student info
    student = {
        "name": "Alice",
        "age": 21,
        "city": "New York"
    }

    # 1. Print name
    print("Name:", student["name"])

    # 2. Change age to 22
    student["age"] = 22
    print("Updated student:", student)

    # 3. Add a new key: grade = "A"
    student["grade"] = "A"
    print("After adding grade:", student)

    # 4. Get all keys
    print("Keys:", student.keys())
```

```
Name: Alice
Updated student: {'name': 'Alice', 'age': 22, 'city': 'New York'}
After adding grade: {'name': 'Alice', 'age': 22, 'city': 'New York', 'grade': 'A'}
Keys: dict_keys(['name', 'age', 'city', 'grade'])
```

NumPy is a Python library for fast mathematical operations on large arrays and matrices.
It is the backbone of scientific computing in Python and is used in data analysis, machine learning and other applications

```
[4]  import numpy as np

     # Create an array
     arr = np.array([1, 2, 3, 4, 5])
     print("Array:", arr)

     # Basic operations
     print("Sum:", np.sum(arr))
     print("Mean:", np.mean(arr))
     print("Max:", np.max(arr))
     print("Min:", np.min(arr))

     # 2D array (matrix)
     matrix = np.array([[1, 2], [3, 4]])
     print("Matrix:\n", matrix)
     print("Transpose:\n", matrix.T)
```

```
Array: [1 2 3 4 5]
Sum: 15
Mean: 3.0
Max: 5
Min: 1
Matrix:
 [[1 2]
 [3 4]]
Transpose:
 [[1 3]
 [2 4]]
```

Mathematical operations performed using the NumPy library

```
[5]  # Power function (x^y)
     print("2^3 =", np.power(2, 3))

     # Square root
     print("Square root of 16 =", np.sqrt(16))

     # Exponential (e^x)
     print("e^2 =", np.exp(2))

     # Logarithm (natural log, base e)
     print("ln(10) =", np.log(10))

     # Trigonometric functions
     print("sin(90) =", np.sin(np.pi/2))
     print("cos(0) =", np.cos(0))
     print("tan(45) =", np.tan(np.pi/4))

     # Absolute value
     print("Absolute of -5 =", np.abs(-5))
```

```
2^3 = 8
Square root of 16 = 4.0
e^2 = 7.38905609893065
ln(10) = 2.302585092994046
sin(90) = 1.0
cos(0) = 1.0
tan(45) = 0.9999999999999999
Absolute of -5 = 5
```

Pandas is a Python library used for handling and analysing data in tables (data frames), similar to working with Excel but much more powerful.

Usage:

- Reading and writing data (CSV, Excel, SQL, etc.)
- Cleaning and preparing datasets
- Performing calculations (sum, mean, groupby, etc.)
- Handling missing values
- Filtering and selecting rows/columns

```python
[6] import pandas as pd

    # Create a DataFrame (like a table)
    data = {
        "Name": ["Alice", "Bob", "Charlie"],
        "Age": [21, 25, 30],
        "City": ["New York", "London", "Paris"]
    }

    df = pd.DataFrame(data)

    print("Aveage age is ",round(df["Age"].mean(),3))

    # Save to CSV
    df.to_csv("people.csv", index=False)
    print("CSV file written!")

    # Read back from CSV
    df2 = pd.read_csv("people.csv")
    print(df2)
```

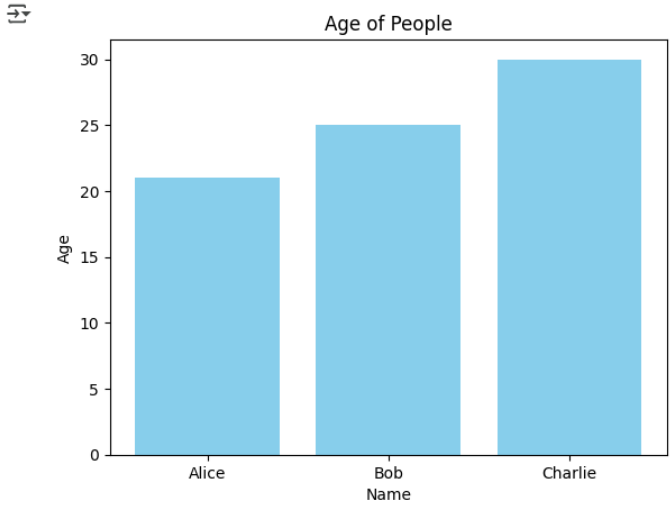
```
Aveage age is  25.333
CSV file written!
      Name  Age      City
0    Alice   21  New York
1      Bob   25    London
2  Charlie   30     Paris
```

Matplotlib is a Python library used to create visualizations like line plots, bar charts, and scatter plots for data analysis.

```python
[7] import matplotlib.pyplot as plt

    # Simple bar plot of Age vs Name
    plt.bar(df["Name"], df["Age"], color="skyblue")

    plt.title("Age of People")   # Title
    plt.xlabel("Name")           # X-axis label
    plt.ylabel("Age")            # Y-axis label
    plt.show()
```



2. Steps to Log in BV-BRC

• Go to the BV-BRC website: https://www.bv-brc.org

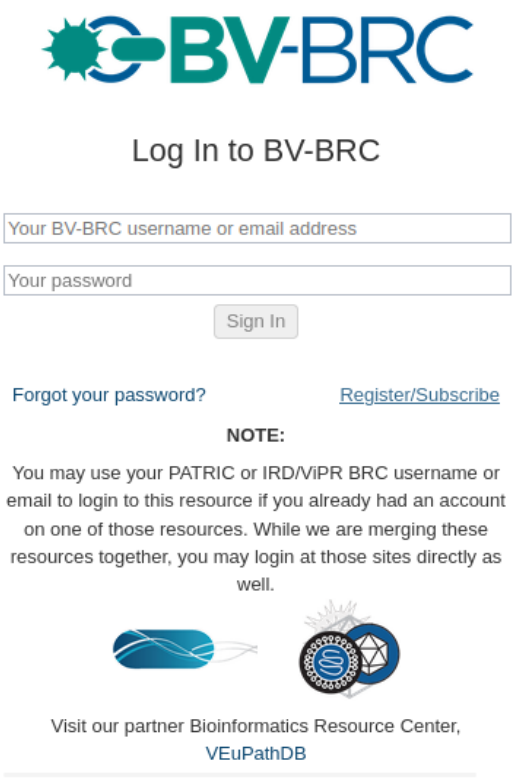
• Click on 'Login' in the top-right corner of the homepage.

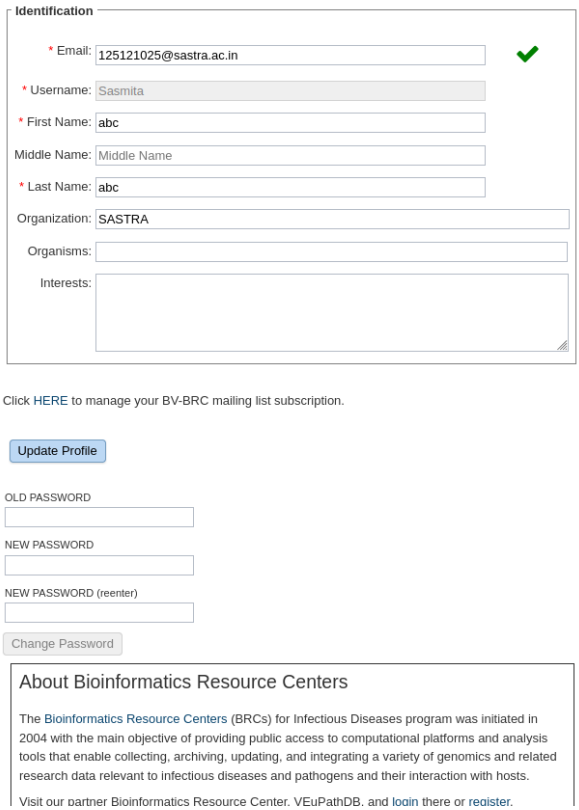
• Choose your login method:

• BV-BRC Account – Enter your email/username and password.

• Globus Authentication – If supported by your institution, log in via Globus.

• If you do not have an account, click 'Sign Up' and register using a valid email address.





3. Galaxy Login Guide

Open your web browser

• Go to the official Galaxy site: https://usegalaxy.org

• Alternatively, use a regional Galaxy server (e.g., EU, India, Australia)

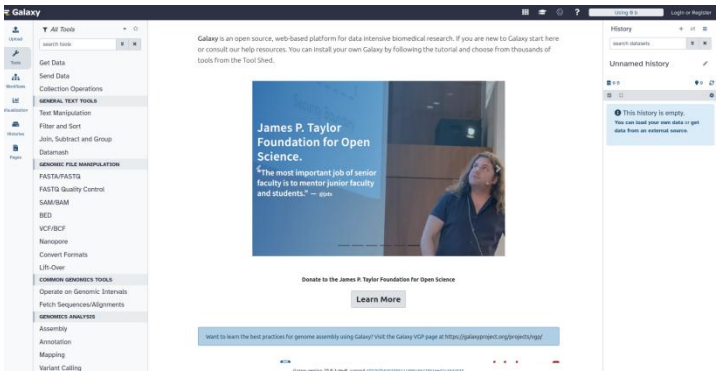
Click 'Login' at the top-right corner

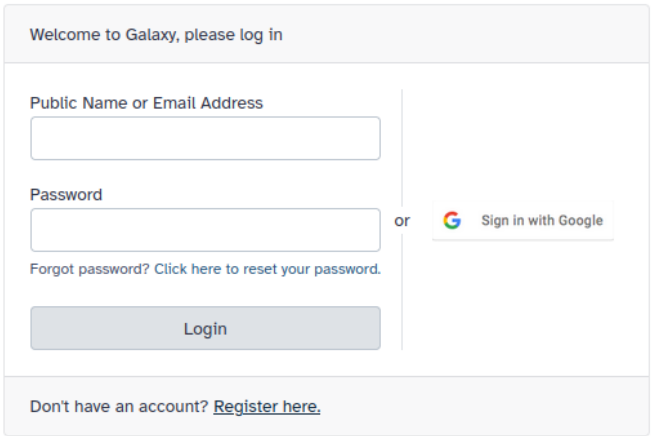
• Choose one of the following methods:

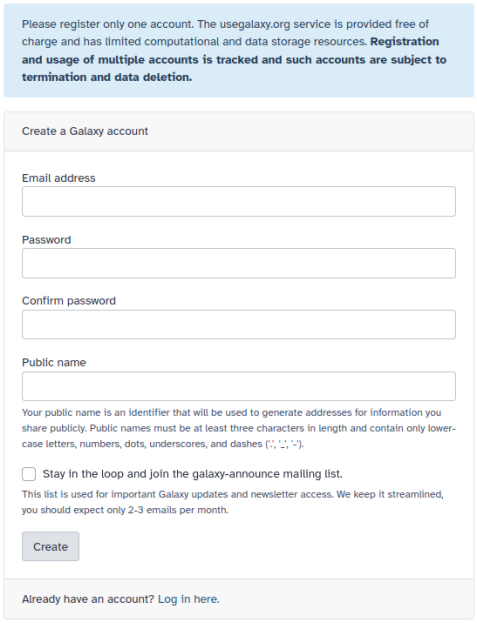
– Galaxy account (username & password)

– Institutional login (if supported)

– Social login (Google, ORCID, etc.)



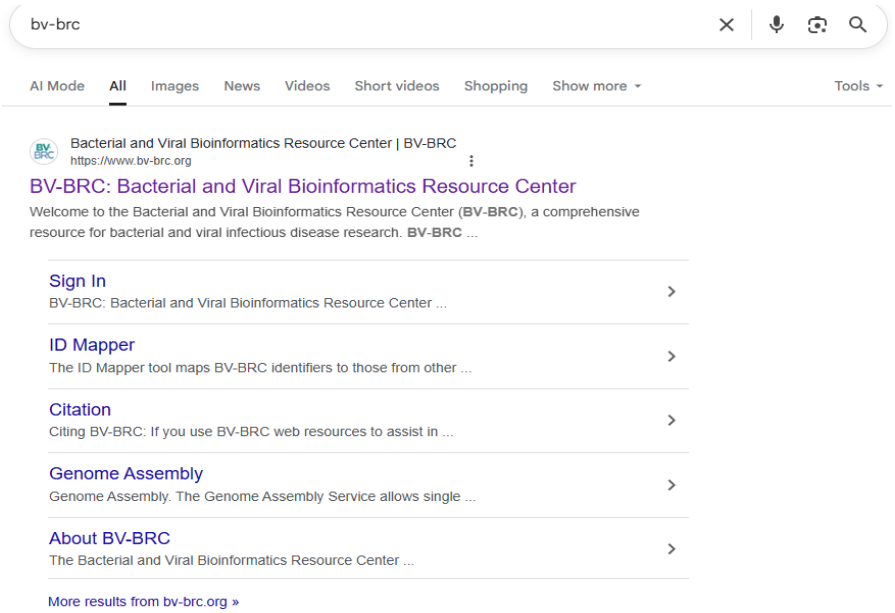




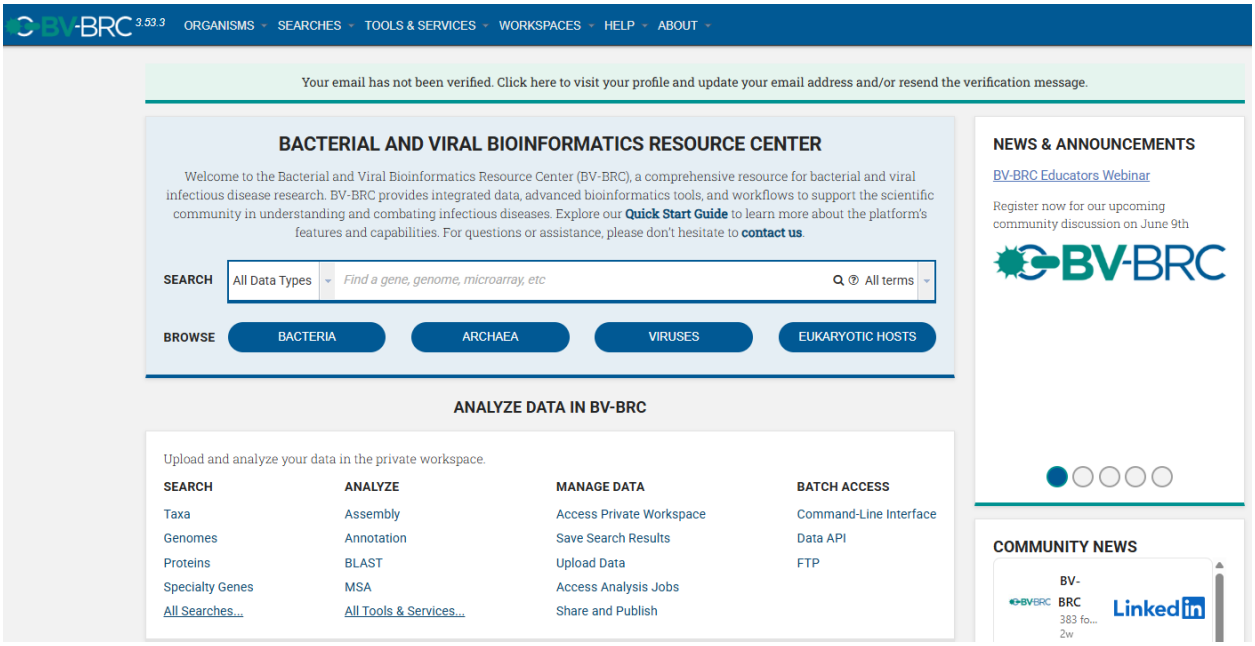
4. BV-BRC Data Collection

First go to Google and type BV-BRC over there

Then click in sign in

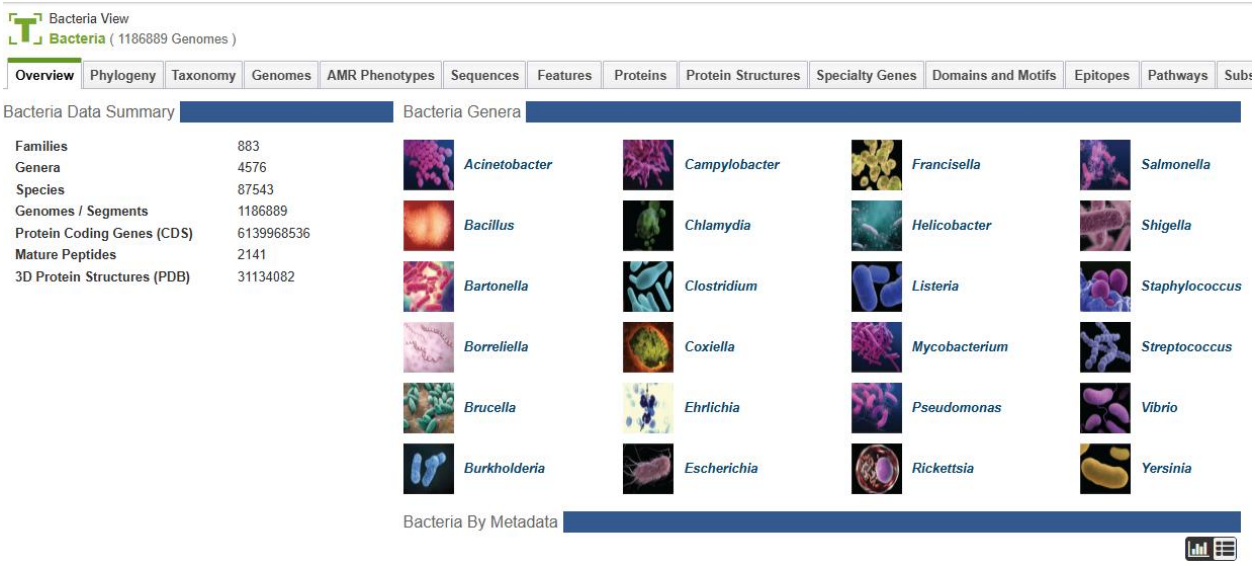


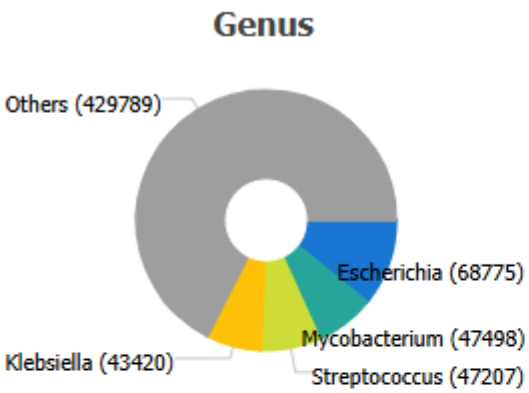
After signing in BV-BRC window will open

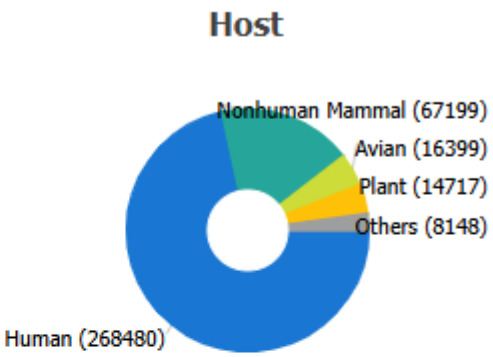


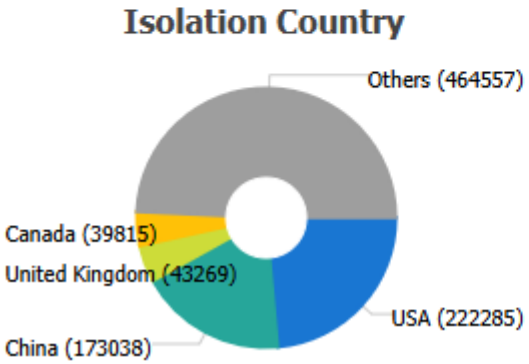
4.1 Selecting Organism and Host

Now select any organism of your interest, here we selected BV-BRC





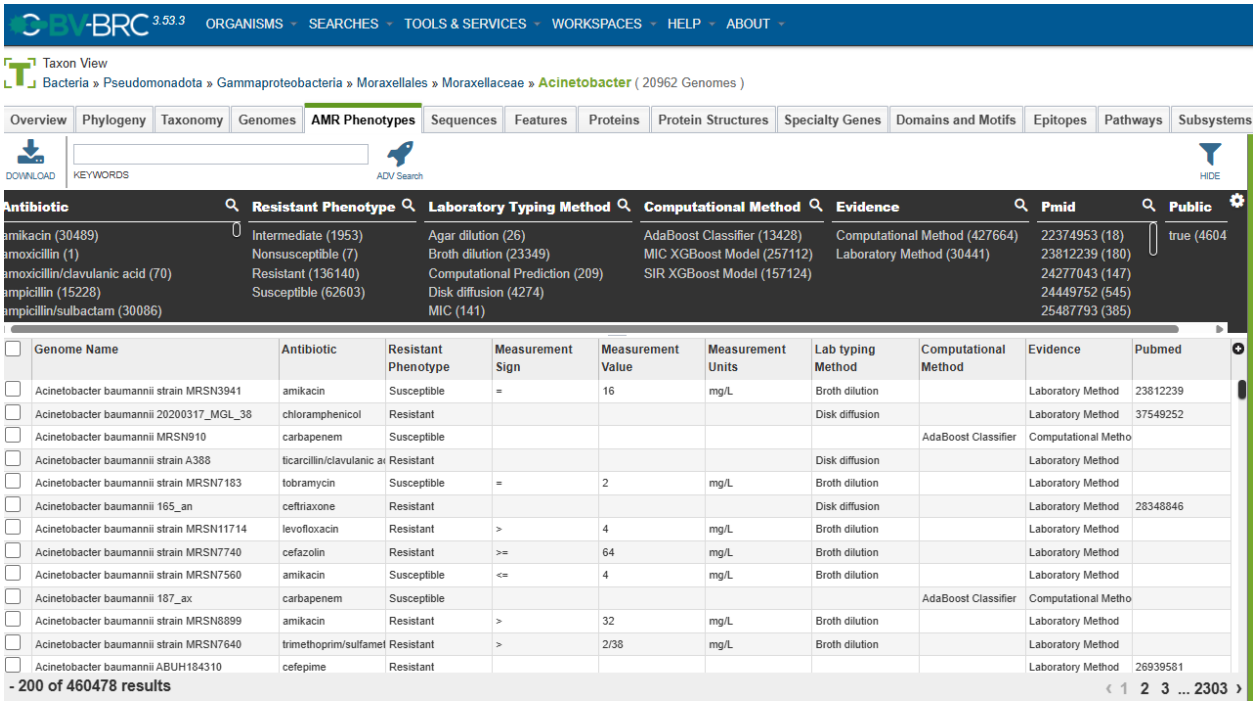


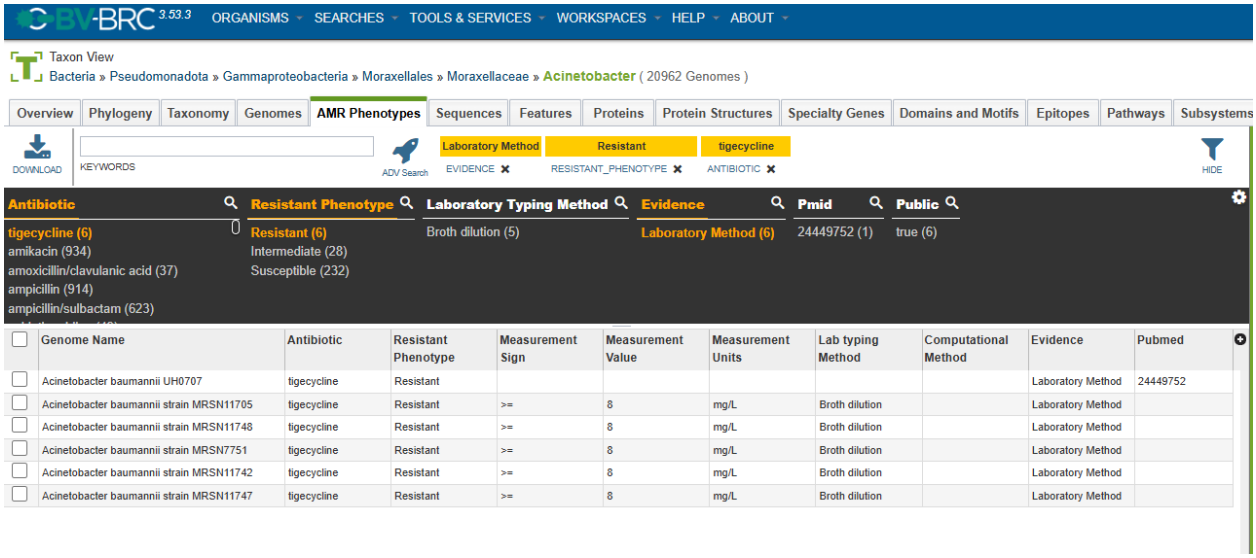


Now click to the human as the host

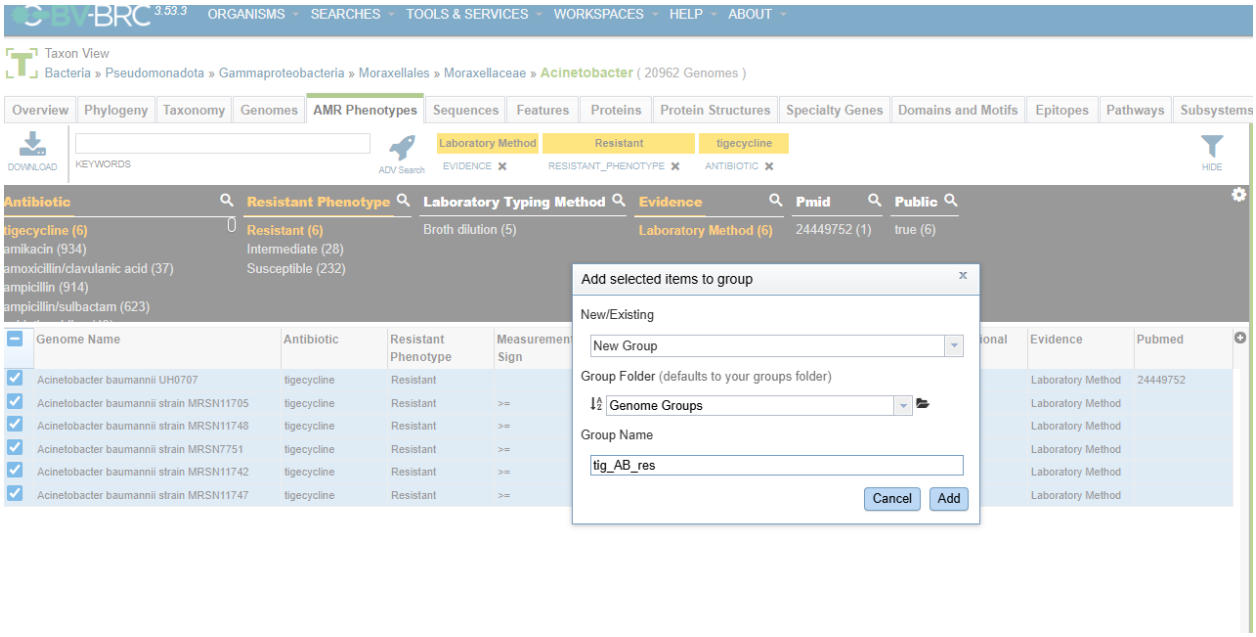
4.2 AMR Phenotype Selection

Next go to AMR phenotype, select laboratory method, resistant, tigecycline antibiotic

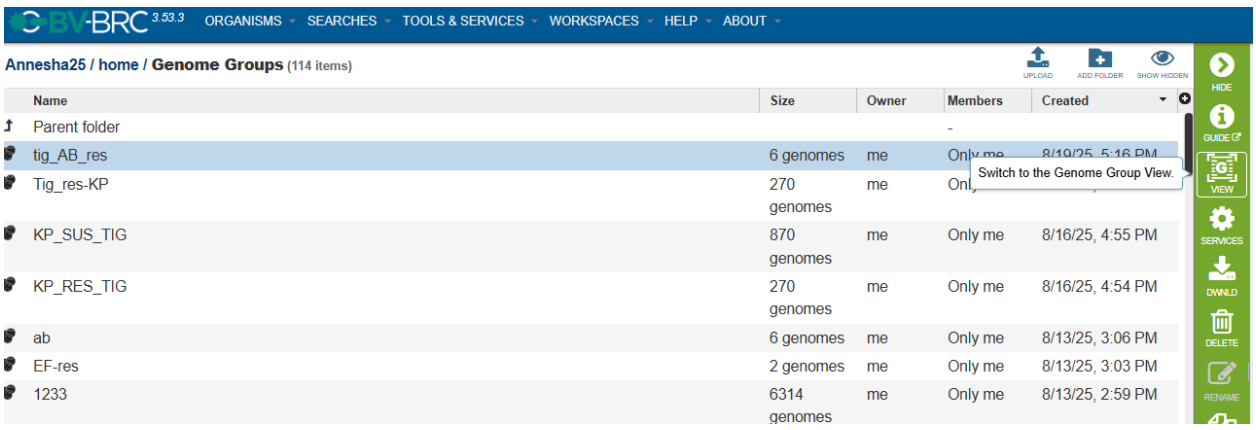




Now group it.

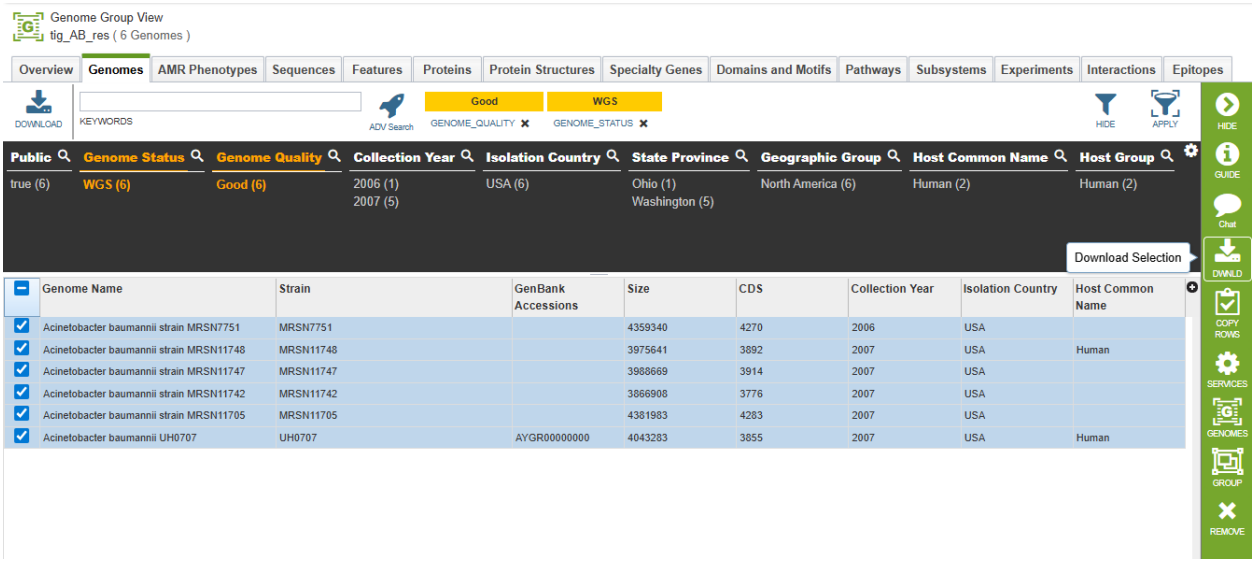


Now go to the view



4.3 Genome Download and Preparation

Go to the genome and select good quality, WGS or complete genome and finally download the txt file of that genome.



5. Antimicrobial resistance profile analysis

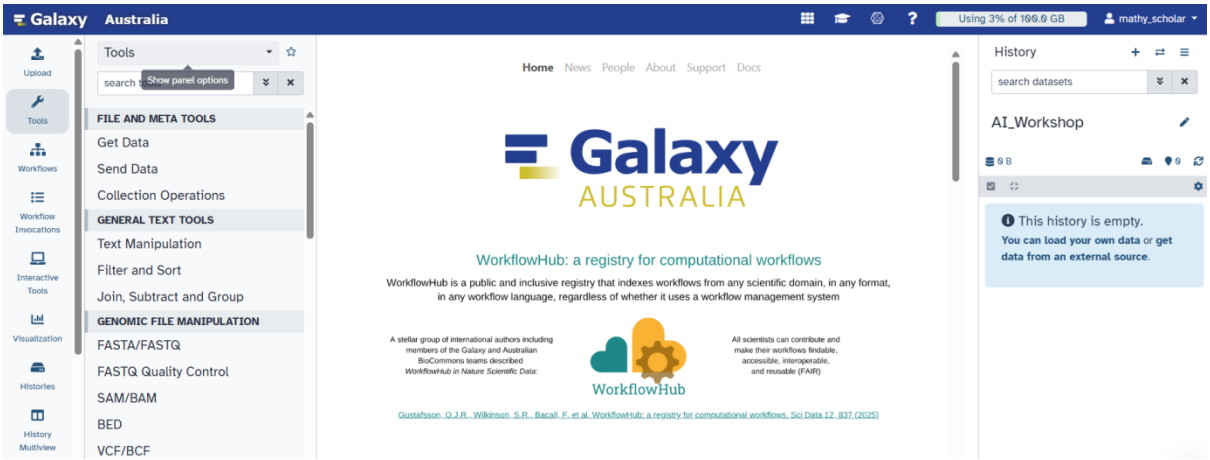
AIM:

To identify the AMR genes present in the collected isolates using ABRicate

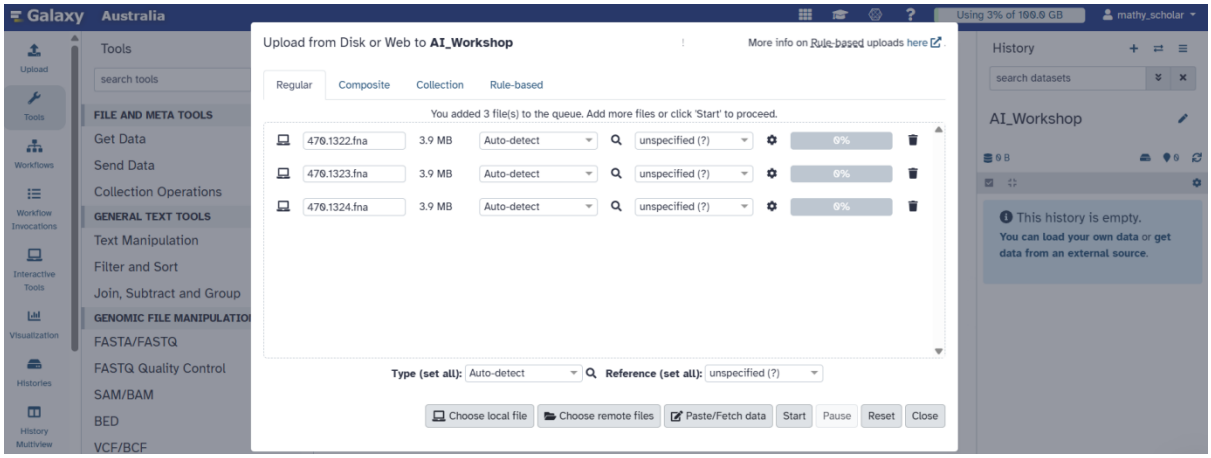
Steps to perform an ABRicate run for resistant and susceptible isolates in Galaxy

5.1 Running ABRicate in Galaxy

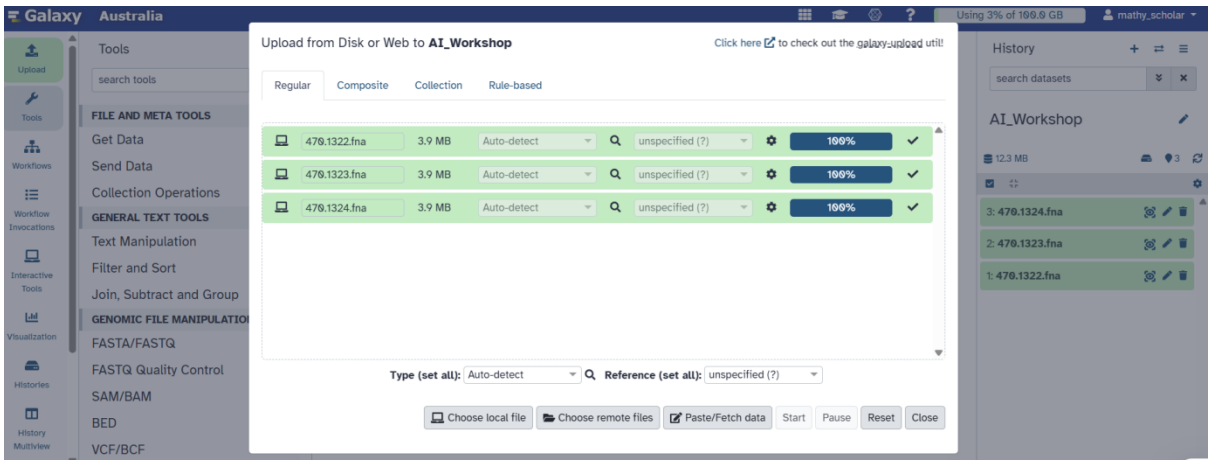
Log in to Galaxy software using the login credentials. Rename the history



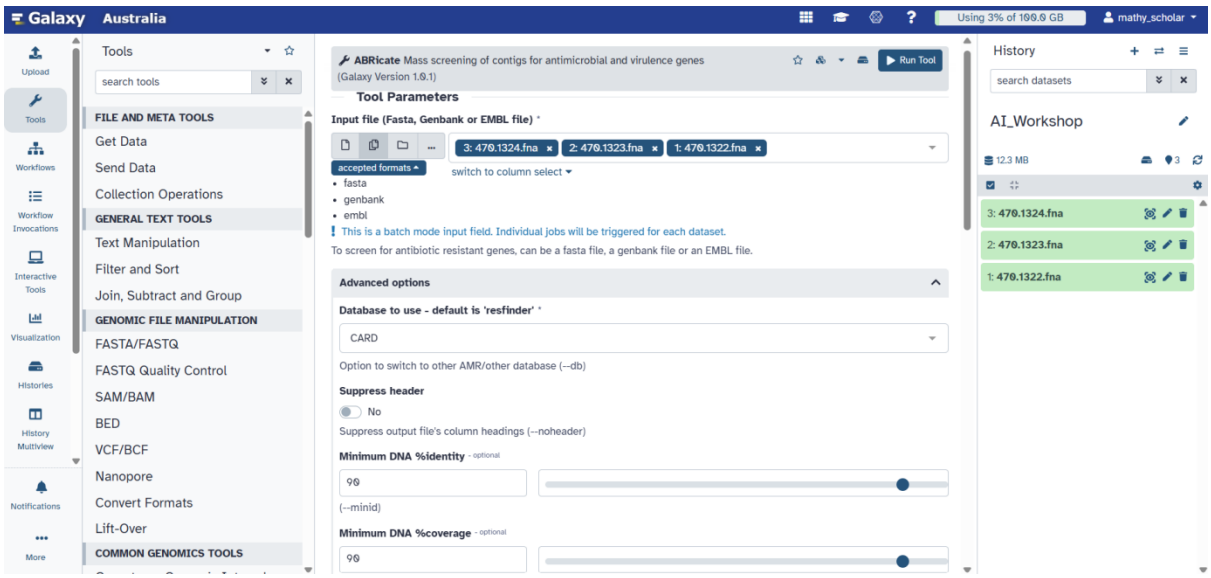
Click the upload button in the top left corner to upload the resistant/susceptible isolates. Choose the file from the local directory and click the “Start” button.



Wait for the file to upload 100% and be available in the history tab.

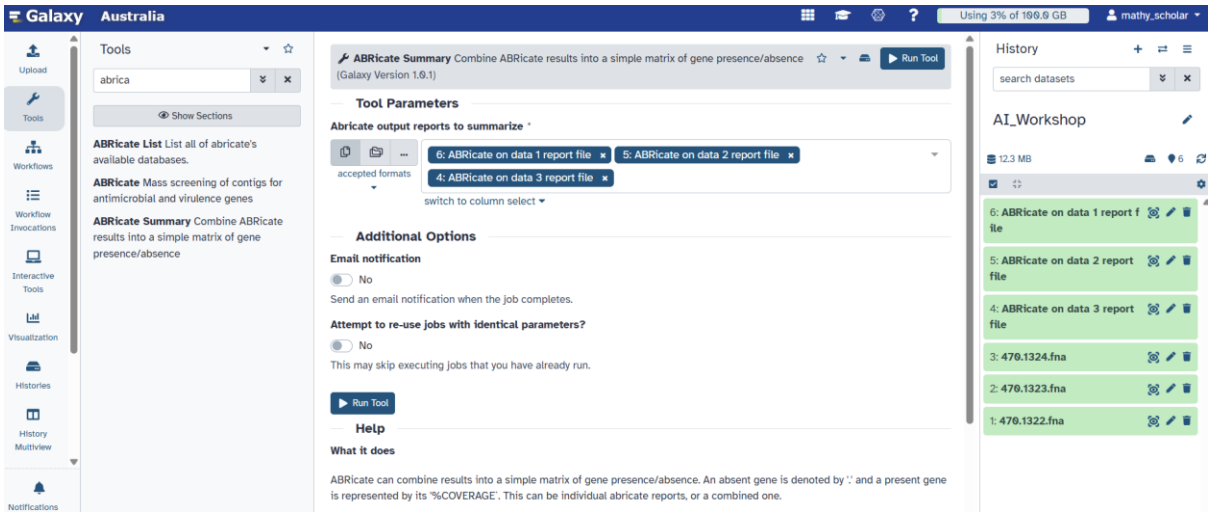


Search for abricate in the tools tab and click on **"ABRicate Mass screening of contigs for antimicrobial and virulence genes"**. Upload the fasta files in the input bar. Select the database as **"CARD"** as we are going to screen for AMR genes. And adjust the coverage and identity threshold as 90%. Click **Run.** Wait till the job gets complete.



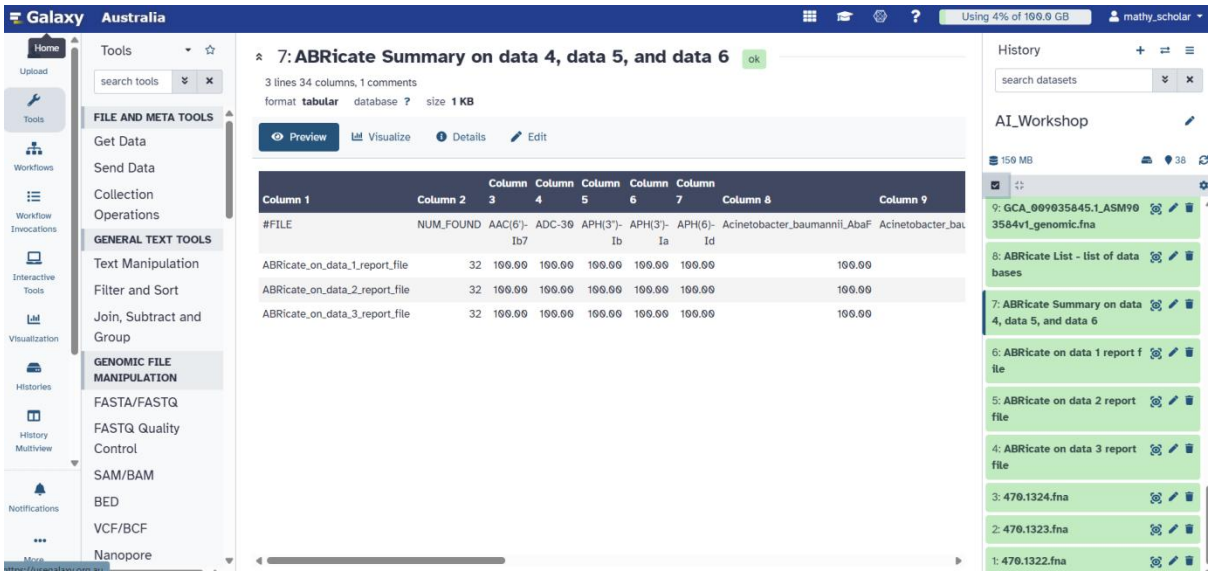
5.2 Combining ABRicate Outputs

After running the Abricate for each resistant and susceptible file we need to combine all the abricate output to get a summary file for further analysis. In tools search for **"abricate summary"** and click on it. Upload all the abricate output files in the summary input bar and click **Run.**



5.3 Summary File Interpretation

View the summary file and download it to the local directory. The summary file provides a consolidated report of AMR gene detection across various isolates. For each isolate, the file indicates the presence of a specific AMR gene by populating the cell with the gene's coverage value. If a gene is not found in an isolate, the cell will contain a '.' instead.



Refer to the GitHub link of ABRicate to perform analysis with a larger number of isolates

https://github.com/tseemann/abricate

6. Snippy Analysis

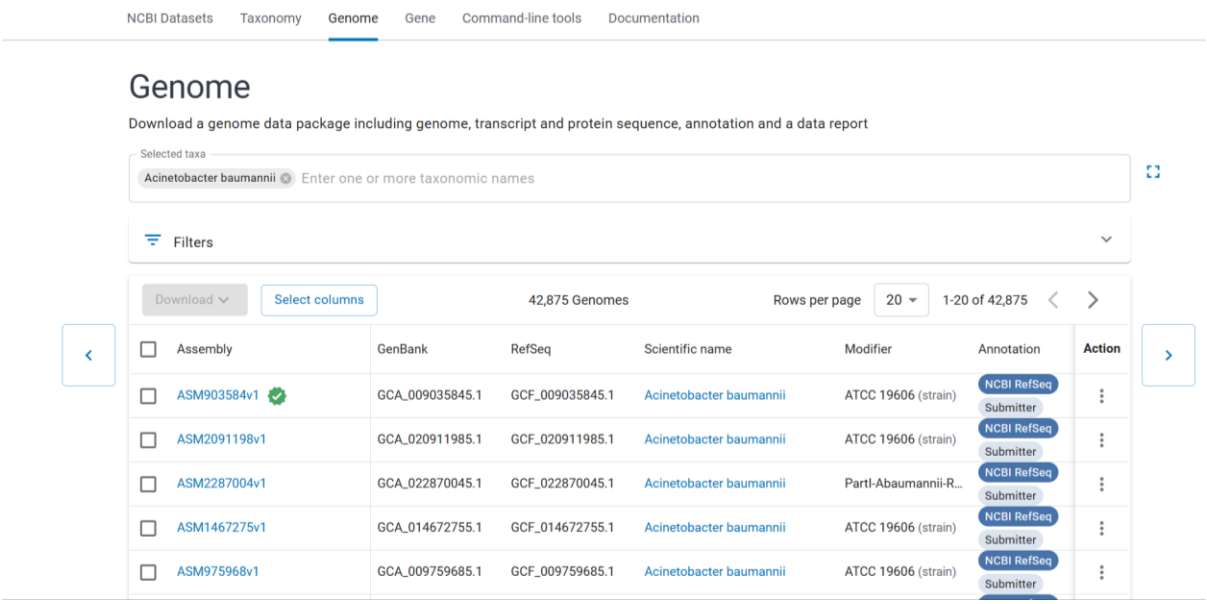
AIM:

To identify the SNP variants in the collected genomes using snippy from Galaxy and annotate them using the reference genome

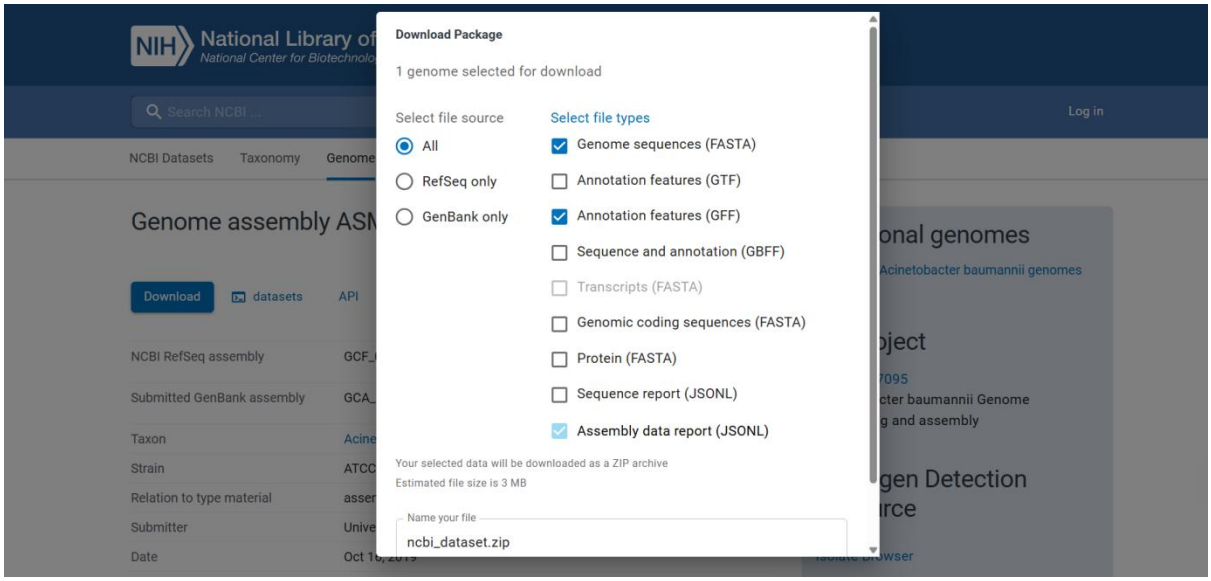
6.1 Download Reference Genome

Using the reference genome, we now identify the mutations that are present in the collected isolates using snippy tool from Galaxy

Download the fasta and gff from the NCBI genome dataset database for the bacteria species that is being analysed https://www.ncbi.nlm.nih.gov/datasets/genome/

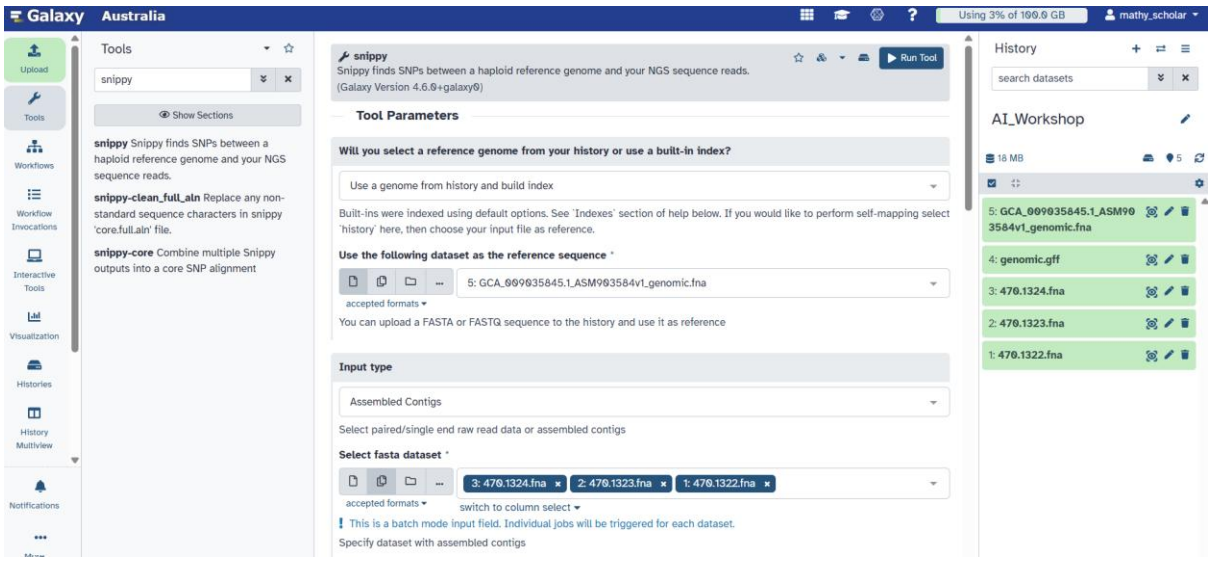


Click on RefSeq (green tick) make an assembly file, select the checkbox as fasta and gff and click download.

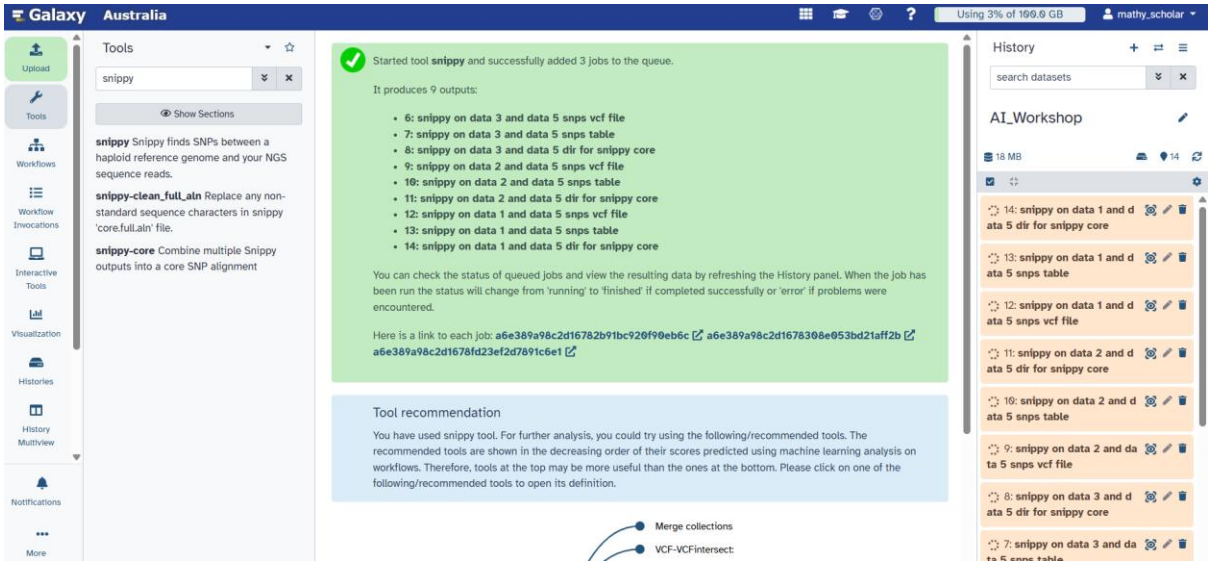


6.2 Running Snippy in Galaxy

Upload the fasta file and the gff file to the Galaxy portal. Search for the snippy in the tools bar. Click on snippy, change the reference genome option to "Use a genome from history and build index". Select the genome file(.fna file) and select the fasta file of the resistant and susceptible isolates.



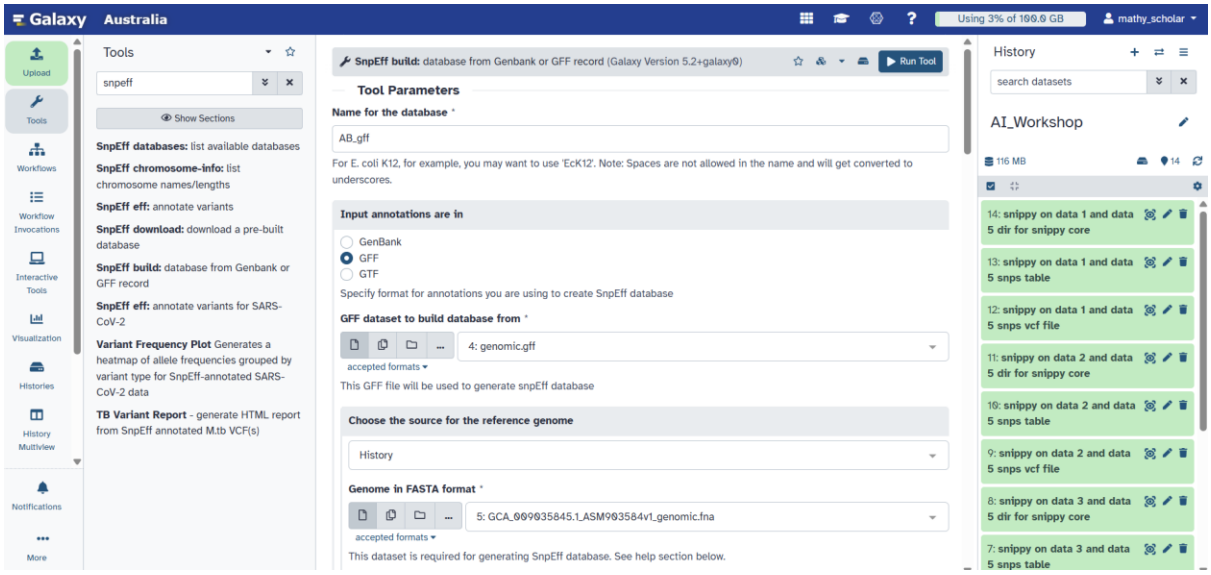
Click Run and wait until the jobs are complete.



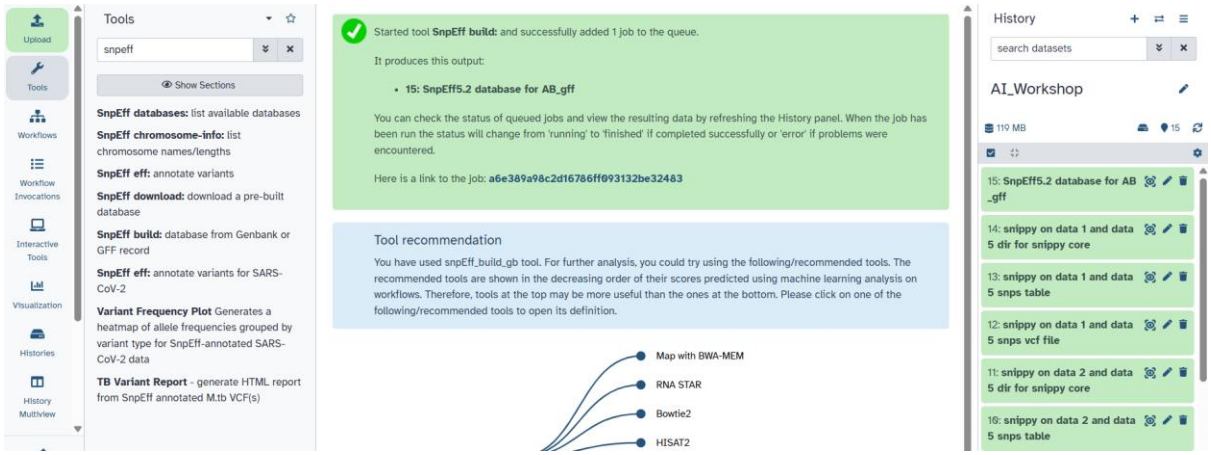
Follow the names carefully as they are automatically generated.

6.3 Building Reference Database with SnpEff

Now to annotate the results of snippy, we need to build the reference database. Search for snpeff in the tools bar and select "Snpeff build". Specify the name of the database based on the reference genome for reference. In the input annotation file, click on the GFF radio button and upload the .gff file that was downloaded from the NCBI. And chose the reference genome fasta file from the history.

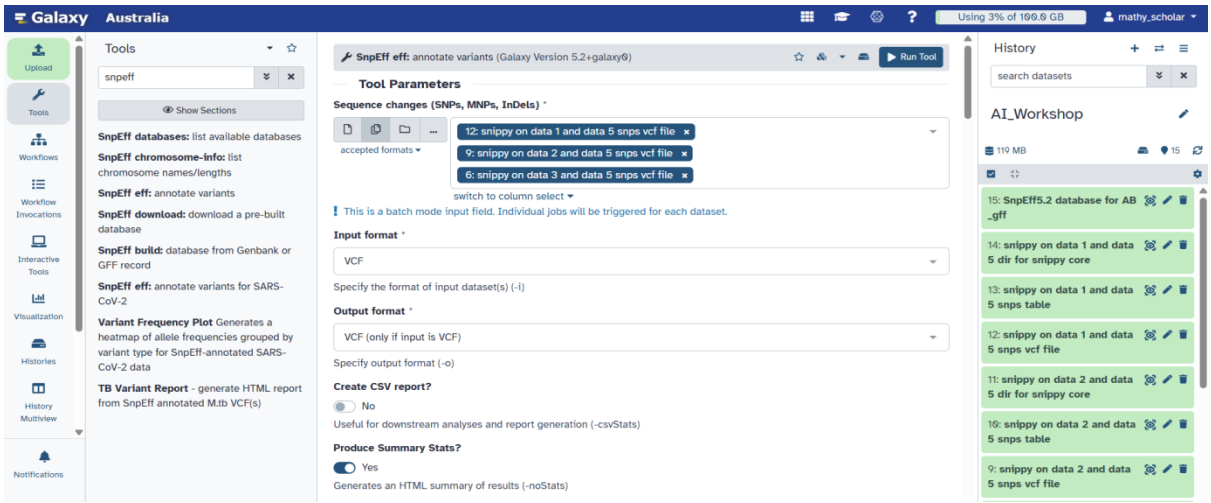


Wait until the database generation is complete

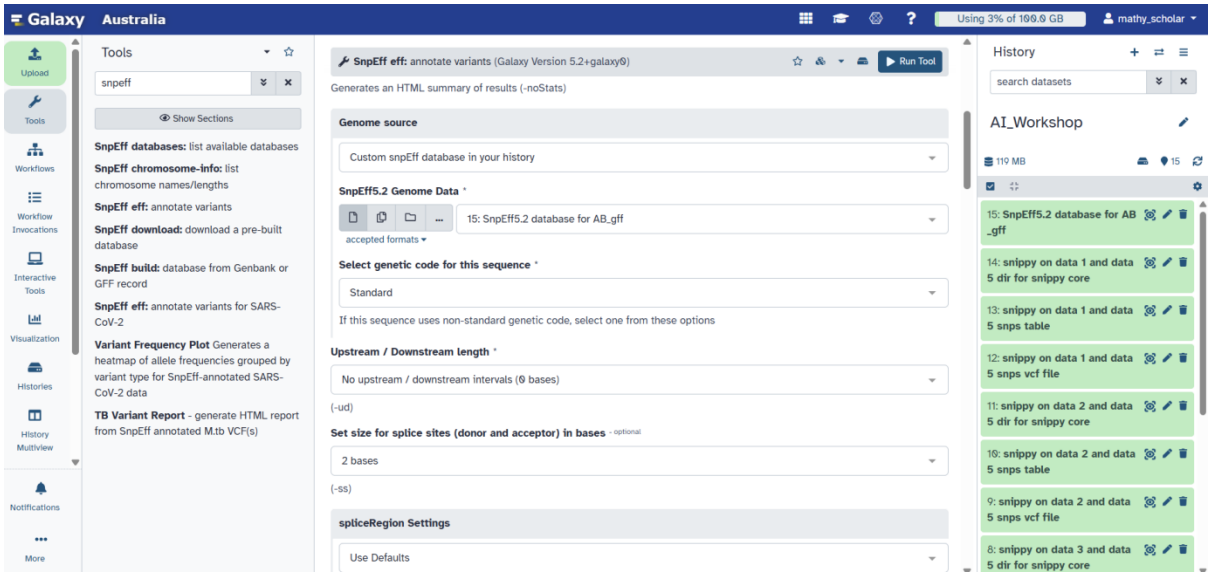


6.4 Variant Annotation with SnpEff

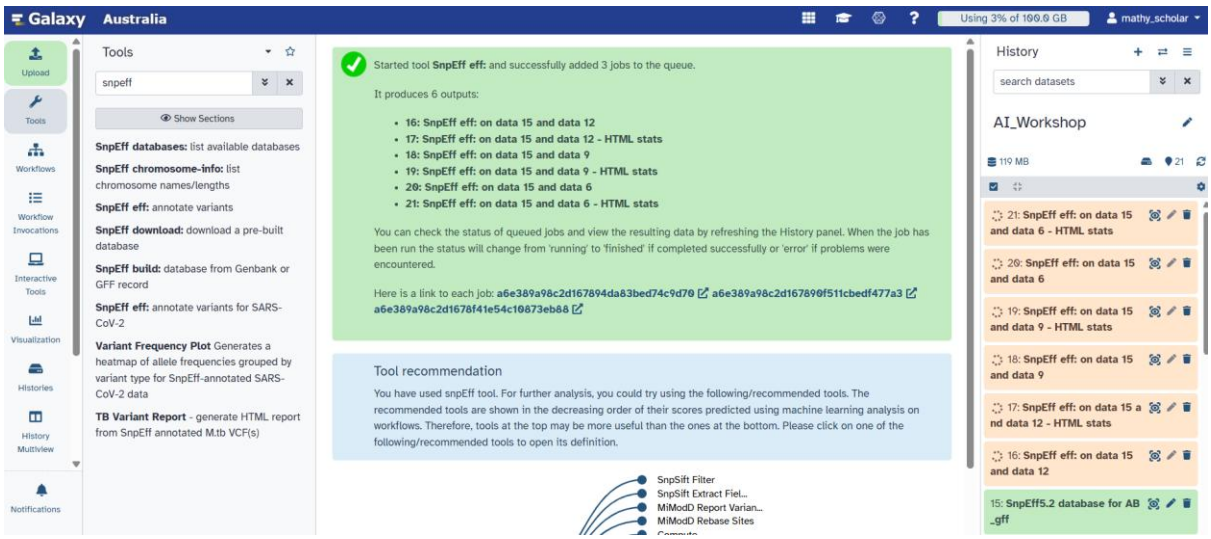
To annotate the output of the snippy file that got generated for the resistant and susceptible isolates, we are using the "SnpEff eff: annotate variants" tool from Galaxy. Search in the toolbar and select the "SnpEff eff". Upload all the Snippy output files to the input bar. The input and output format is VCF.

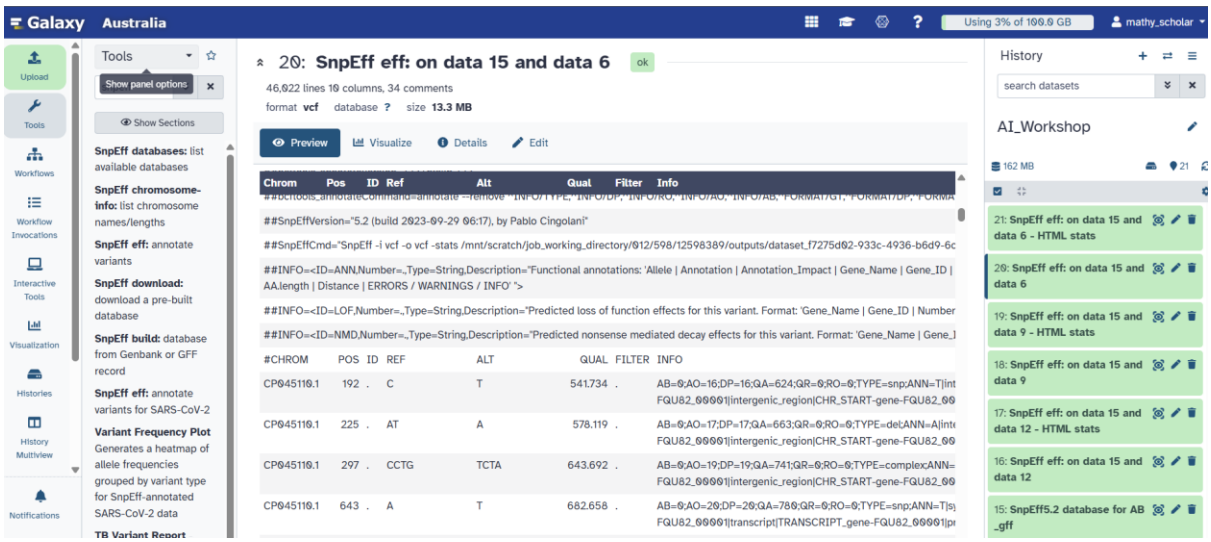


Select the created database in the SnpEff genome data and click on Run with the other default options.



Wait until the jobs are complete. And download the annotated file in the local directory





Use the anaconda package of snippy to install and run for a large number of isolates

7. Machine Learning Models

Refer to the GitHub link below for the code

bic-sastra/Machine-Learning-Workshop-on-Bacterial-Genomes: Predicts the tigecycline resistant and susceptible phenotype for Acinetobacter baumannii isolates collected from public databases using AMR genes and missense mutations

7.1 Logistic Regression

Logistic Regression is a statistical method used for **classification**, not for traditional regression. It is a supervised machine learning algorithm that is designed to predict a **categorical dependent variable**

Input Datatypes: Continuous, Discrete, Categorical (nominal-New York, Canada, ordinal – Very Good, Good, Bad, Worst)

Output Datatypes: Binary, Ordinal, Multinomial

Formula for Logistic Regression

Logistic regression works by first creating a linear equation and then applying a special function to it to "squeeze" the result into a probability.

1. **The Linear Equation:** This part is just like linear regression, where you calculate a "score" based on your features and their weights.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

   - z is the linear score.
   - β0 is the intercept.
   - β1,β2,...βn are the coefficients (or weights) assigned to each feature.
   - x1,x2,...xn are your feature values.

2. **The Sigmoid Function:** This function takes the linear score (z) and maps it to a probability (P) between 0 and 1. This is the **logit function**

$$p = \frac{1}{1 + e^{-z}}$$

By substituting the linear equation into this formula, you get the full logistic regression model:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}}$$

Assumptions of Logistic Regression

Linearity of the Log-Odds: The most important assumption is that the relationship between the independent variables and the **log-odds** (not the probability) is linear.

No Multicollinearity: The independent variables should not be highly correlated with each other.

Independent Observations: The data points should be independent of each other.

Advantages

Simple and easily scalable for multiple classes.

It's a foundational algorithm that is well-understood and computationally efficient.

Disadvantages

The model constructs linear decision boundaries, which means it may not perform well on complex, non-linear problems.

Raw coefficient values of the features are not directly interpretable as probabilities or percentages, requiring an extra step (exponentiation to get the odds ratio) to make their contribution to the prediction meaningful

7.2 Decision Tree

Decision Tree is a popular and effective supervised machine learning technique used for classification and regression problems that works with both categorical and quantitative variables. It's a graphical representation of all possible solutions to a decision based on certain conditions. The algorithm works by splitting training data points into two or more sets based on split conditions over input variables

How it works

The core idea behind decision trees is to recursively partition the data by asking a series of questions. The tree structure includes:

- **Root Node:** The top-most node of the tree, representing the entire dataset. It does not have any incoming branches
- **Decision Nodes:** These are internal nodes where a conditional expression is used on an input feature to split the data.
- **Leaf Nodes (or Terminal Nodes):** These are the final nodes that don't split further, providing the final prediction.

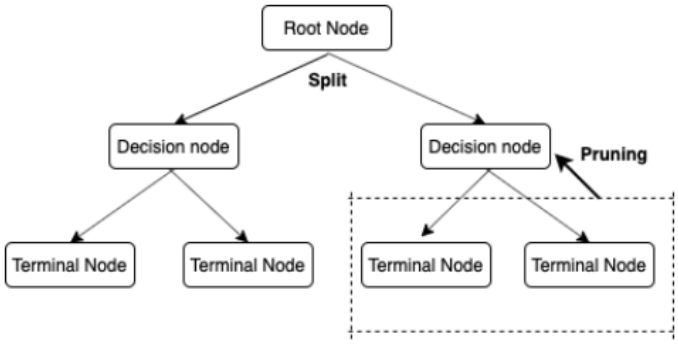
The process of dividing a node is called **splitting** based on a condition on a feature. The algorithm chooses the best feature to split on at each decision node to increase the **homogeneity** or **purity** of the resulting subsets with respect to the target variable. This is done by testing splits on all available variables and selecting the one that produces the purest sub-nodes.

Common measures of impurity are:

- **Gini Impurity:** Measures how often a randomly chosen element from a set would be incorrectly labelled. A lower Gini impurity indicates higher homogeneity.
- **Entropy:** A measure of disorder or impurity in the data.

- **Information Gain:** Measures the reduction in entropy after a dataset is split. The goal is to maximize information gain.

Pruning: A technique to reduce the size of a decision tree by removing branches that are not critical for classifying instances. This helps to prevent **overfitting**, where the model learns the training data too well and performs poorly on new, unseen data.



Advantages

It makes the process computationally efficient and relatively fast

Decision trees don't require necessary normalization or scaling, and some versions can handle missing values directly

It is very easy to explain decision tree to anyone. It does not require much knowledge of statistics and visualization also helps very much.

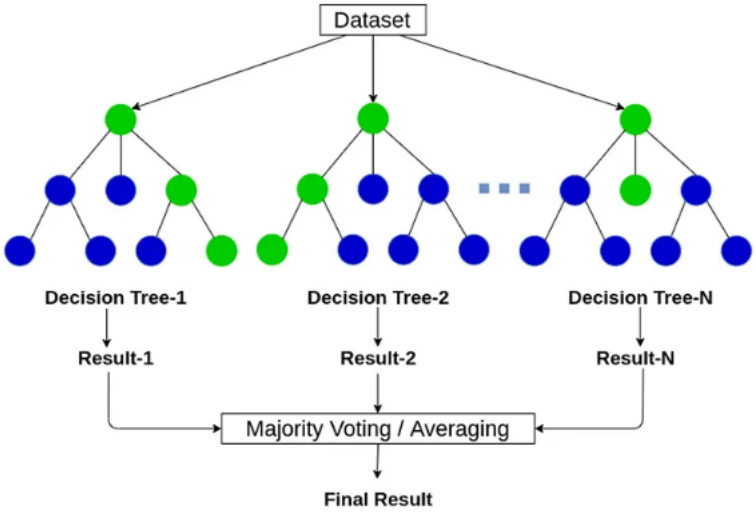
Disadvantages

It doesn't guarantee a globally optimal tree, which can lead to a less accurate model than other methods that search a wider range of possibilities

The assumption that features are independent be a disadvantage if the features in your dataset are highly correlated, which might cause the tree to make suboptimal splits

7.3 Random Forest

A Random Forest is an ensemble machine learning algorithm that builds a "forest" of individual decision trees and then combines their outputs to make a final prediction.



How it works

A Random Forest works by creating a large number of decision trees during the training phase. Unlike a single decision tree, which uses the entire dataset to find the best split at each node, a Random Forest introduces two key forms of randomness:

1. **Bootstrap Aggregating (Bagging):** Each tree in the forest is trained on a different random subset of the original data, with replacement. This means some data points might be included multiple times, while others are left out.
2. **Feature Randomness:** When a tree is being built, instead of considering all features for the best split, it only considers a random subset of features. This forces the algorithm to explore a wider range of possibilities.

For **classification problems**, the final prediction is made by a **majority vote** among all the trees. For **regression problems**, it's the **average** of the predictions from all the individual trees.

Purity Measures

Random Forest uses the same measures to check for node purity as a single decision tree:

- **Gini Impurity**
- **Entropy**

The algorithm calculates these values at each potential split to determine the best feature for partitioning the data in each individual tree

Advantages

- **Reduces Overfitting:** Significantly reduces the risk of overfitting by averaging the predictions that are common in single decision trees.
- **Increased Stability and Accuracy:** It is more robust and less sensitive to small changes in the training data than a single decision tree. The combined wisdom of many trees leads to more stable and accurate predictions.
- **Handles High-Dimensional Data:** Random Forest can handle datasets with many features

- **Works with Missing Data:** The algorithm is generally robust to missing values and can handle both categorical and numerical data

Disadvantages

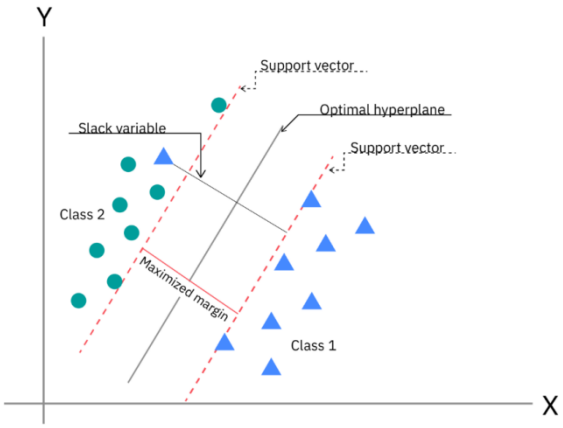
- **Interpretability:** Unlike a single, easy-to-visualize decision tree, a Random Forest is a "black box" model. It's much harder to interpret and understand the reasoning behind its final predictions.
- **Computational Cost:** Training a large number of trees can be computationally expensive and time-consuming, especially with a large dataset.
- **Slower Prediction:** Making a prediction with a Random Forest requires running a new data point through every single tree in the forest, which can be slower than a single tree

Both decision trees and random forests are versatile algorithms that can handle different data types for their variables.

Independent/Dependent features can be of both categorical or numerical types, depending on the problem statement

7.4 Support Vector Machine

A **Support Vector Machine (SVM)** is a supervised machine learning algorithm used for classification and regression tasks. Its primary objective is to find the optimal hyperplane that separates different classes of data points in a high-dimensional space. The "optimal" hyperplane is the one with the largest possible margin between the classes.



How It Works

The core idea of an SVM is to find the best decision boundary, or **hyperplane**, that separates data points of one class from another. A hyperplane is a line in 2D space, a plane in 3D, and a higher-dimensional equivalent in spaces with more features.

1. **Linear Separability:** For a dataset that can be perfectly divided by a straight line or hyperplane, the SVM's goal is to find the one that has the maximum margin. The **margin** is the distance between the hyperplane and the closest data points from each class. These closest data points are called **support vectors**, and they are the only points that influence the position of the hyperplane.

2. **Non-Linear Separability:** When data points cannot be separated by a straight line, SVMs use a technique called the **kernel trick**. This method maps the data into a higher-dimensional space where it becomes linearly separable. The algorithm then finds a linear hyperplane in this new, higher-dimensional space. When this hyperplane is projected back into the original feature space, it creates a non-linear decision boundary. Common kernel functions include polynomial, radial basis function (RBF), and sigmoid.

Advantages

- **Effective in high-dimensional spaces:** SVMs perform well when the number of features is greater than the number of samples. This makes them suitable for tasks like text classification and bioinformatics.
- **Robust against overfitting:** Due to the margin maximization principle and the use of a regularization parameter, SVMs are less prone to overfitting, especially in high-dimensional spaces.
- **Memory efficient:** The algorithm uses only a subset of the training data (the support vectors) during the prediction phase, which makes it memory-efficient.
- **Versatile:** SVMs can be used for both classification and regression problems and can handle both linear and non-linear data through the use of different kernel functions.

Disadvantages

- **Computationally expensive:** Training SVMs on large datasets can be computationally intensive, as the time complexity is typically between O(n2) and O(n3), where n is the number of samples.
- **Sensitive to parameter selection:** The performance of an SVM heavily depends on the choice of parameters, such as the regularization parameter (C) and the kernel parameters. Selecting optimal values can be challenging.
- **Poor performance with noisy data:** If the dataset is very noisy or classes are overlapping, the SVM's performance can suffer significantly, as the support vectors may be mislabeled points, leading to a poor decision boundary.
- **Lack of transparency:** For non-linear SVMs using a kernel trick, it can be difficult to interpret the model and understand the relationship between the features and the outcome.

7.5 Gradient Boosting

Gradient Boosting is a machine learning technique that builds a predictive model as an ensemble of simple, weak models, typically decision trees. It works by training models sequentially, where each new model is built to correct the errors (residuals) of the models that came before it. The process is guided by a **gradient descent** algorithm, which helps minimize the loss function and improve the model's accuracy

7.6 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is a highly optimized and enhanced version of traditional gradient boosting. The "extreme" part comes from a series of improvements that make it faster, more efficient, and more robust.

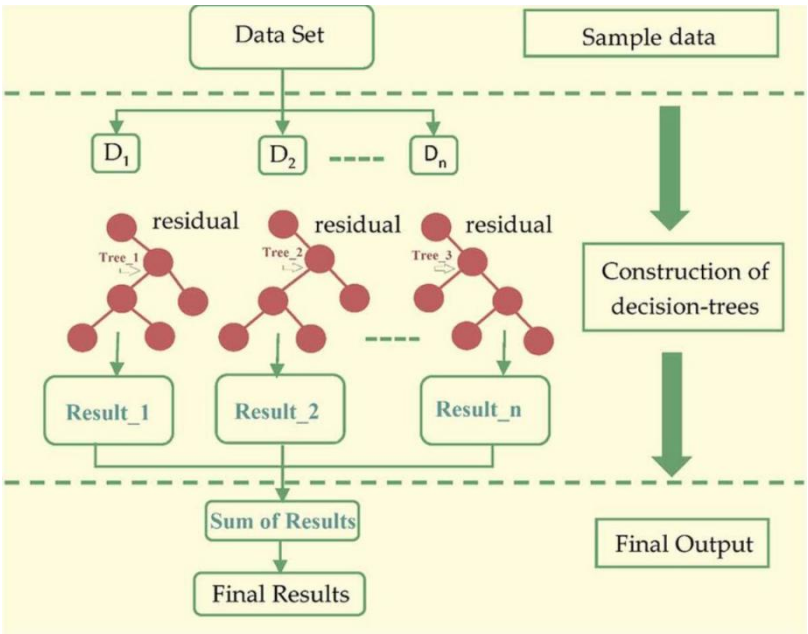
Feature Datatypes: Numerical data, Categorical data need to be encoded as one-hot encoding or Label encoding

How XGBoost Works

XGBoost's core idea is to build on the principles of gradient boosting with three key enhancements:

1. **Second-Order Taylor Expansion:** Instead of just using the first-order gradient (like traditional boosting), XGBoost uses a second-order approximation of the loss function. This allows it to make more precise and efficient steps towards the optimal solution, leading to faster convergence.
2. **Regularization:** XGBoost includes **L1 and L2 regularization** directly in its objective function. This penalizes the complexity of the trees, helping to prevent the model from overfitting the training data.
3. **Parallel Processing:** Although the boosting process is sequential, XGBoost optimizes the tree-building process. It can parallelize the computation of splitting a node in a tree, which significantly speeds up training, especially on large datasets.

The final prediction is the sum of the predictions from all the individual trees, where each tree contributes to the final score.



Simple Formula Explanation

The basic goal of XGBoost is to minimize an objective function that has two parts:

- **Loss Function:** This measures how well the model predicts the training data.
- **Regularization Term:** This penalizes complex trees to prevent overfitting.

The formula can be simply expressed as:

Objective=Loss Function + Regularization Term

The algorithm iteratively adds trees to minimize this objective

Advantages

- **High Performance:** It is known for its speed and accuracy, often outperforming other algorithms on structured data.
- **Regularization:** The built-in regularization prevents overfitting, which is a common problem with traditional boosting models.
- **Handling Missing Data:** It can automatically handle missing values by learning the best direction to send a sample with a missing feature.
- **Flexibility:** It can be used for various tasks, including classification, regression, and ranking.
- **Scalability:** Its optimizations allow it to run efficiently on large datasets and distributed systems.

Disadvantages

- **Complexity and Parameter Tuning:** XGBoost has a large number of hyperparameters, and it can be a complex, time-consuming process to tune them correctly for optimal performance. An improperly tuned model may overfit or perform poorly.
- **High Memory Consumption:** The algorithm can be memory-intensive because it needs to store data and intermediate calculations in memory. This can be a limitation when working with very large datasets.
- **Overfitting Risk:** While it has strong built-in regularization, if not properly configured, XGBoost can still overfit the training data, especially with deep trees and a high learning rate.

	Model Type	**Purpose**	**Data Types**	**Model Complexity**	**Key Characteristics**
Logistic Regression (LR)	Linear Model	Primarily Classification	Numerical (Categorical need encoding)	Low	Interpretable, good baseline. Assumes linear relationship, prone to underfitting.
Decision Trees (DT)	Tree-Based Model	Classification & Regression	Numerical and categorical	Variable	Highly interpretable. Prone to overfitting, not very robust.
Random Forests (RF)	Ensemble (Bagging)	Classification & Regression	Numerical and categorical	High	Robust & powerful, reduces overfitting. Less interpretable,

					slower than a single tree.
Support Vector Machines (SVM)	Linear/Non-linear	Classification & Regression	Numerical (Categorical need encoding)	Variable	Effective in high dimensions, memory efficient. Not suitable for very large datasets, lacks direct probability outputs.
XGBoost (XGB)	Ensemble (Boosting)	Classification & Regression	Numerical and categorical	Very High	High performance, handles missing values. Prone to overfitting if not tuned, less interpretable.

8. Hyperparameter Tuning

Hyperparameter tuning, also known as hyperparameter optimisation, is the process of finding the optimal set of hyperparameters for a machine learning model to achieve the best possible performance. Unlike model **parameters** that are learned from data during training (e.g., weights), **hyperparameters** are external configuration variables that a data scientist sets before the training process begins (e.g., learning rate, number of layers, or a decision tree's depth).

8.1 Need for Hyperparameter Tuning

Hyperparameter tuning is crucial because the performance of a machine learning model is highly sensitive to the values of its hyperparameters. Setting the wrong hyperparameters can lead to a model that is either **underfitting** or **overfitting** the data.

- **Underfitting:** Occurs when the model is too simple to capture the underlying patterns in the data, often resulting from hyperparameters that make the model's capacity too low (e.g., too few layers in a neural network).
- **Overfitting:** Occurs when the model learns the training data and noise too well, failing to generalize to new, unseen data. This can happen with hyperparameters that make the model too complex (e.g., a very high learning rate or an excessive number of trees in a random forest).

By tuning hyperparameters, you can strike the right balance between bias and variance, which helps the model generalize well and achieve optimal performance on new data.

8.2 Types of Hyperparameter Tuning

There are several methods for hyperparameter tuning, ranging from simple to more sophisticated, automated techniques. The most common types include:

- **Manual Search:** This is the most basic approach, where a data scientist manually tries different hyperparameter combinations based on intuition and experience, training the model for each combination. While this can provide good results for simple models, it's inefficient and not scalable.
- **Grid Search:** This method exhaustively searches through all possible combinations of hyperparameters from a predefined grid. For example, if you want to tune learning_rate with values [0.1, 0.01, 0.001] and number_of_layers with values [2, 3, 4], a grid search will train the model for all nine combinations. It's thorough but computationally expensive, especially with many hyperparameters or large ranges.
- **Random Search:** Instead of checking every combination, random search selects a fixed number of hyperparameter combinations from the specified search space at random. This is often more efficient than grid search because it can find a good set of hyperparameters more quickly, particularly when only a few hyperparameters significantly impact performance.
- **Bayesian Optimization:** This is a more advanced and efficient method. It uses a probabilistic model (often a Gaussian Process) to predict the model's performance for a given set of hyperparameters. It then intelligently chooses the next set of hyperparameters to try based on this model, aiming to minimize the number of trials needed to find the optimal configuration. It's more computationally efficient than grid and random search as it learns from past results.

8.3 How to Select Parameters

To select the best hyperparameters, you should follow a systematic process:

1. **Define a Search Space:** First, identify the hyperparameters you want to tune for your specific model (e.g., max_depth for a decision tree, C for an SVM) and define a reasonable range or set of values for each. Avoid large ranges that would make the search inefficient.
2. **Choose a Tuning Method:** Based on your computational resources and the complexity of the problem, select a tuning method.
3. **Use Cross-Validation:** To get a reliable performance estimate for each hyperparameter combination and prevent overfitting to a single validation set, use **k-fold cross-validation** within your tuning process. This splits the training data into k parts, training on k-1 folds and validating on the remaining one, and repeating the process k times. The average performance across all folds is a more robust metric.
4. **Define an Objective Function:** Specify a metric you want to optimize, such as accuracy, F1-score, or mean squared error. The tuning algorithm will use this metric to evaluate the performance of each trial and identify the best set of hyperparameters.
5. **Iterate and Refine:** The process is iterative. After an initial tuning run, analyze the results. If the best hyperparameters are at the edge of your defined search space, expand the range and run the tuning again. This helps ensure you haven't missed a better-performing combination.

9. Feature Selection

Feature selection is the process of choosing the most relevant features (variables) from your dataset to use in a machine learning model. The goal is to improve model performance, reduce training time, and make the model more interpretable by removing irrelevant or redundant features.

Types of Feature Selection Methods

9.1 Filter Methods

- **How it works:** Features are evaluated using statistical tests, independently of any machine learning model. You rank features based on their scores and keep the best ones.
- **When to use:** Great for large datasets. It's fast and computationally cheap, but it ignores how features interact with each other.
- **Examples:** Correlation, Chi-Squared test ($\chi 2$), and ANOVA.

9.2 Wrapper Methods

- **How it works:** A machine learning model is trained on different subsets of features. The subset that results in the best model performance is chosen.
- **When to use:** Ideal for small to medium-sized datasets. It's highly accurate because it considers feature interactions, but it is very computationally expensive.
- **Examples:** Recursive Feature Elimination (RFE) and Forward/Backward Selection.

9.3 Embedded Methods

- **How it works:** The feature selection is built directly into the model's training process. The model selects the best features for itself during training.
- **When to use:** A great balance of speed and accuracy. It's more efficient than wrapper methods and more accurate than filter methods.
- **Examples:** Lasso (L1 regularization), Ridge (L2 regularization), and tree-based models like Random Forest and XGBoost, which provide feature importance scores.

9.4 Univariate vs. Multivariate

- **Univariate:** Evaluates each feature **individually** against the target variable. It's a "one-to-one" analysis. Filter methods often use this approach.
- **Multivariate:** Evaluates how a **group of features** performs together. It considers the relationships and interactions among features. Wrapper and embedded methods are inherently multivariate.

When to use

1. **Start with Filter Methods** for a fast initial pass, especially with large datasets.
2. **Move to Embedded Methods** if your chosen model supports it, as it offers a great balance of performance and efficiency.

3. **Only use Wrapper Methods** if you have a small dataset and are aiming for the absolute best performance possible and have the computational time to spare.

10. Evaluation Metrics for Classification Models

Evaluation metrics are crucial for assessing the performance of a classification model. They provide a quantitative way to understand how well the model predicts outcomes

10.1 Confusion Matrix

A **confusion matrix** is a table that summarizes the performance of a classification model. It compares the model's predictions to the actual ground truth values.

The matrix has four key components, which are essential for calculating other metrics:

- **True Positive (TP):** The model correctly predicted the positive class.
- **True Negative (TN):** The model correctly predicted the negative class.
- **False Positive (FP):** The model incorrectly predicted the positive class (a Type I error).
- **False Negative (FN):** The model incorrectly predicted the negative class (a Type II error).

The total number of predictions is the sum of all these values (TP+TN+FP+FN)

10.2 Accuracy

Accuracy measures the proportion of total predictions that were correct. It is the most intuitive metric, but it can be misleading, especially with imbalanced datasets. Use accuracy when the classes in your dataset are **roughly balanced**

Definition: The number of correct predictions divided by the total number of predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Interpretation:

- A value of 1.0 indicates a perfect model (100% correct predictions).
- A value of 0.5 means the model is no better than random guessing (for a balanced binary classification problem).

10.3 Precision

Precision answers the question: "Of all the positive predictions my model made, how many were actually correct?" It focuses on the model's ability to avoid false positives.

Definition: The number of true positive predictions divided by the total number of positive predictions (True Positives + False Positives).

$$Precision = \frac{TP}{TP + FP}$$

Interpretation:

- High precision means a low false positive rate.
- This is a critical metric when **the cost of a false positive is high**. For example, in a spam email filter, high precision ensures that important emails are not incorrectly flagged as spam.

10.4 Recall (Sensitivity)

Recall answers the question: "Of all the actual positive cases, how many did my model correctly identify?" It focuses on the model's ability to find all positive instances.

Definition: The number of true positive predictions divided by the total number of actual positive cases (True Positives + False Negatives).

$$Recall = \frac{TP}{TP + FN}$$

Interpretation:

- High recall means a low false negative rate.
- This metric is crucial when the **cost of a false negative is high**. For example, in a medical diagnosis for a serious disease, high recall ensures that few cases are missed.

10.5 F1-score

The F1-score is the harmonic mean of precision and recall. It is a single metric that provides a balanced view of both precision and recall.

Definition: The harmonic mean of precision and recall, giving equal weight to both.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Interpretation:

- The F1-Score ranges from 0 to 1.
- It's a useful metric when you need a balance between precision and recall, especially in cases with imbalanced classes. A high F1-Score indicates that the model has both good precision and good recall.

10.6 Classification Report

A **classification report** is a text-based summary of the main classification metrics. It conveniently organizes precision, recall, and F1-score for each class in a multi-class problem, along with other key values like accuracy and support.

Interpretation:

- **Precision, Recall, F1-Score:** These columns show the metric value for each class.

- **Support:** This column indicates the number of actual occurrences of each class in the dataset.
- **Accuracy:** A single value for the overall model performance.
- **Macro Avg:** The unweighted average of the metrics for all classes.
- **Weighted Avg:** The average of the metrics for all classes, weighted by the number of instances (support) for each class. This is usually more representative for imbalanced datasets.

10.7 ROC Curve

The ROC (Receiver Operating Characteristic) Curve is a plot that helps you visualize how well your classification model can distinguish between two classes (diseased vs. healthy).

Imagine your model gives a probability score for each prediction (e.g., an email has a 95% chance of being spam). To make a final decision, you have to set a **threshold** (e.g., if the score is > 50%, classify it as spam).

The ROC curve plots the model's performance at all possible thresholds, from 0 to 1. Specifically, it plots two things:

- **True Positive Rate (y-axis):** This is the same as **Recall**. It tells you how many of the actual positive cases your model correctly identified.
- **False Positive Rate (x-axis):** This is the proportion of actual negative cases your model incorrectly identified as positive.

A good model's curve will bend up toward the top-left corner. This means the model can achieve a high True Positive Rate without a high False Positive Rate. A model with no predictive power (random guessing) will follow the diagonal line.

10.8 AUC

AUC (Area Under the Curve) is a single number that summarizes the performance of the ROC curve. It measures the entire area underneath the ROC curve.

Think of it as a score from 0 to 1. The higher the AUC, the better the model's ability to distinguish between classes.

- **AUC = 1:** Perfect model. It can separate the classes completely.
- **AUC = 0.5:** The model is no better than random guessing. It has no ability to separate the classes.
- **AUC > 0.5:** The model has some discriminatory power. The closer the AUC is to 1, the better.

In summary, the **ROC Curve** shows you a visual trade-off between the true positive and false positive rates at different thresholds, while the **AUC** provides a single, easy-to-interpret number that tells you how good the model is overall at this task.

11. Interpretation

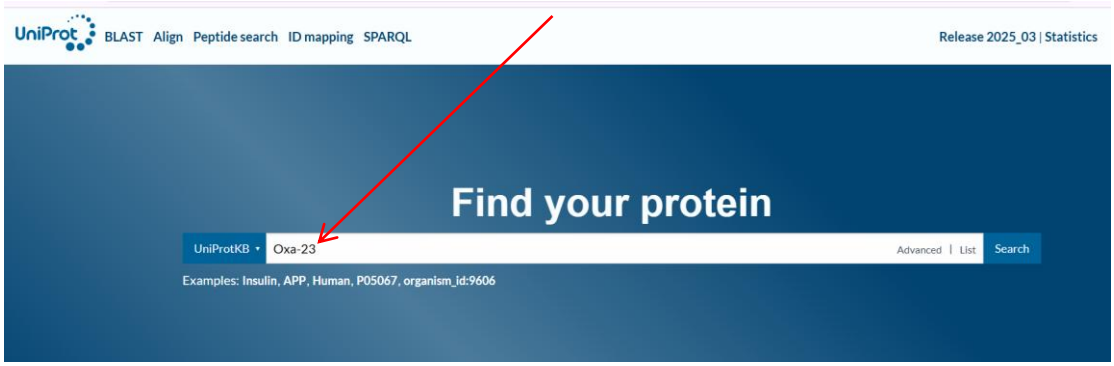
Gene based and SNP based biological significance of top features from various ML Models

After successful identification of top gene clusters (top Features) which were consistently prioritized across models in order to get biological importance or their role we have to go for functional annotation and GO/KEGG pathway enrichment analysis. Here we have taken 25 top features (Fig.1) across all the 5 models

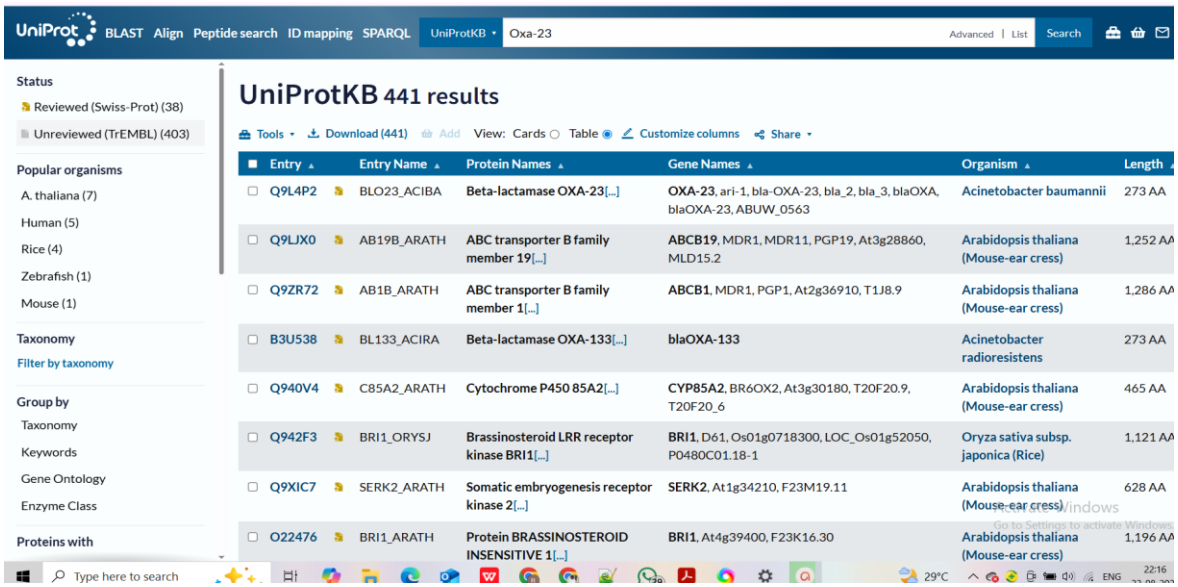
GENE	Coef
OXA-23	1.621858446128929
ADC-73	1.0495716192557019
ADC-30	1.020906315623183
Acinetobacter_baumannii_AbaQ	0.48443104651050445
Acinetobacter_baumannii_AbaF	0.4267450381699455
tet(B)	0.34735141563440813
ADC-25	0.3
TEM-12	0.26216864740911594
armA	0.232637386503812
AAC(6')-Ib7	0.1357186272782454
adeN	0.13551221469519653
qacEdelta1	0.11638650176448401
sul1	0.11638650176448401
adeA	0.009713485799414279
sul2	8.821415485077422e-0
OXA-71	0
ADC-79	0
OXA-66	-0.001658
tet(39)	-0.004402
mphE	-0.067363
msrE	-0.067363
APH(3'')-Ib	-0.120318
APH(6)-Id	-0.120318
ANT(2'')-Ia	-0.16767

For functional annotation:

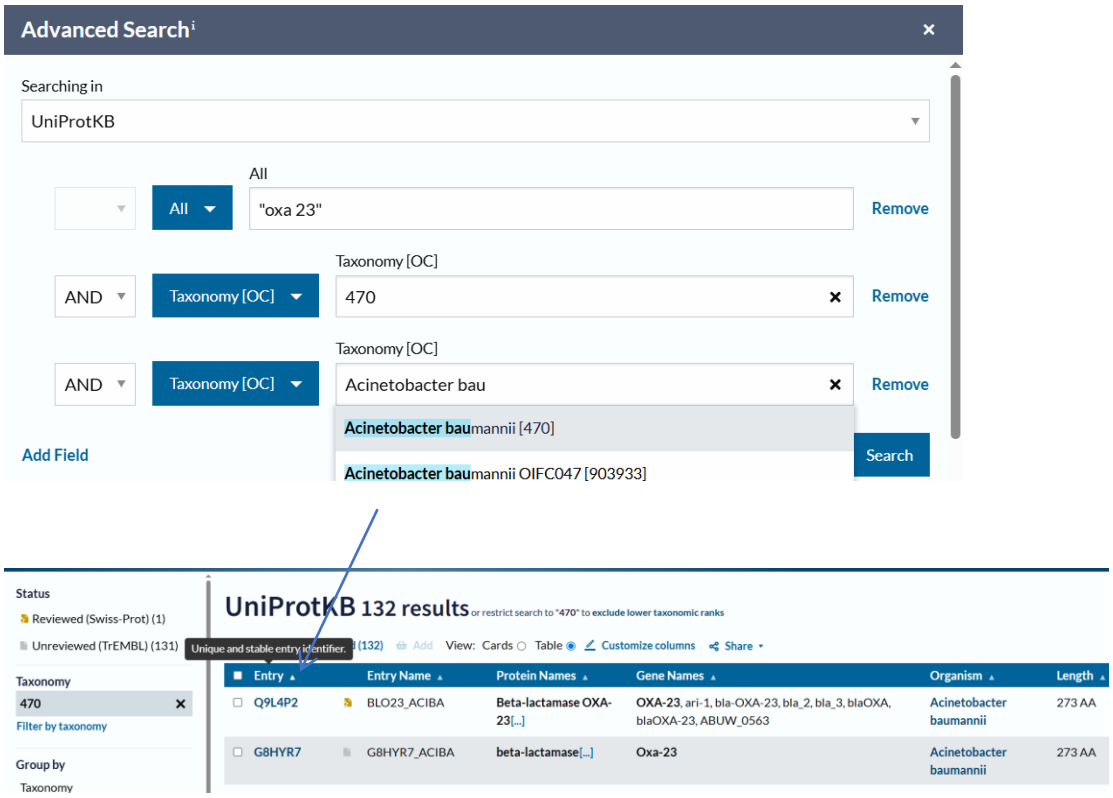
Go to Uniprot: https://www.uniprot.org/



2. Enter the gene name and click search
3. Search for your species of interest. For example, *Acinetobacter baumannii* is there in the left side window of the Uniprot page.

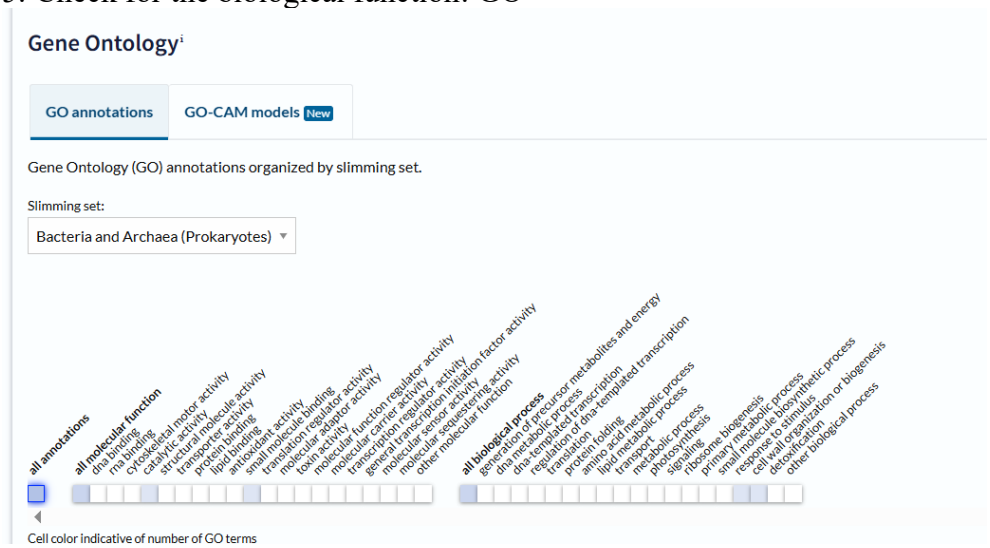


4. IF it is not there. GO to Taxonomy and type *Acinetobacter baumannii*

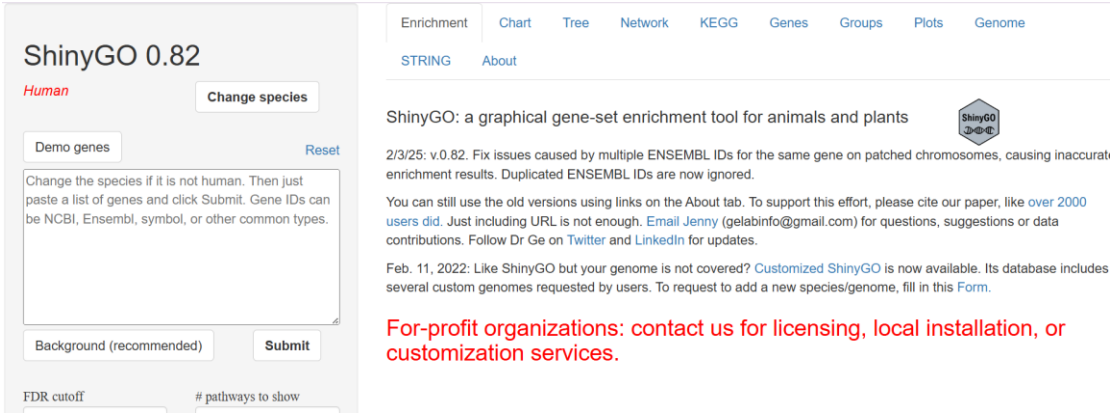


4. Click the entry.

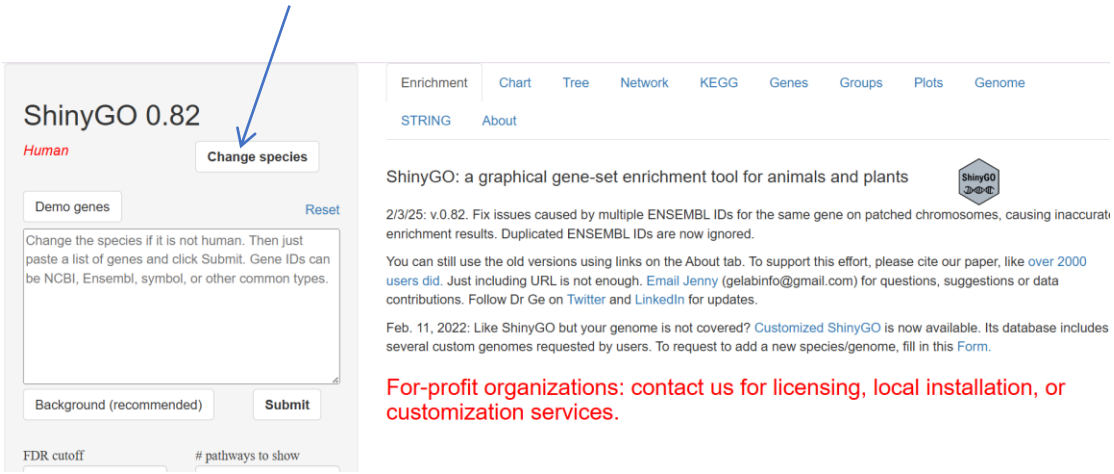
5. Check for the biological function: GO



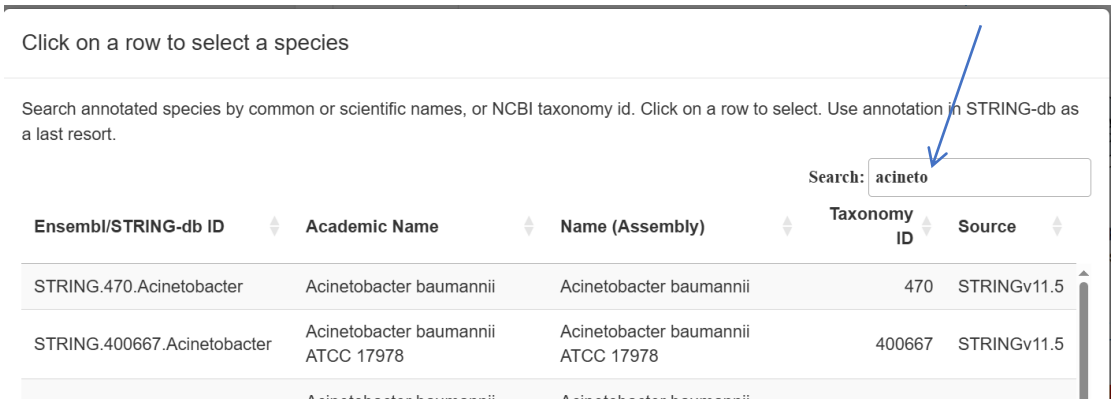
6. For Bulk retrieval of GO and KEGG. Go to ShinyGO 0.82 (https://bioinformatics.sdstate.edu/go/)



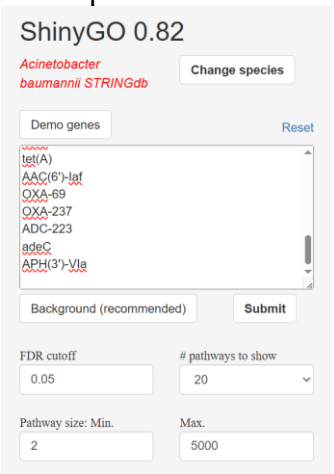
7. Change the species of your interest by click “Change species”



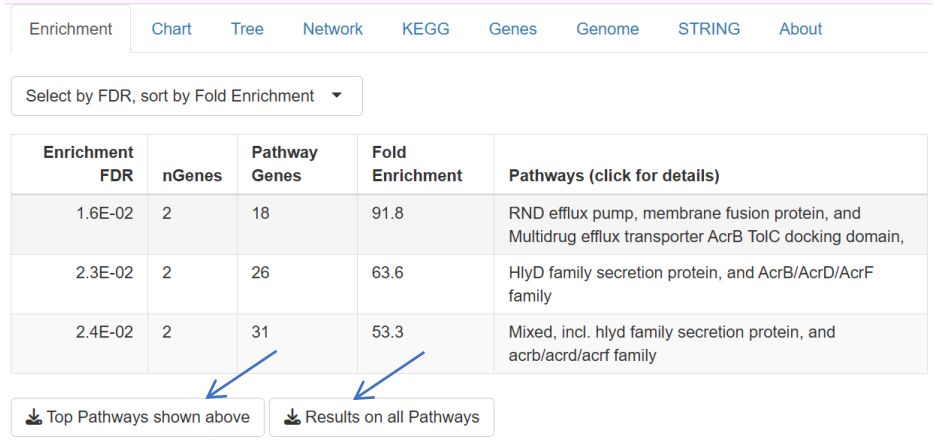
8. Here in this case it is *Acinetobacter baumannii*. So type the species name in Search box and select the hit and click Dismiss



9. Enter or give all the gene names as input here



10. Click submit. You can download your results

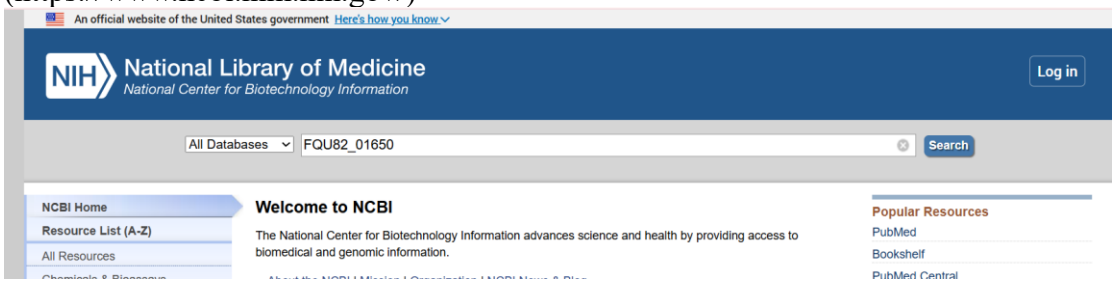


11. For complete GO and KEGG pathway enrichment analysis it is very difficult it for *Acinetobacter baumannii*.

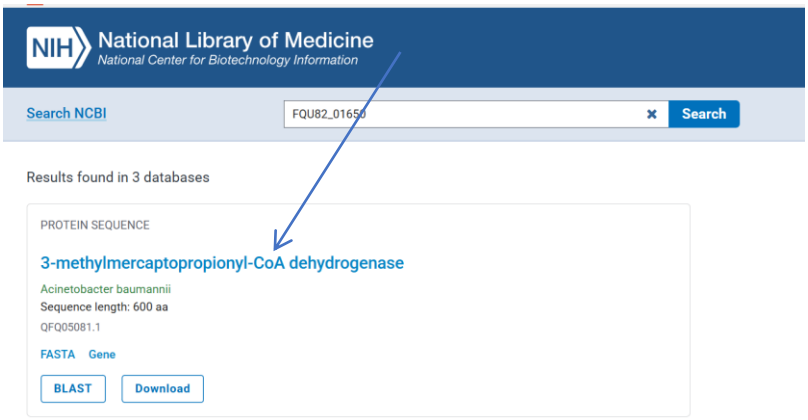
This is a very common situation for non-model organisms like *Acinetobacter baumannii*, which often have incomplete GO/KEGG annotations. Using homologs in a well-annotated organism like *E. coli* is a valid strategy

So in this case we need to download the sequence of all these genes(protein) by downloading the corresponding reference genome genbank (.gbk) file.

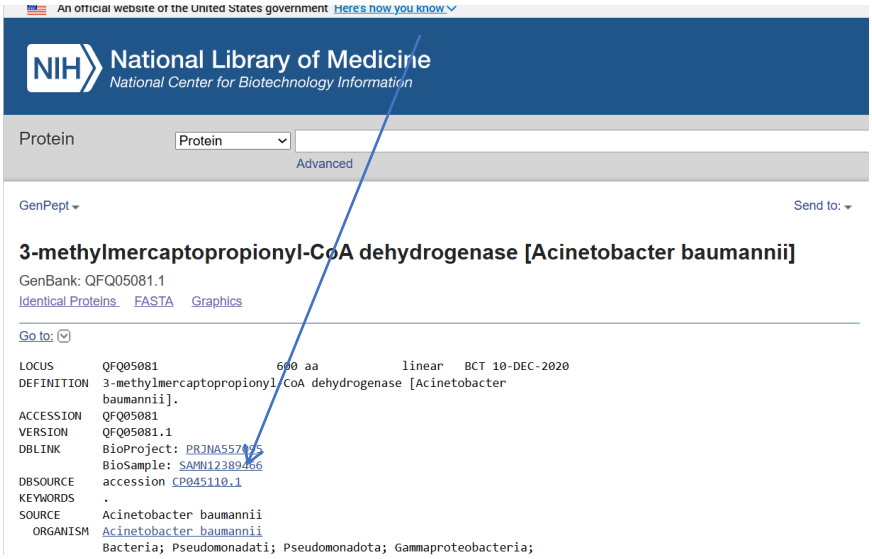
12. Copy any single tag from the SNP based feature result file and paste it in NCBI portal (https://www.ncbi.nlm.nih.gov/)



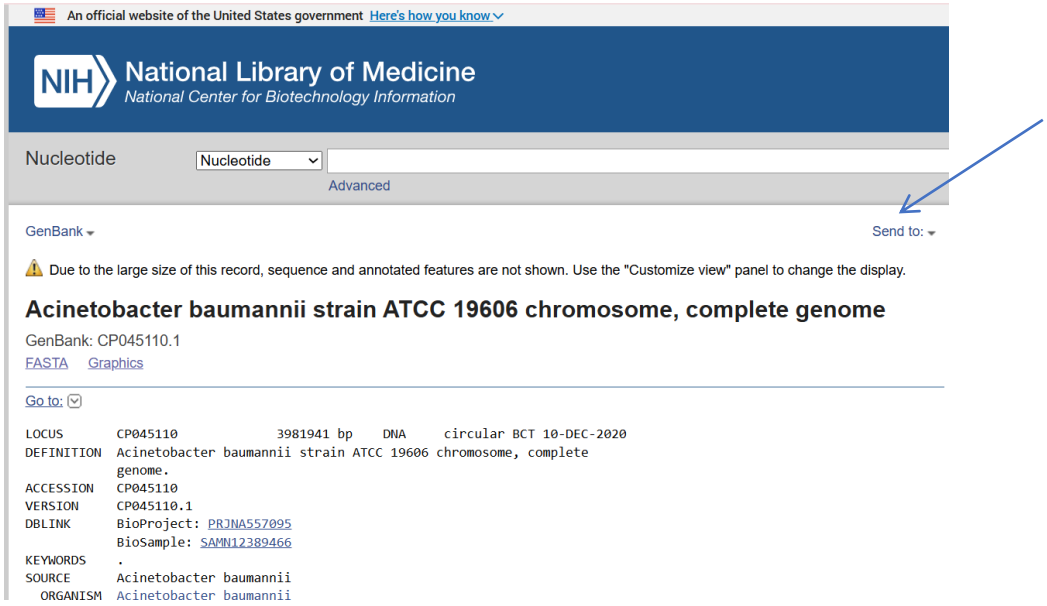
13. Click the species name in the result page, like shown by the arrow mark



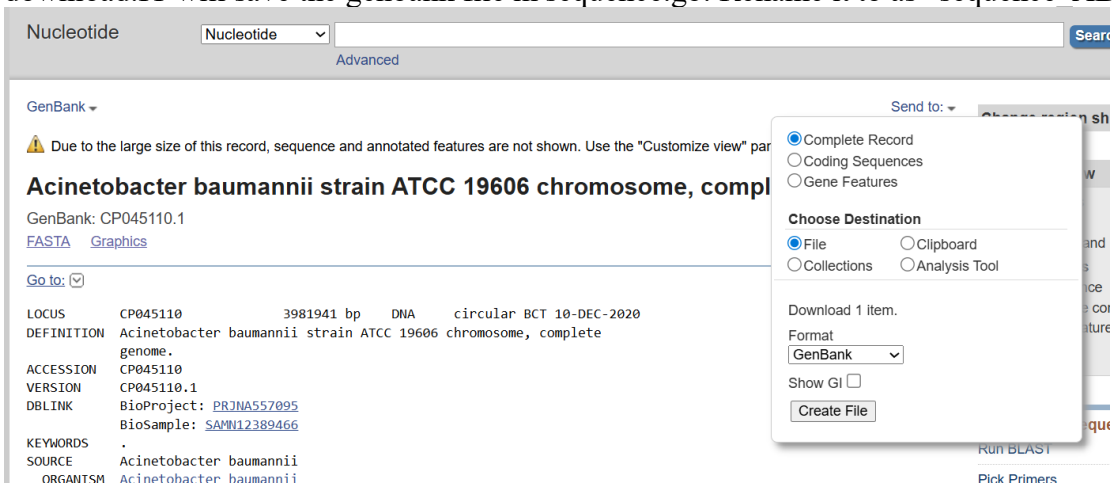
14. Check for the DB source accession in the result page



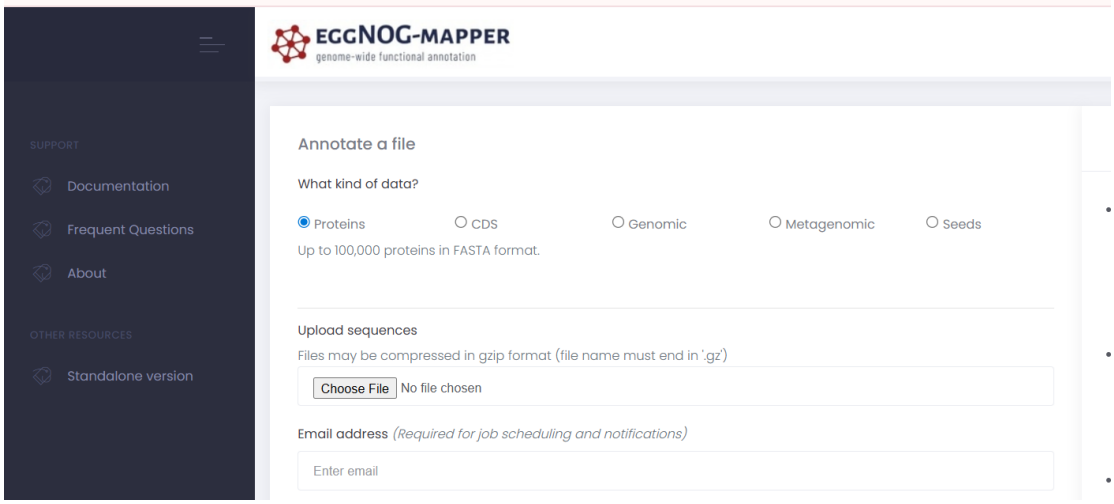
15. Here, Genbank accession is CP045110.1 and thus click it



16. Click send to…and then Choose File and select Genbank (Full) from that option and click download.IT will save the genbank file in sequence.gb. Rename it to as “sequence_AB.gb”



17. Create a text file to save all the IDs as a result of SNP you got. Here I have saved it in the name A_baumanni.txt.
18. Now using these two file, you can download all the protein sequences in fasta format using python script from Google colab.
19. Once downloaded all the sequences. Go to Eggnog mapper



20. Upload the protein sequences and give your email ID. You will get an email link. Click “Manage your job”
21. Need to activate or start your job.
22. Then click “Access your job here”. You will get the result there, and you can get a biological interpretation from those results

To retrieve protein sequences (Reference genome) in batch using google colab:

```
from google.colab import drive
drive.mount('/content/drive')

%cd /content/drive/MyDrive/

!pip install biopython

from Bio import SeqIO
import os

# ------------------------------
# USER PARAMETERS
# ------------------------------

# Google Drive paths to your files
input_locus_file = "/content/drive/MyDrive/A_baumanni.txt"
genbank_file = "/content/drive/MyDrive/sequence_AB.gb"

# Output folder (will create in Colab environment or in Drive)
output_folder = "/content/drive/MyDrive/Protein_FASTA"
output_fasta_name = "ab_proteins.fasta"

# ------------------------------
# SCRIPT START
# ------------------------------

# Make sure output folder exists
os.makedirs(output_folder, exist_ok=True)

# Read locus tags into a set for faster lookup
with open(input_locus_file) as f:
    locus_tags = set(line.strip() for line in f)

# Path for output FASTA
output_fasta_path = os.path.join(output_folder, output_fasta_name)

# Counter for found sequences
found_count = 0

# Open output FASTA for writing
with open(output_fasta_path, "w") as out_fasta:
    # Parse GenBank file
    for record in SeqIO.parse(genbank_file, "genbank"):
        for feature in record.features:
            if feature.type == "CDS" and "locus_tag" in feature.qualifiers:
```

```
            tag = feature.qualifiers["locus_tag"][0]
            if tag in locus_tags:
                # Get protein sequence
                seq = feature.qualifiers.get("translation", [""])[0]
                if seq:
                    out_fasta.write(f">{tag}\n{seq}\n")
                    found_count += 1

print(f"Done! Found and wrote {found_count} sequences to:\n{output_fasta_path}")
```

12. References

Scikit-learn. "3.2. Tuning the hyper-parameters of an estimator." Scikit-learn Documentation.

Jain, O. (2024, May 15). *Unlock the power of logistic regression: A comprehensive introduction*. Medium. https://osheenjain.medium.com/unlock-the-power-of-logistic-regression-a-comprehensive-introduction-e0e8ba98917d

GeeksforGeeks. "Decision Tree Algorithms". https://www.geeksforgeeks.org/machine-learning/decision-tree-algorithms/

Bhakta, S. S. (2025, July 23). *Random Forest Algorithm in Machine Learning*. GeeksforGeeks. Retrieved August 23, 2025, from GeeksforGeeks website

IBM. "Support vector machines (SVMs)". Retrieved from https://www.ibm.com/think/topics/support-vector-machine\

Ahmed, Saghir & Raza, Basit & Hussain, Lal & Aldweesh, Amjad & Omar, Abdulfattah & Khan, Mohammad Shahbaz & Tageldin, Elsayed & Nadim, Muhammad. (2023). The Deep Learning ResNet101 and Ensemble XGBoost Algorithm with Hyperparameters Optimization Accurately Predict the Lung Cancer. Applied Artificial Intelligence. 37. 10.1080/08839514.2023.2166222.

Ali, S., & Awan, S. M. (2021). Supervised learning algorithms and evaluation metrics in machine learning. *Mathematical Problems in Engineering, 2021*, 1–17. https://doi.org/10.1155/2021/4761306