# Antibiotic Resistance Gene Prediction (Deep Learning)

**\*What is it and why is it required?**

Antibiotic resistance is the process exhibited by bacterial species (mainly); this allows the bacteria to grow even in the presence of an antibiotic exhibiting resistance and no sensitivity to the antibiotic. Such resistances are incorporated due to the production of specific resistance mechanisms or proteins which either cleave the antibiotic (when enters the bacteria) or inactivates it by destroying the Beta-Lactam rings (penicillin-based antibiotics).

The prediction of antibiotic resistance exhibited by a particular microbe is extremely crucial for healthcare and medicine as it would ensure appropriate dosage and incorporation of antibiotic (to which microbe is not resistant) at the early cycles of a particular infection.

In this project CNN (Convolution Neural Network) based learning and prediction models would be employed to predict AMR (Anti-Microbial Resistance). Appropriate AMR +ve and AMR -ve dataset would be used to train, test and validate the models.

The aim of this project is to make a robust prediction model which distinguish ARG (Antibiotic Resistance Gene) and Non-ARG from a data which is unknown or novel.

**GOAL**

Predict antibiotic resistance genes (ARGs) using deep learning from DNA sequences.

**TOOLS**

Python, Deep Learning using Convolution Neural Network (CNN)

**NOTE: All of the codes have been compiled into a jupyter notebook titled “Codes for 3rd Project.ipynb”, which code to refer has been mentioned with its specific number at the end of each step.**

**STEPS FOLLOWED:  
1. Loading Libraries in VS Code**

import **os**

from Bio import SeqIO

import numpy as **np**

import **random**

from **sklearn**.**model\_selection** import **train\_test\_split**

from **pathlib** import **Path**

from **sklearn**.**model\_selection** import **train\_test\_split**

import tensorflow as **tf**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

**2. Preparing ARGs (Antibiotic Resistance Genes) and Non-ARG sequences:**

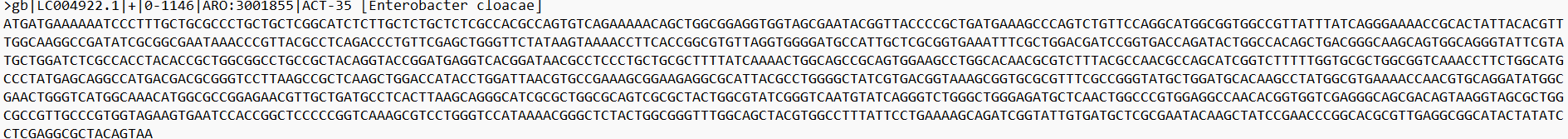
Positive and Negative samples are needed for a deep learning model like CNN so, ARG would serve as positive samples and Non ARG sequences as negative ones.

1. CARD data has been downloaded from: <https://card.mcmaster.ca/latest/data>

This contains several fasta files which give nucleotide and protein sequences using various methods of analysis (like overexpression, variant etc.) but for this project a particular file titled: “nucleotide\_fasta\_protein\_homolog\_model.fasta” serves as a positive sample data.

This file encompasses various SampleIDs (wherein each SampleID denotes a new sample) with nitrogen bases sequences. These nitrogeneous base sequences when transcribed would form RNA sequences which when further grouped by codon (3 RNA sequences = 1Amino Acid) mechanism would form the protein. This protein is responsible for the generation of antibiotic resistance in that particular organism. The data file also contains the Name of the organism in which this particular sequence has been found.

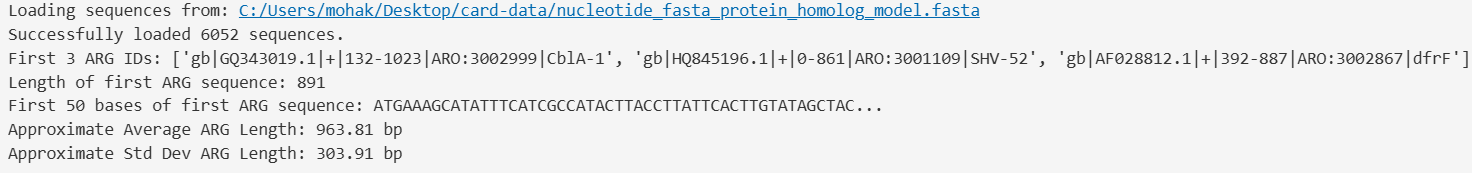
Protein homology model has been used to predict if the sequence would create AMR or not meaning as stated earlier that the 3AA would form 1 protein which would be responsible for antibiotic resistance, this protein (target) is compared to another protein (template) which is homologous to itself. If both the proteins achieve higher levels of matching scores then the target protein is considered as the one which induces the antibiotic resistance.



The image above depicts one of the samples from the fasta file.

gb|LC004922.1|+|0-1146|ARO:3001855|ACT-35 is the ID

*Enterobacter Cloacae* is the organism in which these sequences have been found to induce antimicrobial resistance.

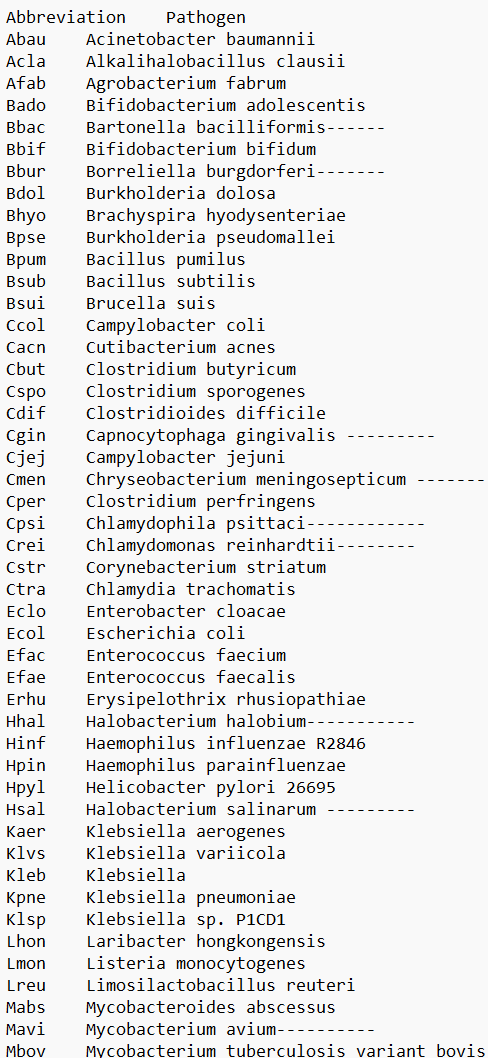


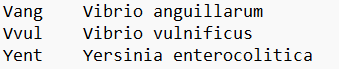
* **6052** are positive samples (show antibiotic resistance)
* **gb|GQ343019.1|+|132-1023|ARO:3002999|CblA-1**: These are the identifiers from the CARD database, showing GenBank IDs, sequence ranges, ARO (Antibiotic Resistance Ontology) IDs, and gene names (CblA-1 etc).
* **891** is total number of ATGC present in the first sample.
* **963.81bp** denotes the average ATGC present in all of the samples.
* **303.91bp** is the standard deviation. These parameters would help in training the model based on the ATGC and not on the total length of ATGC sequences present in each sample which would help to prevent data loss.

**REFER #1 *Collecting ARG Sequences (Positive Samples)***

**3. Preparing Non-ARGs**

We have harvested positive samples but we also need negative samples which would help improve the overall robustness of the model. Now, a list of organisms has been given by the card-data file and these are given below:

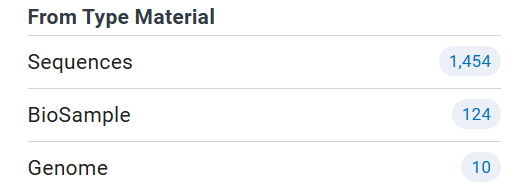




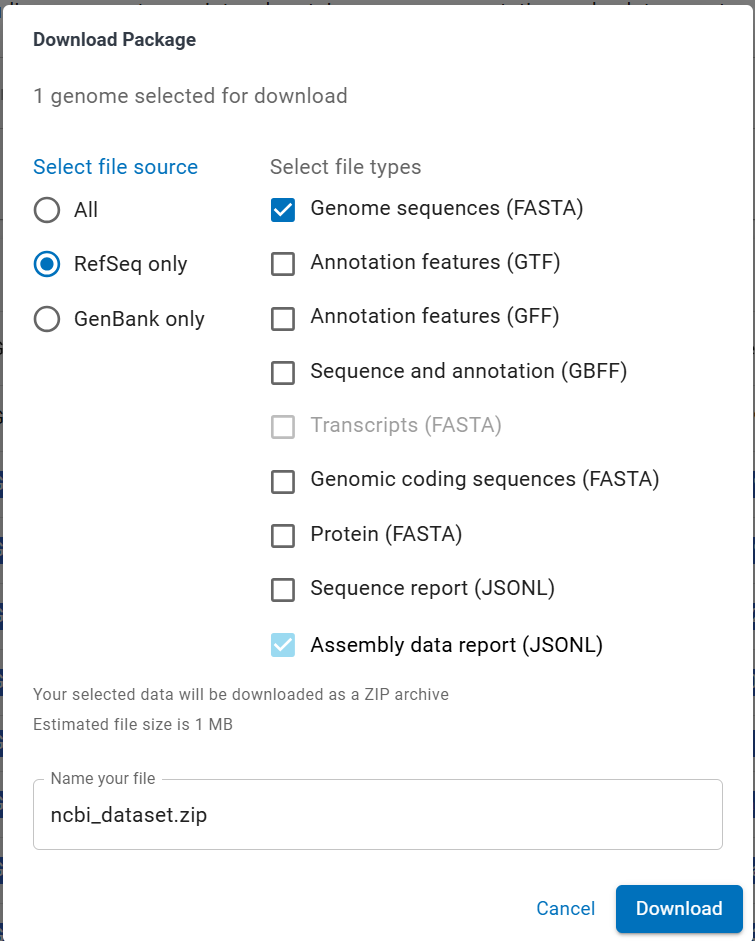


The genomic data for all of the mentioned organisms have been downloaded from <https://www.ncbi.nlm.nih.gov/datasets/>

* Consider the first one: *Acinetobacter baumannii*
* Steps to download the genomic data:
* Search *Acinetobacter baumannii* in the search box.



Navigate to “Genome”

* 
* A green tick dataset has been downloaded. This particular tick denotes that extreme measures have been taken to ensure that the sequence is correct and also the full sequence for that particular strain might be recorded.
* Click on the  icon and “Download” option.
*  All the genomic data for the succeeding organisms must be downloaded using the same process.
* The organism’s genome being extremely data-rich needs to be prepared to match with our ARG sequences length. As we have found that average length or average number of nucleotides the ARG contains are 963.81 bp with a standard deviation of 303.91 bp, it is evident that the Non-ARG sequences (being in thousands of base pairs) need to be reduced and prepared roughly the similar lengths as that of the ARG sequences.

OUTPUT:



The overall genomic data of 97 organisms has lacks of nucleotides so, for CNN the dimensions of both the positive and the negative samples need to be relatively equal.

Every time the Non-ARGs are prepared a random fragment which is roughly the length of the ARG sequence (sometimes counts the standard deviation) is taken out. Like take for example the genomic data of *Acinetobacter Baumanii*, this particular organism has lacks of base pairs so, out of them only 993 (for example) are taken which roughly match our average ARG length. Every time the code cell is run, a new fragment of random length (but always roughly equal to the average ARG length) is taken from each of the organisms genomic DNA data file (in .fna).

**\*Pre-requisite:**

Prior step to installing TensorFlow is the installation of Python 3.12 version. TensorFlow is not available for latest python versions especially above 3.12

**REFER #2 *Preparing Non ARG***

**4. Data Splitting for CNN:**

The Data has been split into train, test and validation set meaning the overall positive [1] and negatively labelled [0] (where “1” and “0” are labels given to each sequence in respective category) samples have been initially combined into a single variable and it has been split. The 70% has been used for training, 15% for testing and rest of the 15% for validation.

* Over here the “**MAX\_SEQUENCE\_LENGTH = 2000**” variable has been used to define a maximum sequence length that the CNN model should be expecting. This has been done so, that the model can allocate specific memory to train, test and validate itself upon the data provided.
* A point worthy to note is that we have not defined each sequence to be of 2000 base pairs long but rather we have defined that the sequences that CNN would be expecting would not exceed 2000 base pairs so, the actual length of the sequence may vary but it is evident to be less than 2000 base pairs.

**REFER #3 *Data splitting for CNN***

**5. Encoding:**

The DNA sequences are in the form of alphabets, particularly “A”, “T”, “G”, “C” and occasional “N”, this makes the model incapable of reading the DNA sequence so, to solve this issue encoding of each of the nitrogenous bases comes into play.

For this project One-hot encoding has been used which to put simply, encodes each letters in binary form (0s and 1s) which the model can read and interpret easily.

For Example:

Consider a DNA sequence AATGCCG

This sequence is humanized but cannot be interpreted by the model so, it needs to be converted in a form which the machine can read and interpret easily and this conversion is called encoding, meaning we are “encoding” a sequence which is interpretable for the machine into a variable which is in humanized form.

Encodings are:

**A = [1,0,0,0]**

**T = [0,1,0,0]**

**C = [0,0,1,0]**

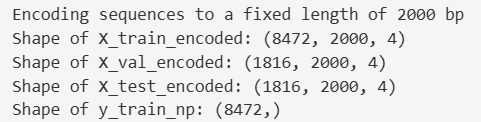
**G = [0,0,0,1]**

**N = [0,0,0,0]**

Based on this encoding our DNA sequence in computerised form would look like this:

Humanized: AATGCCG

Encoded: **[1,0,0,0] [1,0,0,0] [0,1,0,0] [0,0,0,1] [0,0,1,0] [0,0,1,0] [0,0,0,1]**



The figure above depicts the completion of encoding and also depicts the overall shape of the encoded data set which would be used for training, testing and validation.

Consider X\_train\_encoded

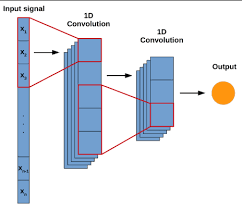
8472, 2000, 4 are the dimensions of this variable meaning the model will be trained on 8472 samples with each sample being encoded of 2000 base pairs with only 4 as encoding dimension.

To explain, take 1 sample from 8472 samples, this sample has consider 982 base pairs of positive (ARG gene) and 500 base pairs of negative (Non-ARG) sequence remaining 518 are blank spaces. Now, these blank spaces have been formed due to our “MAX\_SEQUENCE\_LENGTH=2000” variable meaning our encoding would use [0,0,0,0] array for these blank spaces.

The reason to do so is one, “Making the overall data dimensions identical”. This means that all our training, testing and validation encoded data variables store overall data which has identical dimension of length being 2000 but each individual ARG and Non-ARG has varied sequence. So, considering X\_train\_encoded there are 8472 samples being taken all of which have a base pair length of 2000 and 4 encoding dimensions which are “A”, “T”, “C”, “G”. Note that “N” has not been considered here because “N” is our 0-array meaning wherever there are blank spaces (like 518 for example) they would be filled with the 0 array so, that the dimension of 2000 base pair length can be achieved.

**REFER #4 *Encoding***

**6. Defining and Training the CNN model:**



A diagrammatic representation of how 1-Dimensional Convolution Neural Network works.

Over here the Input Signal is the combined DNA sequence (Containing both negative and Positively labelled sequences).

The first layer identifies this sequence as positively or negatively labelled sequences.

The second layer dives deep into each individual component of this sequence to learn and train itself on patterns of bases so, that prediction can be accurate on novel and unknown data as well.

In the code:

1. Sequential: Used to define the architecture of our model. A list has been passed which will be stacked upon each other.
2. Conv1D(128, 8, activation='relu', input\_shape=(MAX\_SEQUENCE\_LENGTH, 4)),

This is the first layer of our model:

* Conv1D: Defines a 1-D CNN
* 128: Number of filters that our model would have, these filters are used to find out the labels of the sequences as stated earlier.
* 8: Kernel Size, meaning each filter would look for 8-nucleotide patterns
* Activation =’relu’: This induces non-linearity in our model, this is extremely crucial so that the model would know train itself on non-linear relationships of the DNA sequence as well further increasing accuracy and robustness.
* Input\_shape = (MAX\_SEQUENCE\_LENGTH,4): This tells the model that first layer would have sequences with the length of 2000 nucleotide base pairs (reason stated earlier) with 4 encoding dimensions which are “A”,”T”,”C”,”G”.

1. MaxPooling1D(2)

This is the max pooling layer which reduces the dimension or length of the maps and relationship established in the previous layer. The role of “2” here is to select a maximum value.

The maxpooling is like a sliding window meaning this window slides over 2 feature maps which are generated from the first layer, selects the maximum value from those two and passes onto the next layer.

Each filter in the first layer assigns a certain weight to each of the nucleotide this weight is then subjected to the sliding window which chooses the highest amongst two of them.

1. Conv1D(64, 8, activation='relu')

This is the second convolution layer but with lesser number of filters which are 64 in this case. Now, the pooled layer which was generated before is sent to this second layer which then learns and identifies several patterns in-depth.

This again assigns weights to each of the data point which is further processed by pooling.

1. MaxPooling1D(2): Similar to the first layer, this now creates another pooled layer and reduces the dimension.
2. Flatten()

This flattens the multidimensional output generated by both the layers and pooling into one single 1-D vector and marks it as an input data vector.

Note the raw DNA sequence provided earlier (before setting up the model) and this input data vector are completely different. The flattened input data vector contains relationships, patterns of each and every encoded nucleotide with the others in the sample whereas the initial data was just an array of DNA sequence.

1. Dense(128, activation='relu')

This forms a dense connected layer of neurons which are 128 in this case. The relu function helps to learn more non-linear relationships after the network is established.

H. Dropout(0.5)

During each training step 50% of neurons are randomly deactivated to prevent overfitting which increases robustness and overall prediction accuracy.

I. Dense(1, activation='sigmoid'):

This is the final output layer, over here the 1 has been used to denote a binary relationship of output meaning our DNA sequence would be classified into either ARG or Non-ARG, 1 or 0 respectively.

Sigmoid activation helps to let the model understand where the DNA sequence lies to provide better classification. It uses probability function wherein it establishes a 0 to 1 range and fills each of this sequence in this range (based on the CNN) now, if the sequences lie above 0.5 or have a probability greater than 0.5 then they are termed to be ARG. This particular method adds an extra accuracy layer to the final predictions done by the model.

\*Training the model:

A. model.compile('adam', 'binary\_crossentropy', metrics=['accuracy'])

This compiles all the layers to prepare the model to train itself.

* Adam: It is an optimizer algorithm that adjusts the weight of the neural network during training which minimizes the data loss function.
* Binary\_crossentropy: This is the loss function which calculates how wrong the model’s predictions are. The aim to develop any predictor model is to minimize the loss function and maximize the accuracy.
* Metrics=[‘accuracy’]: This evaluates the model’s performance wherein the accuracy is simply the proportion of correctly classified samples.

B. model.fit(

    X\_train\_encoded, y\_train\_np,

    epochs=50,

    batch\_size=32,

    validation\_data=(X\_val\_encoded, y\_val\_np),

    verbose=0 *# Sets verbose to 0 to hide training progress per epoch*

)

* Model.fit: This starts the actual training process
* X\_train\_encoded is our training data
* Y\_train\_np are the corresponding labels to our training data which are 1s for ARG and 0s for Non-ARG DNA sequences.
* Epochs=50: These are the number of epochs which defines one complete pass through the input data. Over here the model would pass 50 times through the input data.
* Batch-size = 32: During each epoch or pass the training data is divided into batches wherein each weight is adjusted after each pass cycle until one weight which is the most accurate is achieved.
* validation\_data=(X\_val\_encoded, y\_val\_np): Defines the validation data
* verbose=0: This controls the information which would be displayed in our console, over here 0 establishes that no information would be displayed on the console during each epoch or pass cycle. This reduces overall load on the system.

**REFER #5 *CNN***

**7. Model Evaluation**

The model is trained, tested and then validated on the validation set.

Test Loss: 0.0809

Test Accuracy: 0.9840

Model Evaluation Complete.

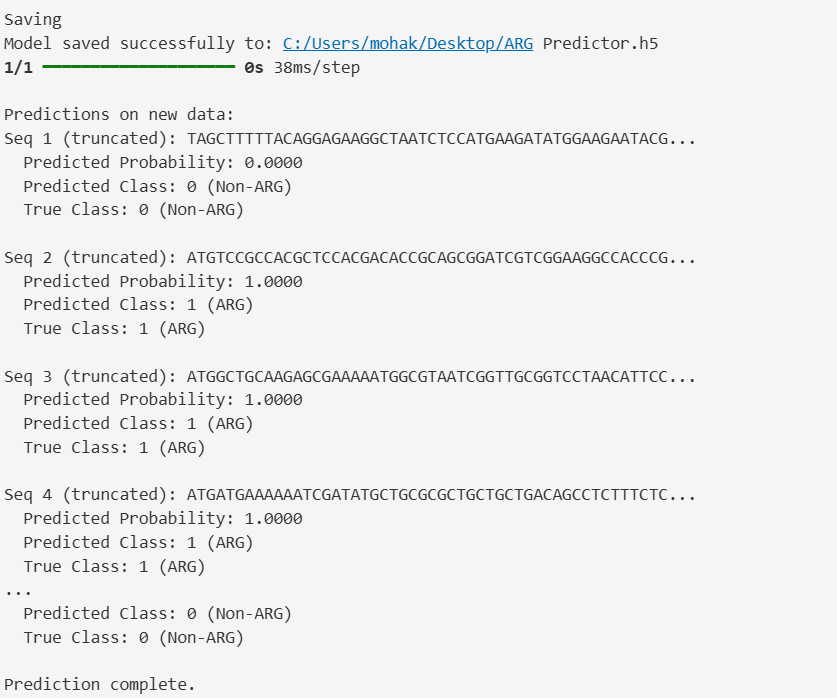
The test accuracy is 98.4% which is extremely high and excellent for any deep learning predictor model.

**REFER #6 *Model Evaluation***

**8. Saving the Model**

The trained model is saved in an “.h5” which is a general extension used to store data-rich compounds which is our model itself. This enables us to directly use the predictor model when a novel fasta sequence is received which allows us to eradicate the hassle of training the model everytime we want to classify a DNA sequence into ARG or Non-ARG.

The model is also deployed on first 5 sequences of randomly split testing data.



new\_dna\_sequences = X\_test[:5]

true\_labels = y\_test[:5]

This specifies that first 5 sequences of the test data must be predicted.

**REFER #7 *Saving the model***

**9. Saving a Predictor.py file which is executable via Command Line Interface**

Saving an executable python script via the command line interface helps to eradicate the hassle of opening VS Code environment which makes the model readily deployable.

**\*NOTE:**The trained model “.h5” file and this python file must be in the same directory in order to ensure the model is deployable. Whenever deploying a model on a novel fasta sequence it is advised to bring this py file (Predictor.py), the models “.h5” file and the fasta sequence on desktop.

**Troubleshooting**:

This particular file has been coded in VS Code’s jupyter notebook which by default saves the files in an “.ipynb” format. This format is not readable by the CLI (Command Line Interface) as it redirects to an online browser which is not needed in this project. So, this file has been converted to “.py” file by using online jupyter notebook’s “Save and Export as----🡪 Executable file” feature.

**REFER “Predictor for 3rd Project.py” for the code.**

**10. Deploying the model on a novel sequence using CLI:**

As stated earlier, the “.h5”, “Predictor.py” and “1.fasta” sequence were moved to the desktop. The “1.fasta” file was the fasta sequence of **mecA gene** known to induce antibiotic resistance, this was completely novel to the model and the reason to download a sequence which is known to be ARG is to let us identify whether the model is predicting the novel sequence accurately or not.

The CLI in windows is opened and these codes are used to run the Prediction:

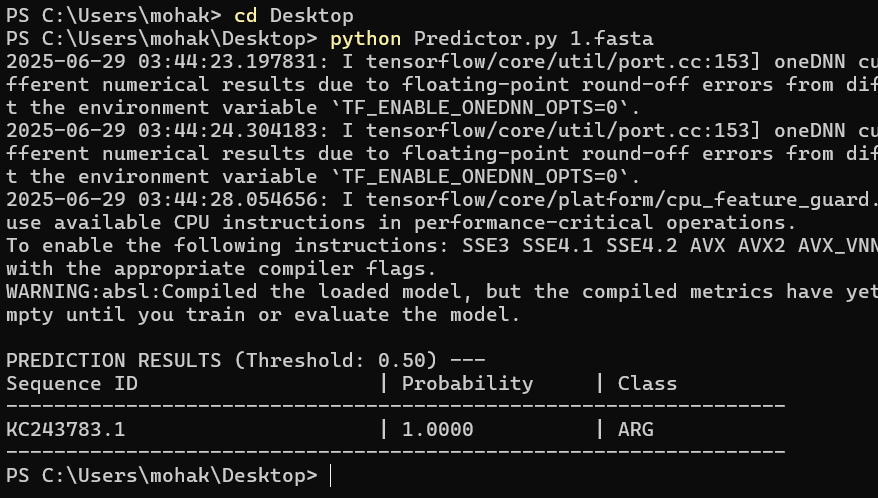
**cd Desktop** //Sets the file path to Desktop (this is where all our executable scripts are)

**python Predictor.py 1.fasta**

This particular code tells the CLI to deploy Predictory.py (which is linked to the already trained model) on the novel fasta sequence which is “1.fasta” in this case:



**OUTPUT:**



The fasta sequence is known to be ARG but completely novel to the model. The accuracy being 98.4% it was evident that the model would predict the novel sequence accurately.

**\*CONCLUSION:**The project accurately predicts a completely novel antibiotic resistance gene with consistent accuracy of 98.4% which is achieved due to accurate processing of both the positive and negative samples and also due to the appropriate methods used in this project.