|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Date | 19 November 2023 | | Team ID | **Team 592303** | | Project Name | Project – Predicting lumpy skin disease | | Maximum Marks | 4Marks |   **Predicting Lumpy Skin Disease** | | | |  | |
| **Introduction**  Lumpy Skin Disease (LSD) is a highly contagious viral disease that affects cattle and poses a significant threat to livestock industries worldwide. Early detection and accurate prediction of LSD outbreaks are crucial for effective disease control and preve  project, we aim to develop a machine learning model to predict the occurrence of Lumpy Skin Disease using a dataset containing various geographical and environmental factors.  **Dataset Description**  The dataset used for this project includes the following columns: | | | |
|  | ● | Longitude (X-axis spatial coordinates) | |
|  | ● | Continent of the outbreak | |
|  | ● | Latitude (Y-axis spatial coordinates) | |
|  | ● | Monthly Cloud Cover in percent | |
|  | ● | Diurnal Temperature Range in degrees Celsius | |
|  | ● | Country of outbreak | |
|  | ● | Frost Day Frequency in a month | |
|  | ● | Potential Evapotranspiration in millimetres per day | |
|  | ● | Precipitation in millimetres per month | |
|  | ● | Daily Mean Temperature in degrees Celsius | |
|  | ● | Temperature in degrees Celsius | |
|  | ● | Monthly Average Maximum and Minimum Temperature in degrees Celsius | |
|  | ● | Vapor Pressure in hectopascals | |
|  | ● | Wet Day Frequency in days | |
|  | ● | Altitude of geographic location in meters | |
|  | ● | Dominant Land Cover | |
|  | ● | Lumpy (target variable) | |

**Project Flow:**

1. Data Collection and Preparation
   * Collect the dataset from reliable sources.
   * Perform data cleaning and preprocessing.
2. Exploratory Data Analysis (EDA)
   * Analyse the dataset using descriptive statistics and visualizations.
   * Explore the distribution of variables and identify any patterns or trends.
3. Feature Engineering
   * Extract relevant features from the dataset.
   * Handle missing values and outliers, if any.
   * Transform categorical variables into numerical representations, if required.
4. Model Building
   * Split the dataset into training and testing sets.
   * Train various machine learning models on the training set.
   * Evaluate the performance of each model using appropriate evaluation metrics.
   * Select the best-performing model for further analysis.
5. Model Evaluation
   * Evaluate the optimized model on the testing set. ● Assess its predictive accuracy and reliability.
6. Model Deployment
   * Deploy the final model to make predictions on new, unseen data.
   * Develop a user-friendly interface or API for easy access to the model's predictions.
7. Documentation and Reporting
   * Prepare a comprehensive project report documenting the entire process.
   * Present the findings, insights, and conclusions derived from the project.
   * Provide recommendations for further improvements or future research.

By accurately predicting the occurrence of Lumpy Skin Disease, this machine learning project can significantly contribute to early detection and effective management of the disease, ultimately leading to improved livestock health and the prevention of economic losses in the livestock industry.

# Milestone 1: Define Problem / Problem Understanding

## Activity 1: Specify the Business Problem

The business problem for the accurate prediction of Lumpy Skin Disease is to develop a machine learning model that can effectively predict the occurrence of Lumpy Skin Disease in cattle. Lumpy Skin Disease is a highly contagious viral disease that affects cattle, causing significant economic losses in the livestock industry. By accurately predicting the disease occurrence, proactive measures can be taken for disease control and prevention, reducing the spread and impact of Lumpy Skin Disease.

## Activity 2: Business Requirements

To ensure that the Lumpy Skin Disease prediction model meets business requirements and can be deployed effectively, the following rules and requirements need to be considered:

|  |  |  |
| --- | --- | --- |
|  | 1. | Accuracy: The model should demonstrate a high level of accuracy in predicting the occurrence of Lumpy Skin Disease. It should provide reliable and precise predictions to support decision-making processes related to disease control and prevention. |
|  | 1. Early Detection: The model should be able to detect the presence of Lumpy Skin Disease at an early stage to facilitate timely intervention and minimize the risk of disease spread within cattle populations. 2. Scalability: The model should be scalable to handle large volumes of data and accommodate future growth in the livestock industry. It should be capable of processing data from multiple sources and adapting to evolving disease patterns. 3. Interpretability: The model should be interpretable, meaning that its predictions can be explained and understood by stakeholders. Interpretability is essential for building trust in the model and enabling informed decision-making based on its outputs. 4. Privacy and Security: The model should adhere to privacy and security regulations to protect sensitive data. Measures should be implemented to ensure secure storage, handling, and access to data used for training and prediction purposes. | |

## Activity 3: Literature Survey

A literature survey for the accurate prediction of Lumpy Skin Disease would involve researching and reviewing existing studies, articles, and publications related to Lumpy Skin Disease in cattle. The survey aims to gather insights on the following aspects:

|  |  |
| --- | --- |
|  | 1. Disease Characteristics: Understanding the aetiology, epidemiology, and clinical manifestations of Lumpy Skin Disease in cattle. Exploring factors that contribute to disease transmission and spread. 2. Risk Factors: Identifying risk factors associated with Lumpy Skin Disease, such as breed susceptibility, age, geographical location, and environmental conditions. 3. Diagnostic Methods: Reviewing existing diagnostic methods for Lumpy Skin Disease, including clinical observations, laboratory tests, and imaging techniques. Exploring their limitations and potential for improvement. 4. Machine Learning Approaches: Investigating previous studies that have utilized machine learning techniques for disease prediction in cattle. Assessing the performance of different algorithms and feature selection methods. 5. Data Availability: Identifying potential sources of data for training and validating the prediction model. Assessing the quality, completeness, and reliability of available datasets. |

The literature survey will help in gaining a comprehensive understanding of Lumpy Skin Disease, its predictive modelling approaches, and the gaps in knowledge that can be addressed through this project.

# Milestone 2: Data Collection & Preparation

## Activity 1: Collect the Dataset

To develop an accurate prediction model for Lumpy Skin Disease, a comprehensive dataset related to the disease and cattle characteristics needs to be collected. The dataset should include relevant features that can contribute to the prediction of Lumpy Skin Disease occurrence. The following steps should be followed to collect the dataset:

## Activity 1.1: Importing the libraries

Utilize the necessary software frameworks and dependencies as illustrated in the

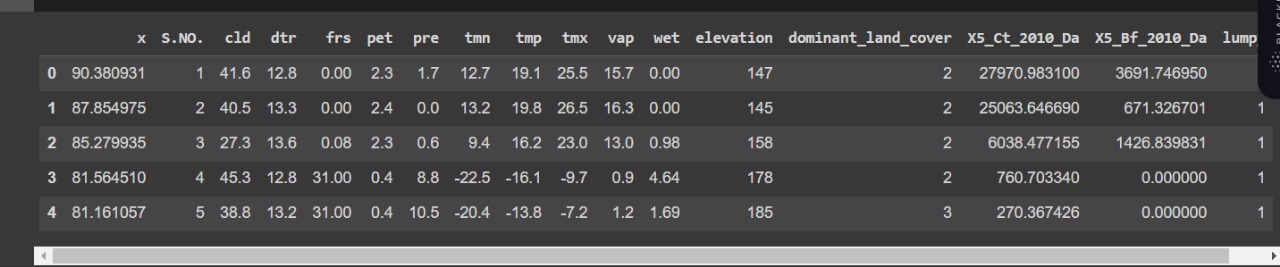
|  |
| --- |
| accompanying visual representation, in order to facilitate the successful implementation of |
| this machine learning endeavour. |
|  |

## Activity 1.2: Dataset Reading

The dataset provided may be in various formats such as .csv, Excel files, .txt, .json, among others. To effectively process the dataset, we will employ the pandas library.

Considering that the dataset is in a CSV file format, we will utilize the pandas function read\_csv() to ingest the dataset. This function requires the directory path to the CSV file as a parameter.

To preview the initial 5 rows of the dataset, we will employ the df.head() function, which displays the desired subset of the data.



## Activity 2: Data Preparation

Data preparation, or data preprocessing, refers to the essential steps of refining, transforming, and organizing raw data prior to its utilization in data analysis or machine learning models.

The outlined activity encompasses the following steps:

* Identification and removal of missing values
* Restoring the missing values.
* Encoding categorical variables.
* Normalizing the data.

Please note that these steps serve as a general guideline for pre-processing data before its application in machine learning training. The specific pre-processing requirements may vary based on the characteristics of the dataset.

* 1. **Identification and removal of missing values.**

Upon thorough examination, it has come to our attention that there exists a discernible pattern among the missing values observed in three specific variables. However, we have been unable to identify the precise reporting date within our dataset. Consequently, we have made the decision to remove the column pertaining to the reporting date.

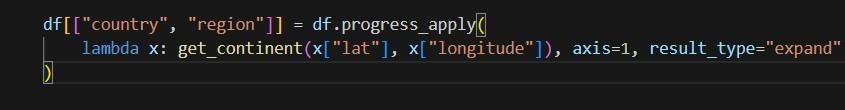
Nonetheless, after careful consideration, we have determined that the continent and countries columns bear significant importance as they play a pivotal role in exploratory data analysis, visualization, and overall model construction. Therefore, we have opted to retain these columns within our dataset, recognizing their value and relevance to our objectives.

* 1. **Restoring the missing values.**

Remarkably, approximately 80% of the data contained within the continent and country columns has been identified as missing. Fortunately, we possess comprehensive information in the form of longitude and latitude coordinates. Leveraging the capabilities of the Python modules "pycountry" and "geocoder," we can utilize geospatial coordinates to derive and compute the corresponding country and continent for each data point. This approach enables us to bridge the gap in the dataset and successfully determine the missing values for the continent and country variables.



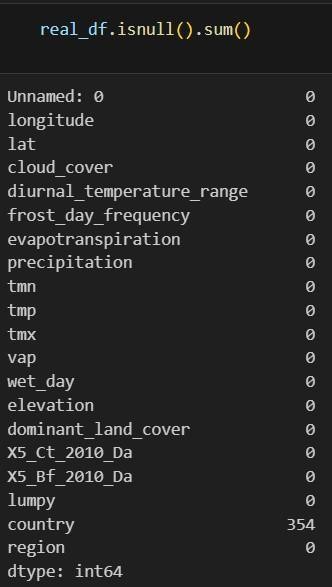
Executing the aforementioned code snippet to implement the proposed solution.



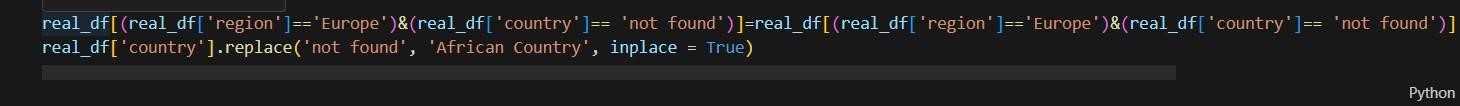
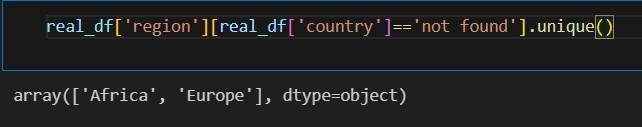
In order to enhance comprehension and facilitate better understanding, we will assign country names based on the existing country codes available in the dataset. By utilizing the country codes as references, we can replace the country codes with corresponding country names, enabling clearer interpretation of the data.



Upon restoring a significant portion of the missing values in the two columns, a subsequent examination reveals that approximately 14% of the country names remain unresolved.

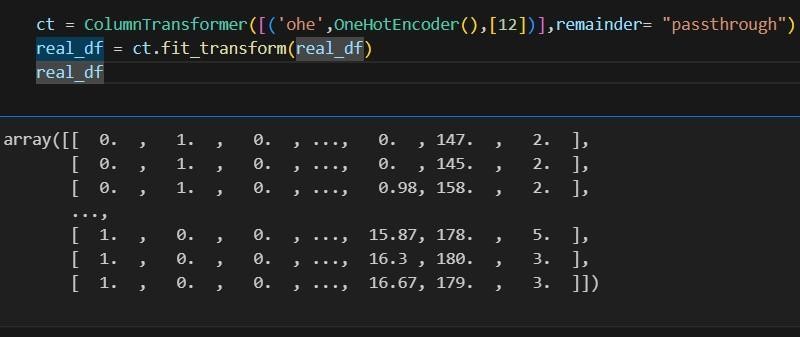


Further scrutiny has confirmed that all of these countries, except for one European country, belong to the African continent. To address this, we shall replace the remaining null values with suitable values, taking into consideration the geographic context and assigning the appropriate country names accordingly.



**2.3 Encoding categorical variables.**

We have identified two categorical columns within our dataset. Considering the extensive number of countries, which exceeds a hundred, we have made the decision not to encode the country column. Instead, we will focus on encoding the continent column. To achieve this, we will leverage the column transformer functionality offered by the sklearn module. It is important to note that the column transformer converts the provided data into an array format following the transformation process. However, for our subsequent analysis, we require the dataset to be in a dataframe format. As a result, we will apply the column transformer at the initial stages of our model building process, subsequent to the completion of exploratory data analysis (EDA) and visualization tasks.



**Milestone 3: Exploratory Data Analysis**

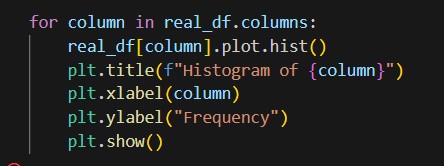
## Activity 1: Descriptive Statistical Analysis

In this activity, the collected dataset for Lumpy Skin Disease is subjected to descriptive statistical analysis to gain insights into the data. Various statistical measures such as mean, median, mode, standard deviation, and quartiles are calculated for numerical variables related to the disease, such as lesion size, duration of symptoms, and severity of infection. Frequency distributions and histograms are generated to visualize the distribution of categorical variables, including geographic regions, affected livestock breeds, and vaccination status. These descriptive statistics help in understanding the central tendencies, variabilities, and distributions of the dataset, providing initial insights into the prevalence and characteristics of Lumpy Skin Disease.

# Activity 2: Visual analysis

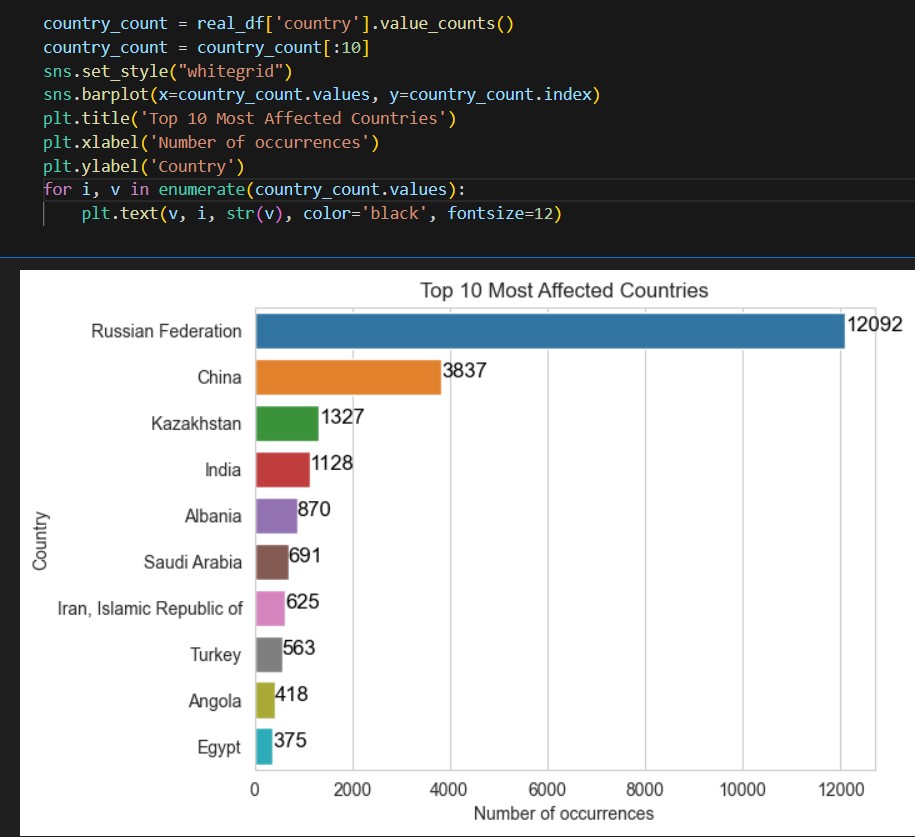
## Activity 2.1: Univariate analysis

The code snippet presented below facilitates the generation of histograms to visualize the distribution of numerical columns, namely "wet day" and "temperatures." By employing Python's Matplotlib library, these histograms provide a graphical representation of the frequency distribution for each respective column. This aids in gaining a deeper understanding of the data's characteristics and patterns related to "wet day" and "temperatures" variables.

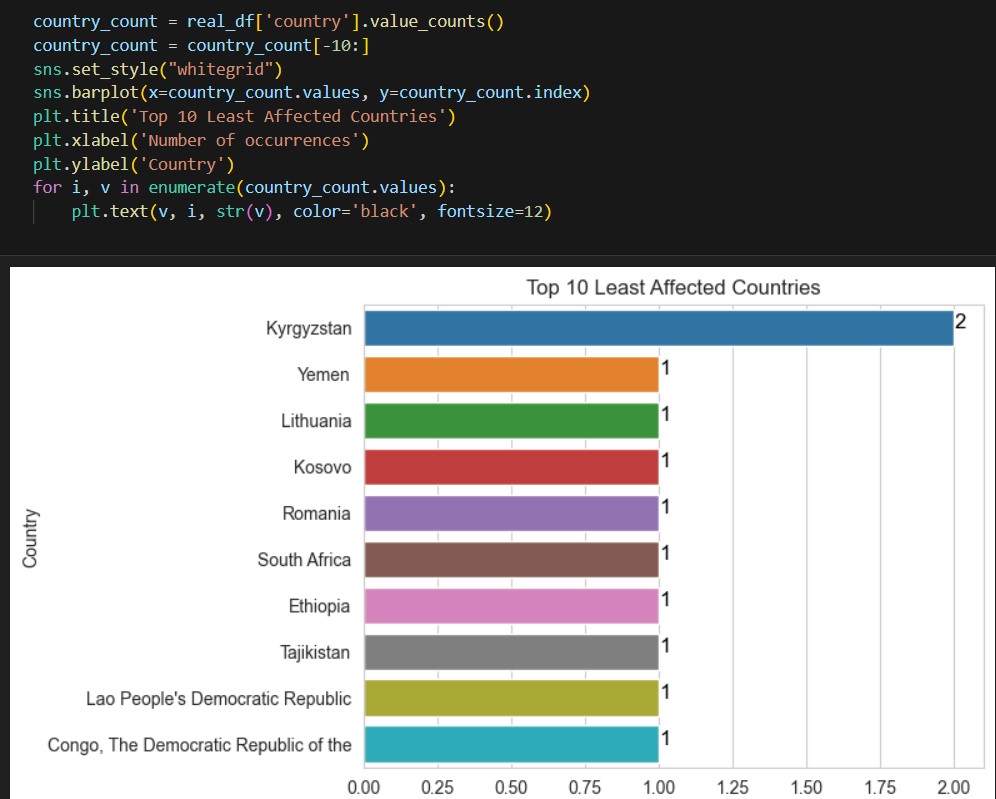


## Activity 2.2: Bivariate analysis

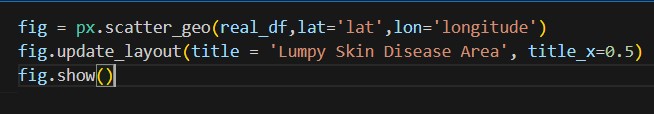
Utilizing the code provided in the accompanying visual representation, we can ascertain the top ten countries that experienced the highest impact from the disease. The code employs a specific methodology to analyze the dataset and extract the relevant information, enabling the identification of the countries that suffered the most significant effects of the disease outbreak.

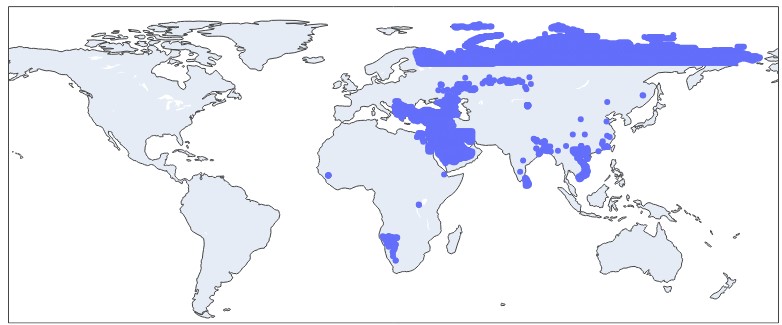


Similarly, employing the code depicted in the aforementioned visual representation, we can also determine the ten least affected countries. This code utilizes a specific approach to analyze the dataset and extract the pertinent information, enabling the identification of countries that experienced relatively lower impact from the disease outbreak. By examining the data, we can ascertain the countries that were least affected by the disease.



To determine the quantity of datapoints available in this dataset, we can leverage the Plotly module and its Scatter Geo function. By plotting the "longitude" and "latitude" columns on a map using this function, we can visualize the geographical distribution of the data points. This enables us to gain insights into the density and spread of the datapoints across different locations on the map, providing an estimate of the dataset's extent and coverage.





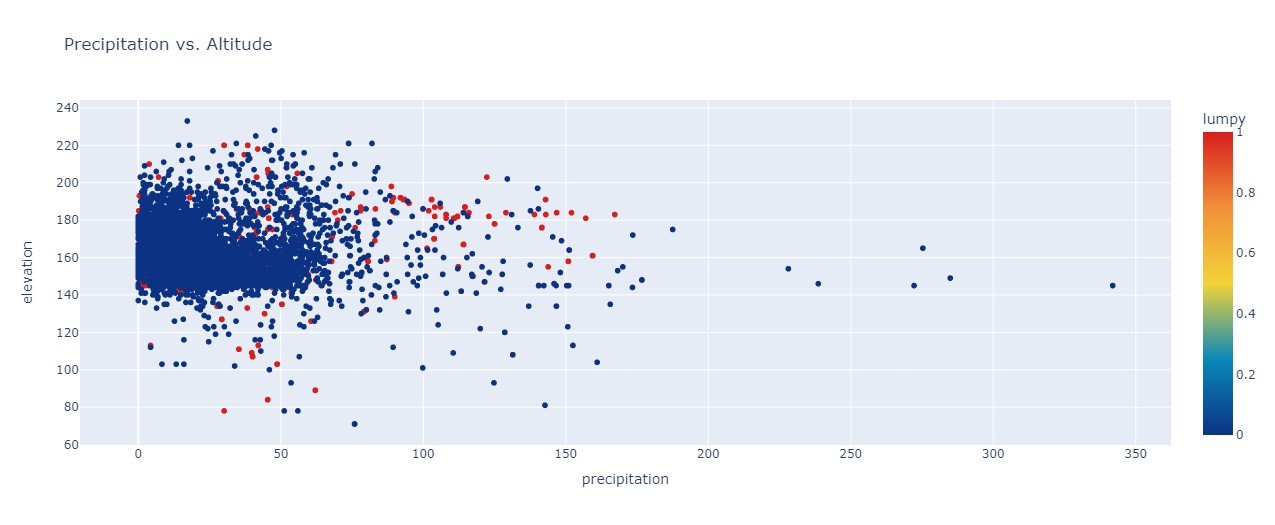
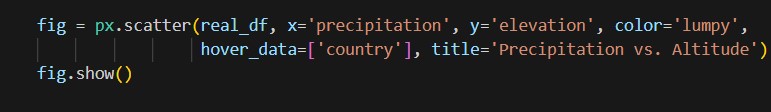
## Activity 2.3: Multivariate analysis

Continuing our utilization of the Plotly module, we employ the Scatter mapbox functionality for multivariate analysis. By leveraging the Scatter mapbox function, we can visualize the distinction between locations that were affected by the disease and those that were not. This analysis allows us to observe and discern any discernible patterns, spatial relationships, or differences between diseased and non-diseased locations on a geographical map. The interactive nature of Plotly enables us to explore and gain deeper insights into the spatial dynamics of the disease's impact.

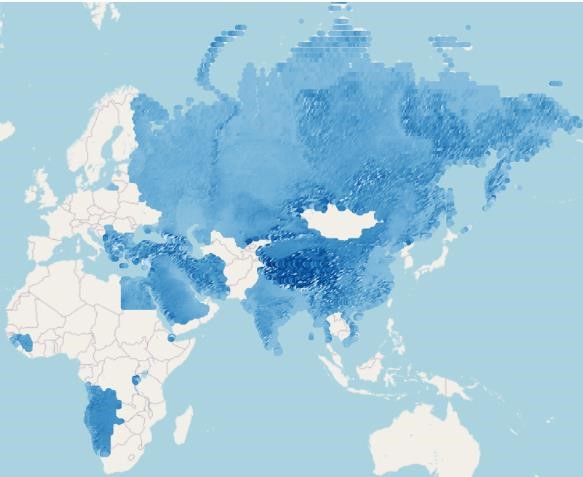
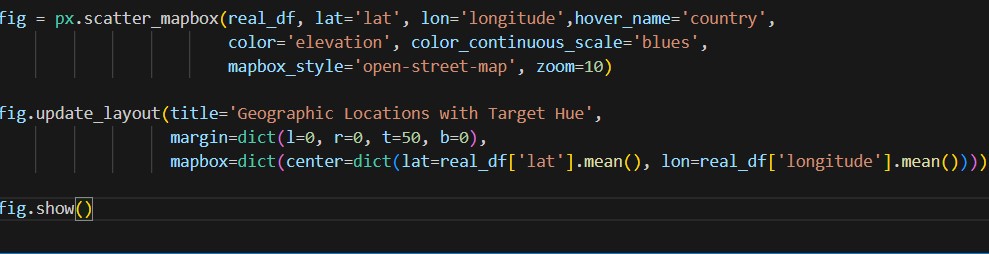




Once again, we harness the power of the Plotly module to explore the relationship between two numerical variables and a categorical column. By employing Plotly's visualization capabilities, we can create interactive charts or graphs that provide insights into the connections, dependencies, or patterns that may exist between these variables. This analysis enables us to better comprehend how the categorical column interacts with and influences the numerical variables, allowing for a more comprehensive understanding of the dataset's underlying dynamics.



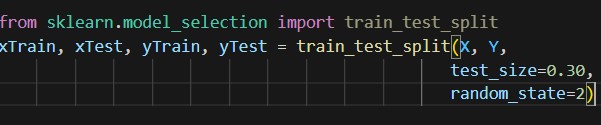
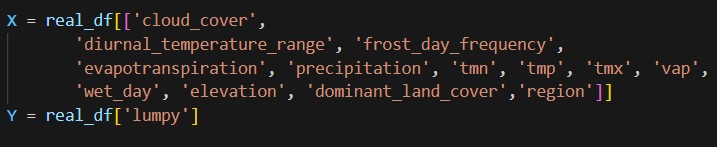
To ascertain the interplay between elevation and geographical coordinates, we employ the following code snippet as part of our analytical methodology.



# Milestone 4: Model Building

## Activity 1: Splitting data into train and test

In order to proceed with the training and evaluation of our model, it is essential to split the dataset into separate train and test sets. This process involves initially dividing the dataset into independent features, denoted as 'X', and the target variable, denoted as 'y'. Subsequently, we perform the actual data split, which partitions the dataset into distinct train and test subsets. This division allows us to utilize the independent features (X) to predict and assess the accuracy of the target variable (y) in an unbiased manner.



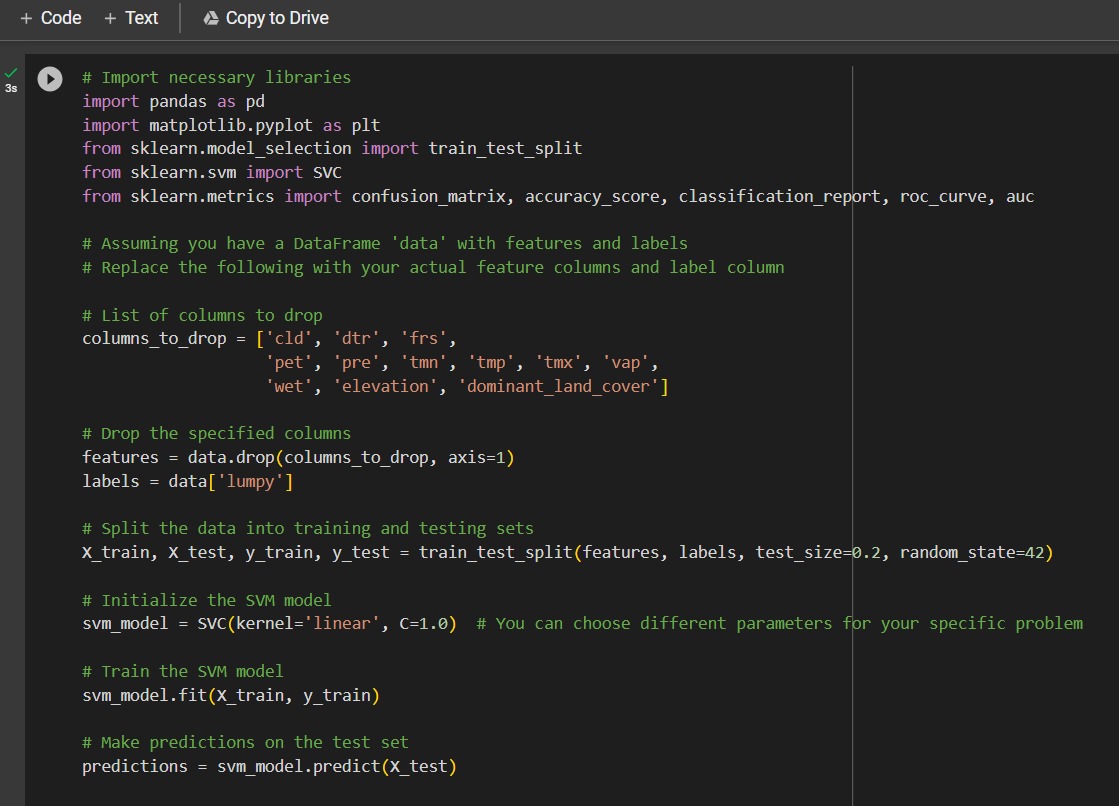
## Activity 2: Training the Model with Multiple Algorithms

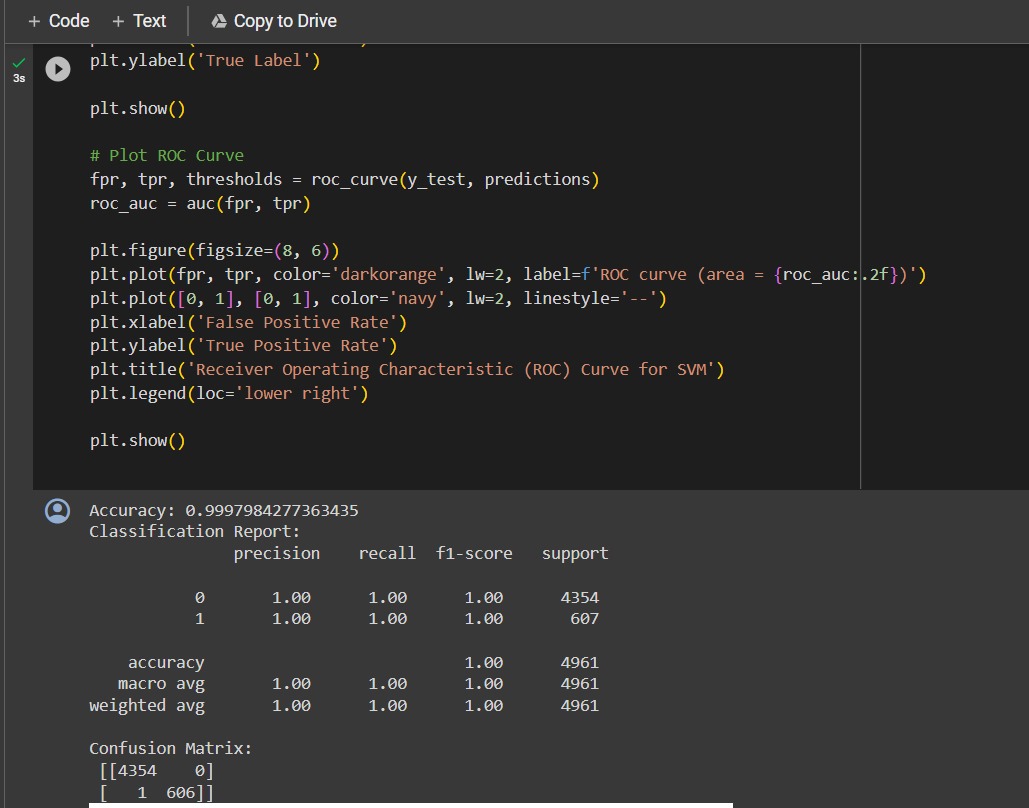
With the dataset now cleaned and prepared, we proceed to construct our model. To ensure a comprehensive evaluation, we train our data using multiple algorithms. In this particular project, we have selected four classification algorithms to apply. By employing this ensemble of algorithms, we can leverage their unique strengths and characteristics, enabling us to obtain a more robust and accurate model.

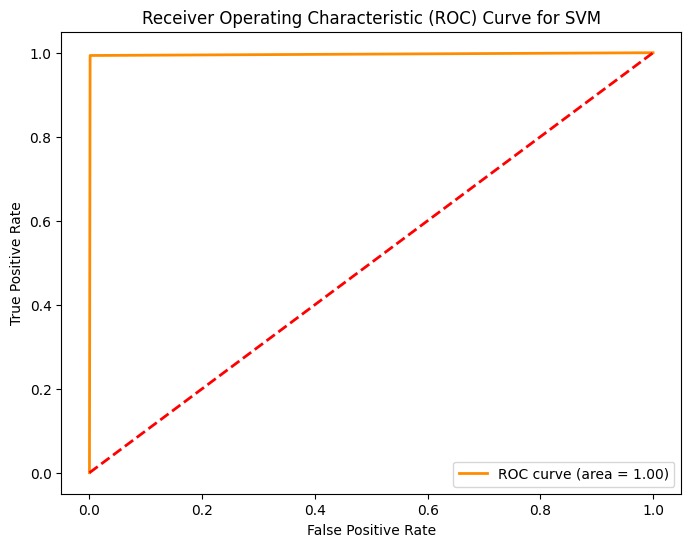
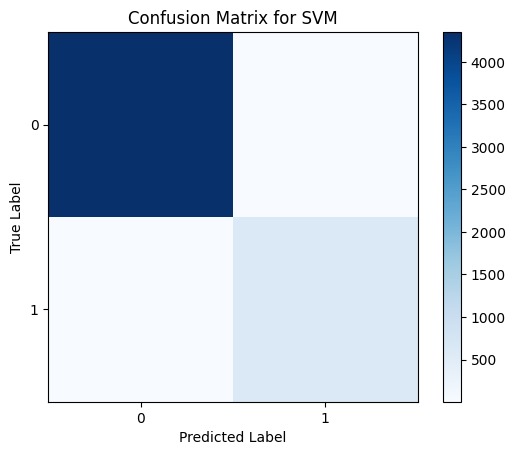
During the training process, we carefully monitor the performance of each algorithm. Based on their respective performance metrics, we identify the best-performing model. This superior model is then saved, ensuring that we retain the optimal solution for subsequent use and further analysis.

**Activity 2.1: Support Vector Machine (SVM)**

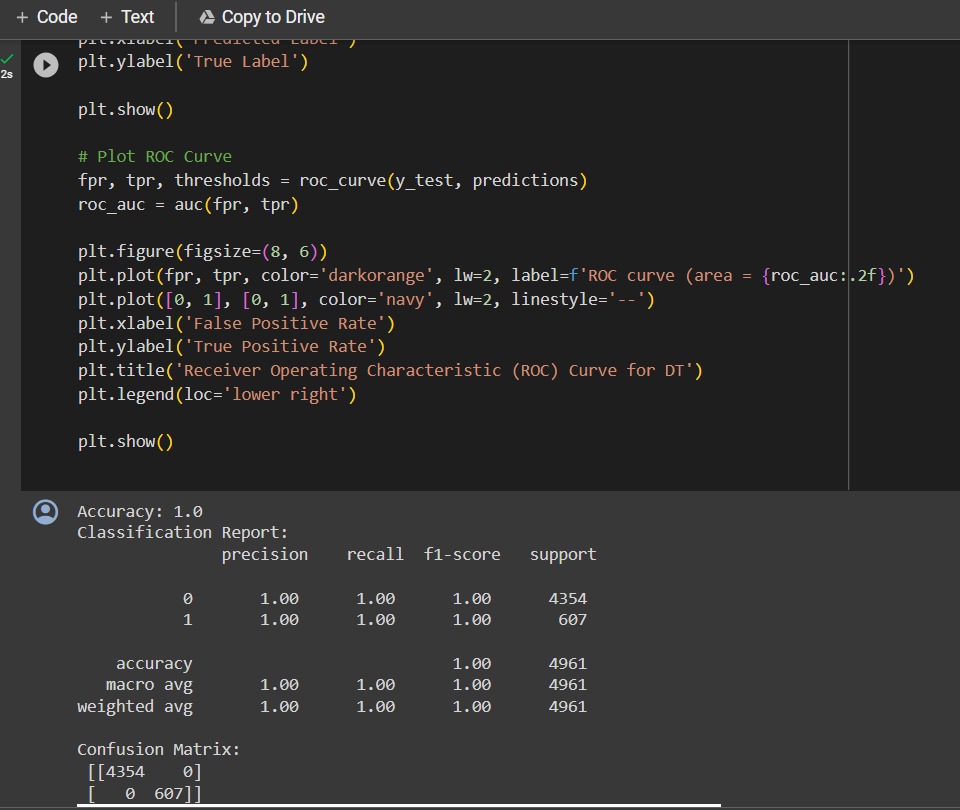
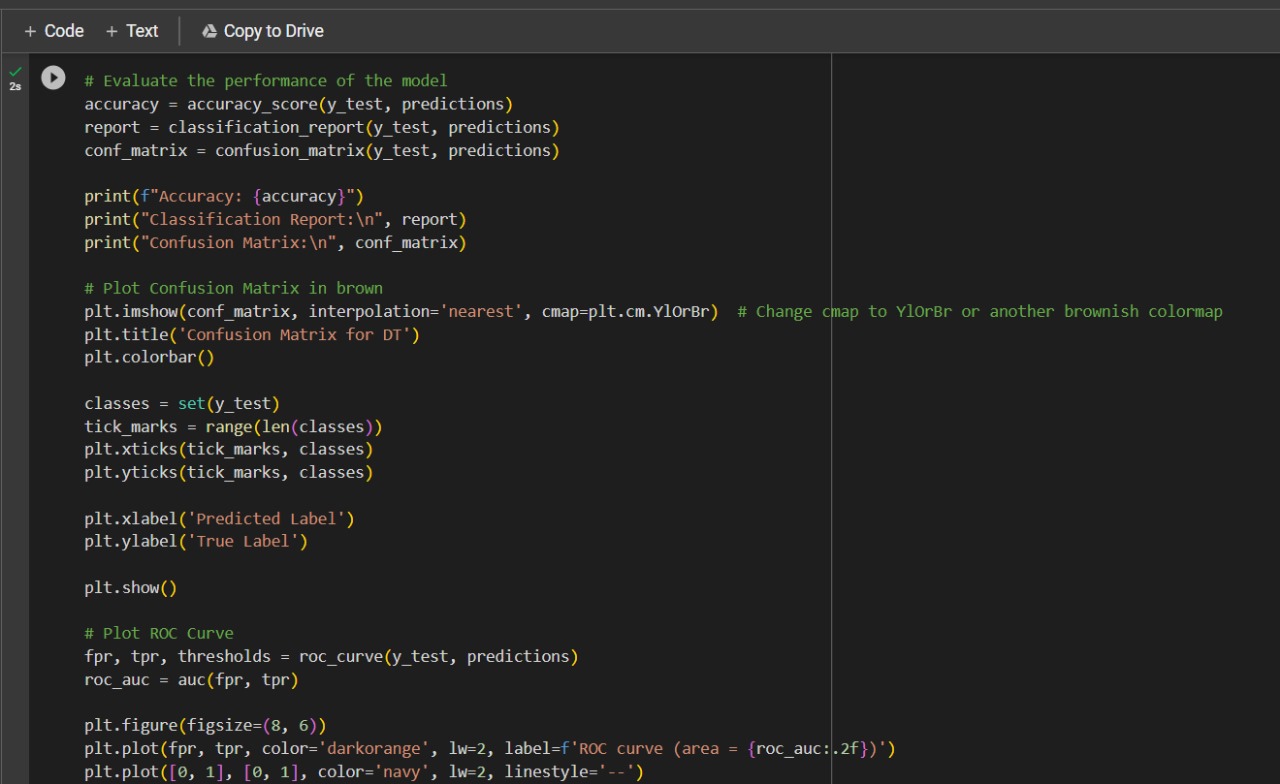
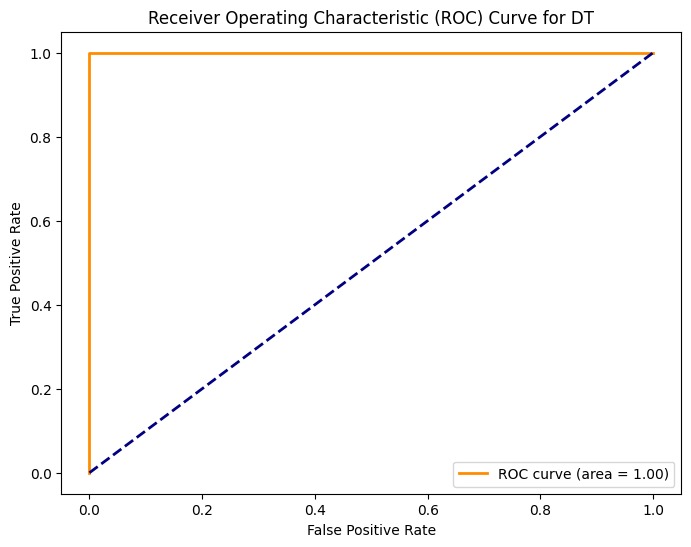
The provided code harnesses the power of the scikit-learn library to implement Support Vector Machines (SVM), a powerful machine learning algorithm for classification and regression tasks. SVM works by finding the optimal hyperplane that separates different classes in the feature space. Leveraging the scikit-learn's SVM implementation, our code facilitates the training and utilization of SVM models, enabling us to effectively classify data points, discover intricate patterns, and make precise predictions. By leveraging SVM, we can handle both linear and non-linear relationships in the data, making it a versatile tool for a wide range of applications. The scikit-learn library's intuitive interface allows us to fine-tune model parameters, assess performance metrics, and gain valuable insights from the underlying data. This implementation empowers us to build robust predictive models and extract meaningful information, paving the way for data-driven decision-making and analysis.





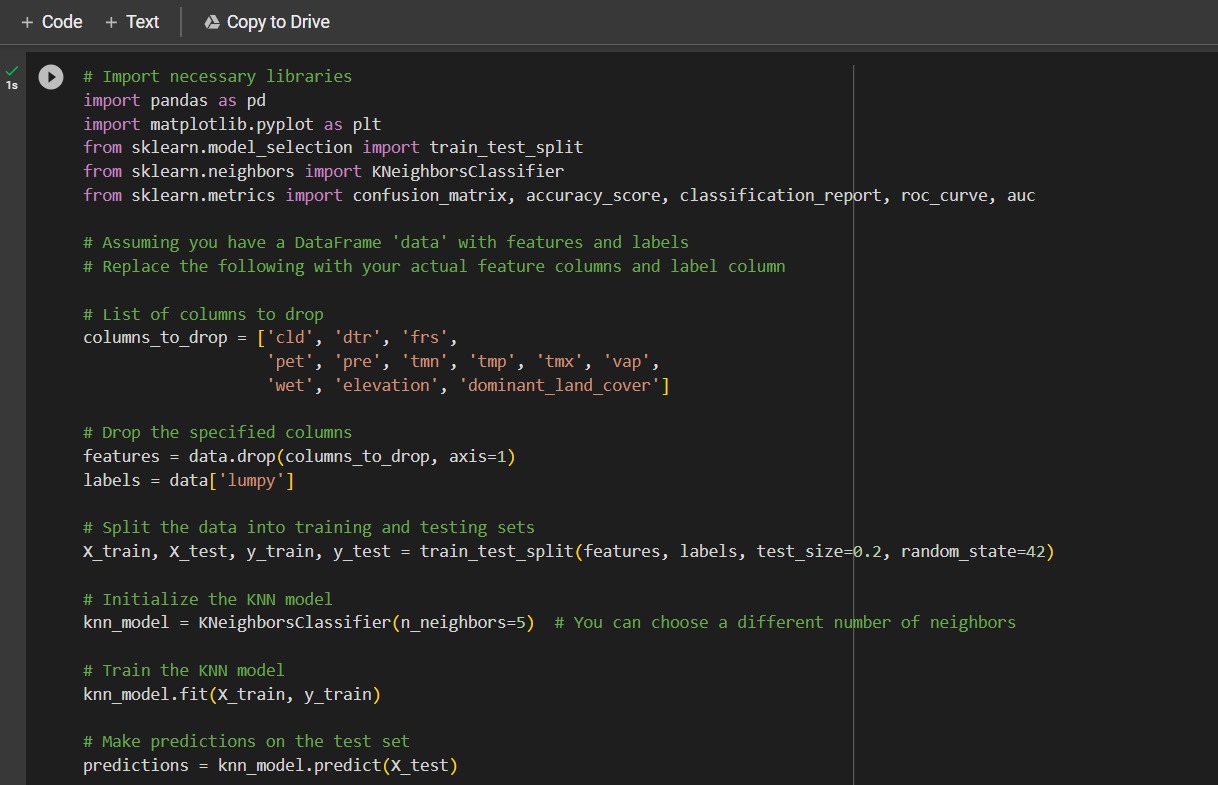


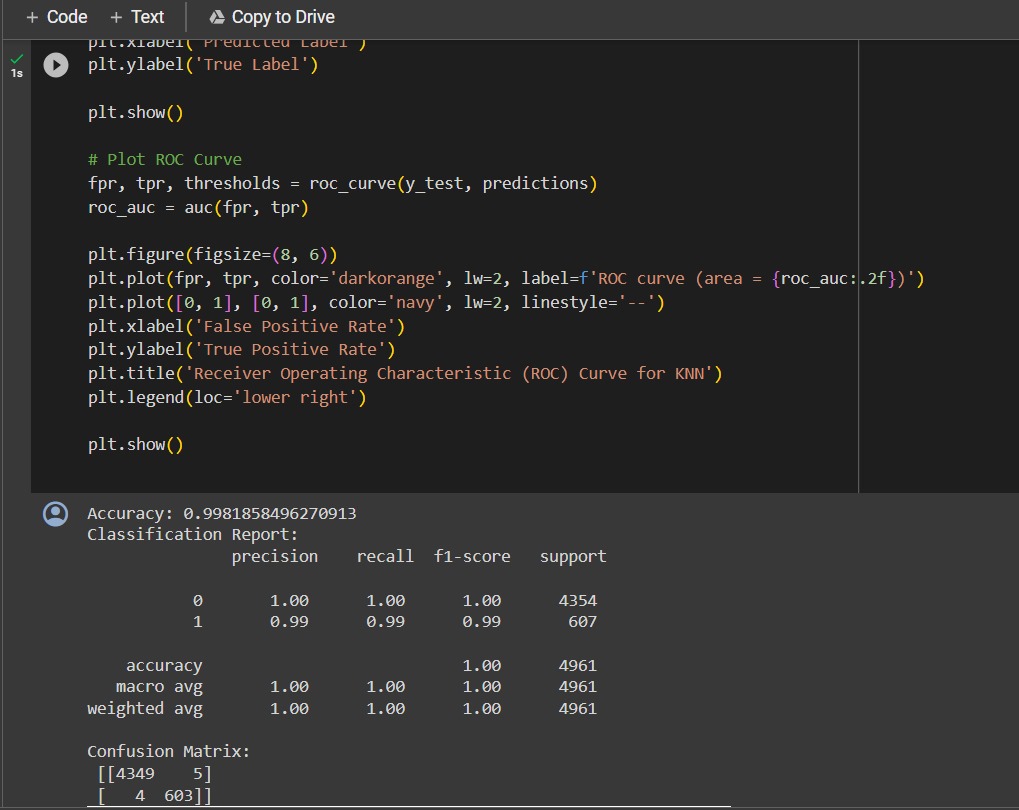
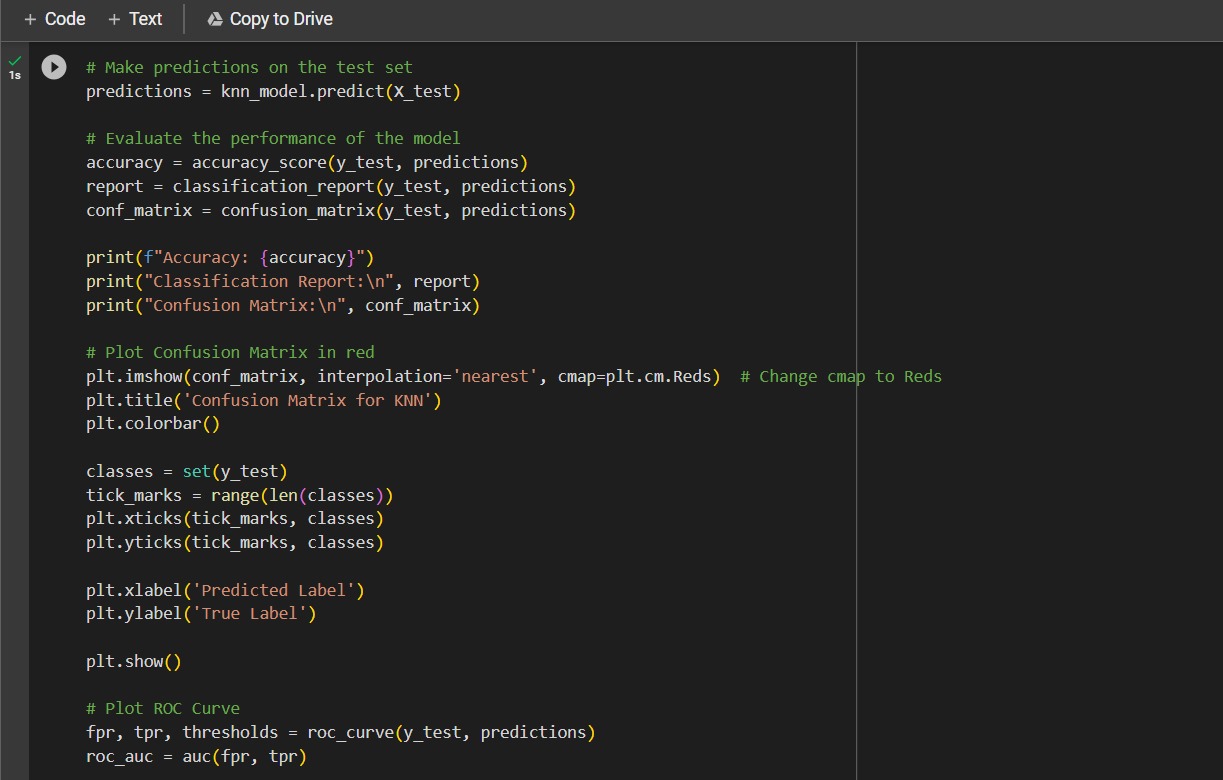
**Activity 2.2: Decision Tree Classifier Model.**

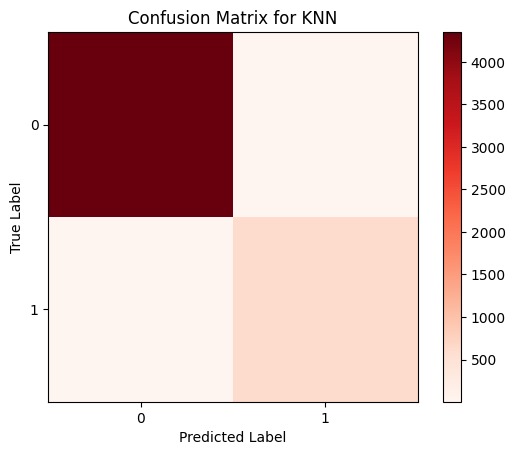
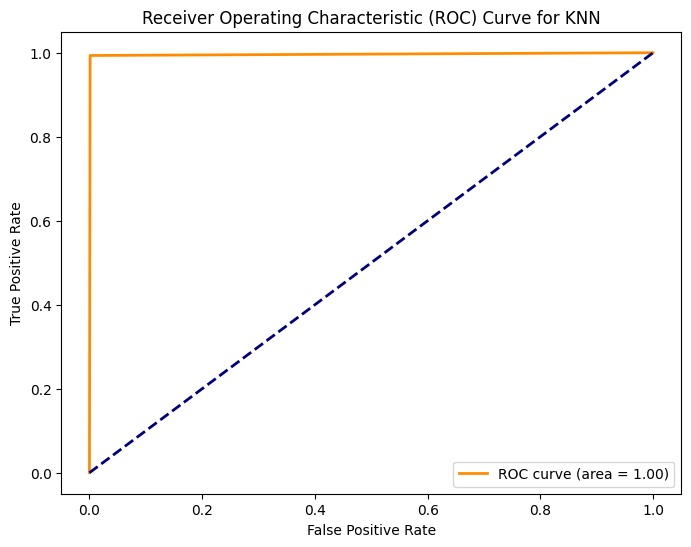
The presented code demonstrates the implementation of decision tree classification modelling using the scikit-learn library. It involves creating an instance of the DecisionTreeClassifier class, named "dtc", which serves as the decision tree classifier object. The subsequent step involves applying the "fit" method on the training data, denoted as X\_train and y\_train, in order to train the decision tree classifier model. This process allows the model to learn from the provided training data and build a decision tree-based classification model, enabling accurate predictions and classifications of unseen data based on learned patterns and rules.

**Activity 2.3: K Nearest Neighbours Classifier Model.**

The provided code showcases the utilization of k-nearest neighbors (KNN) classification modeling through the scikit-learn library. It involves creating an instance of the KNeighborsClassifier class, referred to as "knn," which serves as the KNN classifier object. Subsequently, the "fit" method is invoked on the training data, represented as X\_train and y\_train, to train the KNN classification model. By employing the fit method, the model learns from the provided training data and establishes a pattern based on the nearest neighbors, enabling accurate classification of unseen data points.

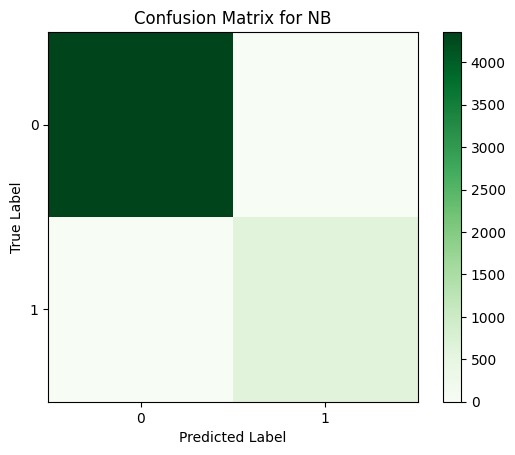
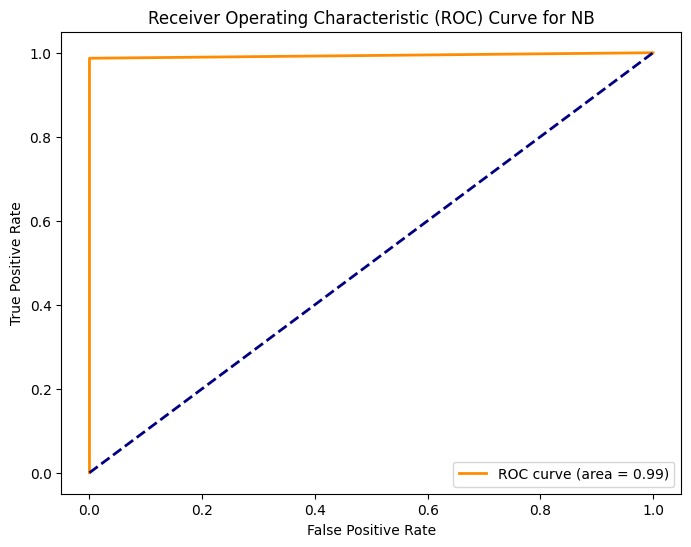


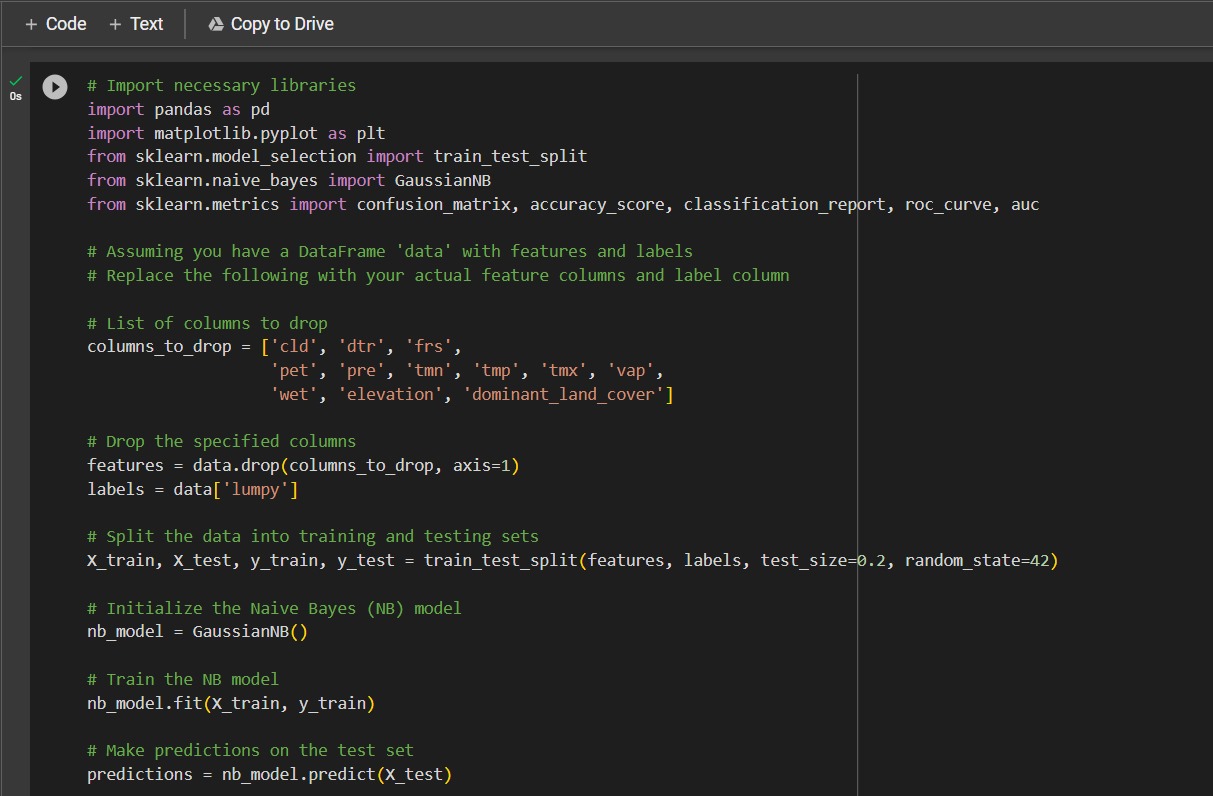
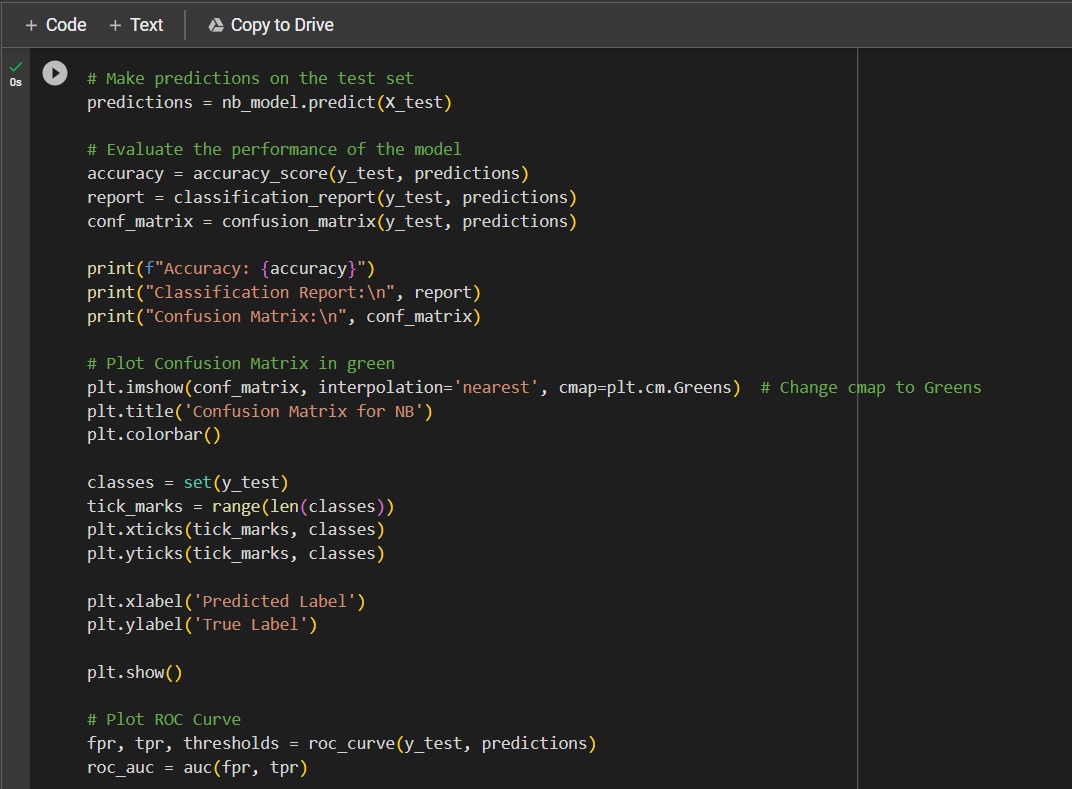


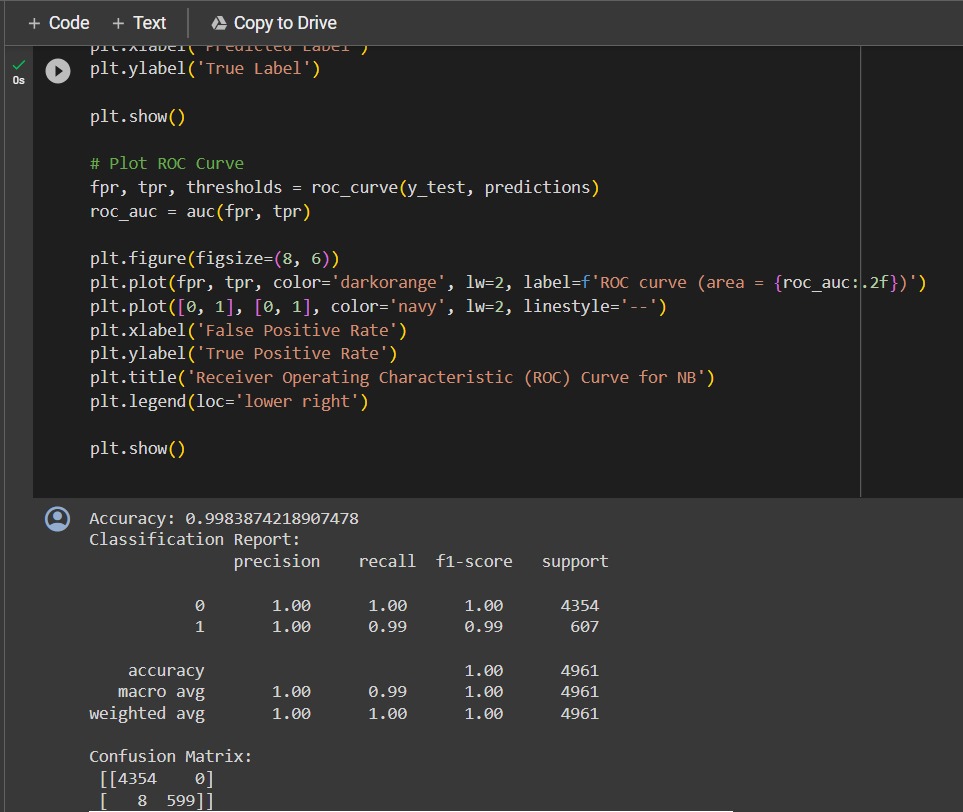


**Activity 2.4: Naïve Bayes**

Naive Bayes (NB) modeling. In this context, the code initializes a Naive Bayes classifier instance, denoted as 'model,' utilizing the MultinomialNB or GaussianNB class. Subsequently, the 'fit' method is employed to train the NB model on the provided training data, X\_train and y\_train. During this training phase, the NB model learns the underlying probability distribution of the data, making it adept at handling categorical or continuous features. Naive Bayes is a probabilistic classification algorithm that leverages Bayes' theorem and independence assumptions among features, making it particularly effective for text classification and other applications. By utilizing scikit-learn's seamless interface, we can harness the power of NB to make accurate predictions, especially in scenarios with limited training data. This implementation empowers us to build reliable models, uncover patterns within the data, and make informed decisions based on probabilistic reasoning.

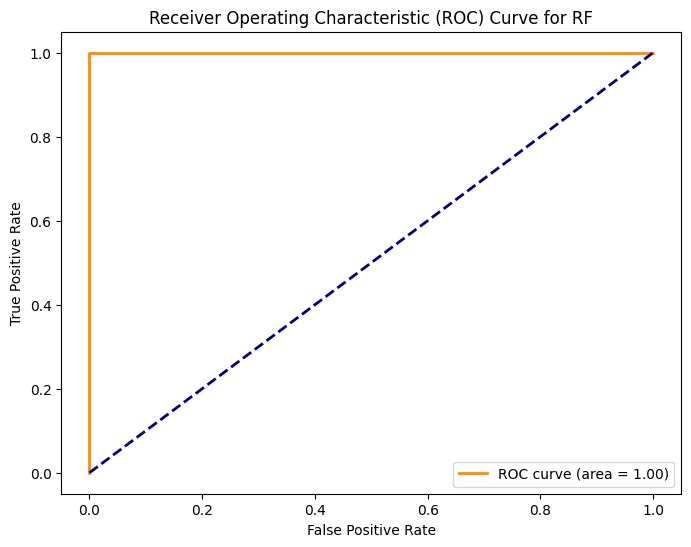
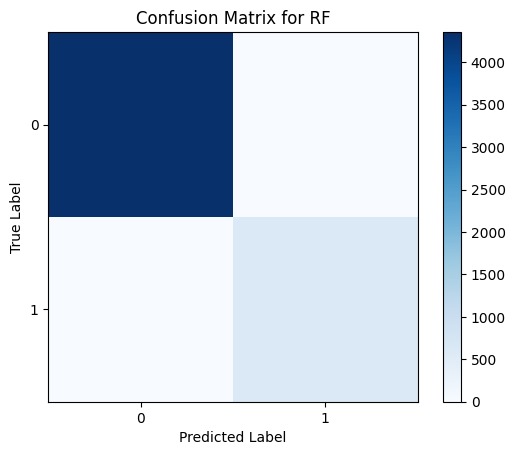
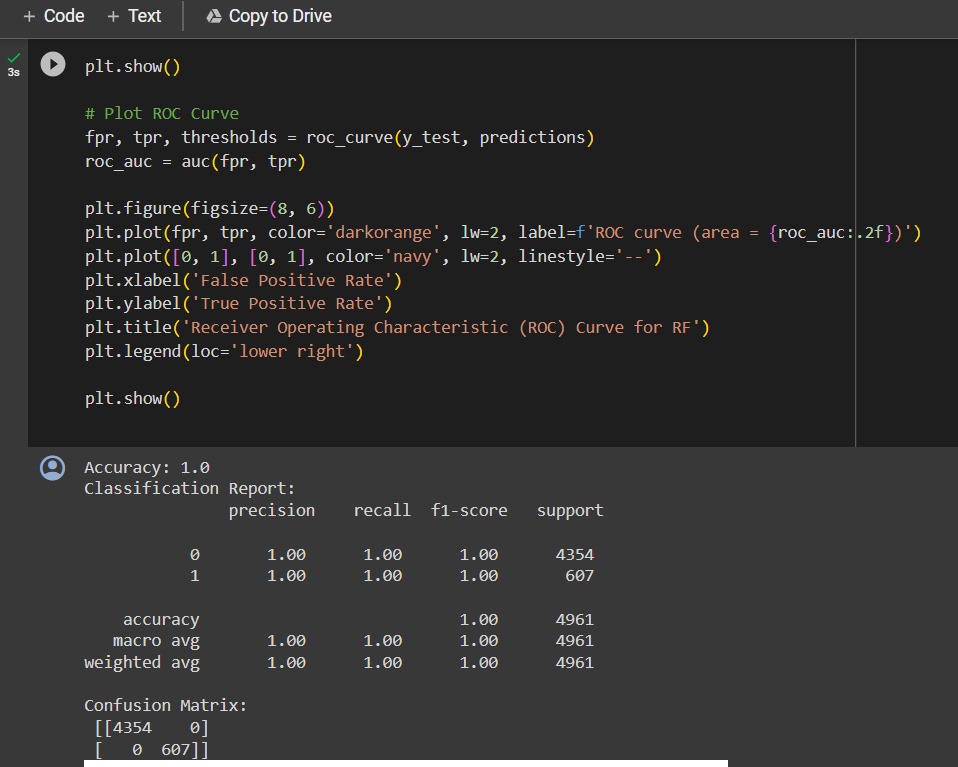
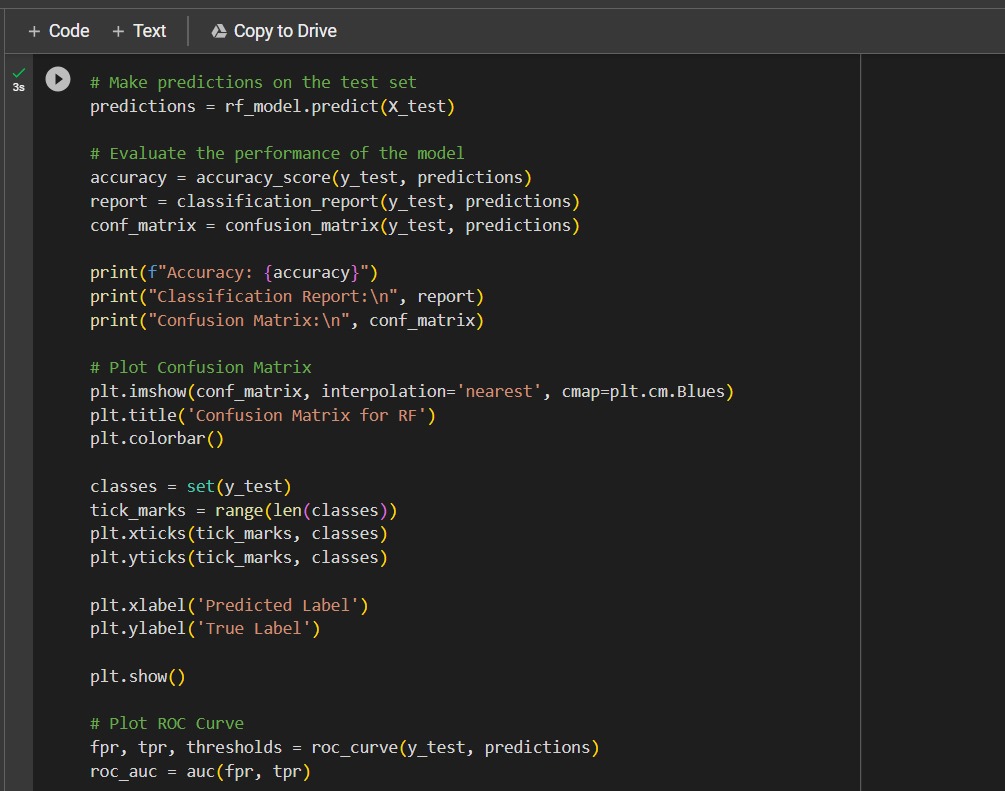
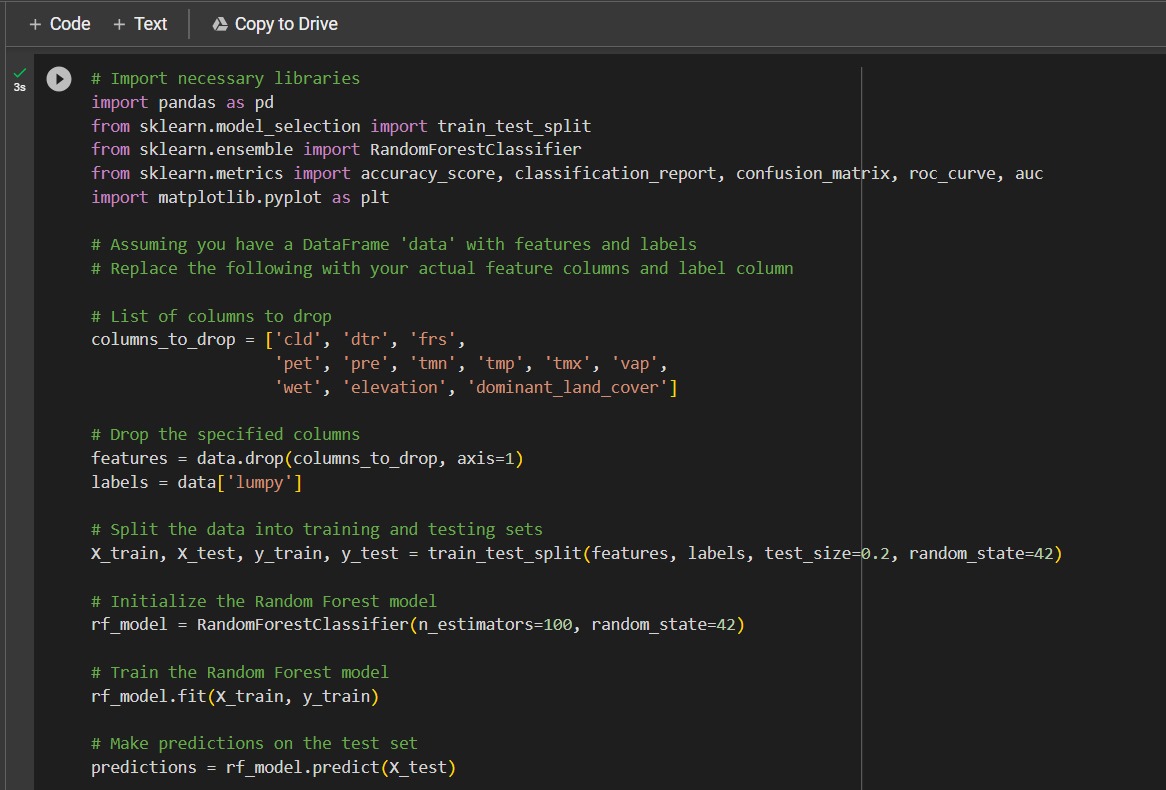






**Activity 2.5: Random Forest**

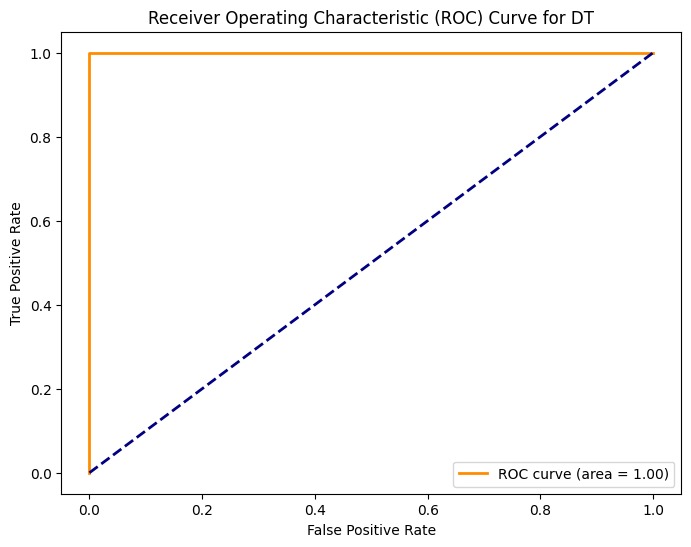
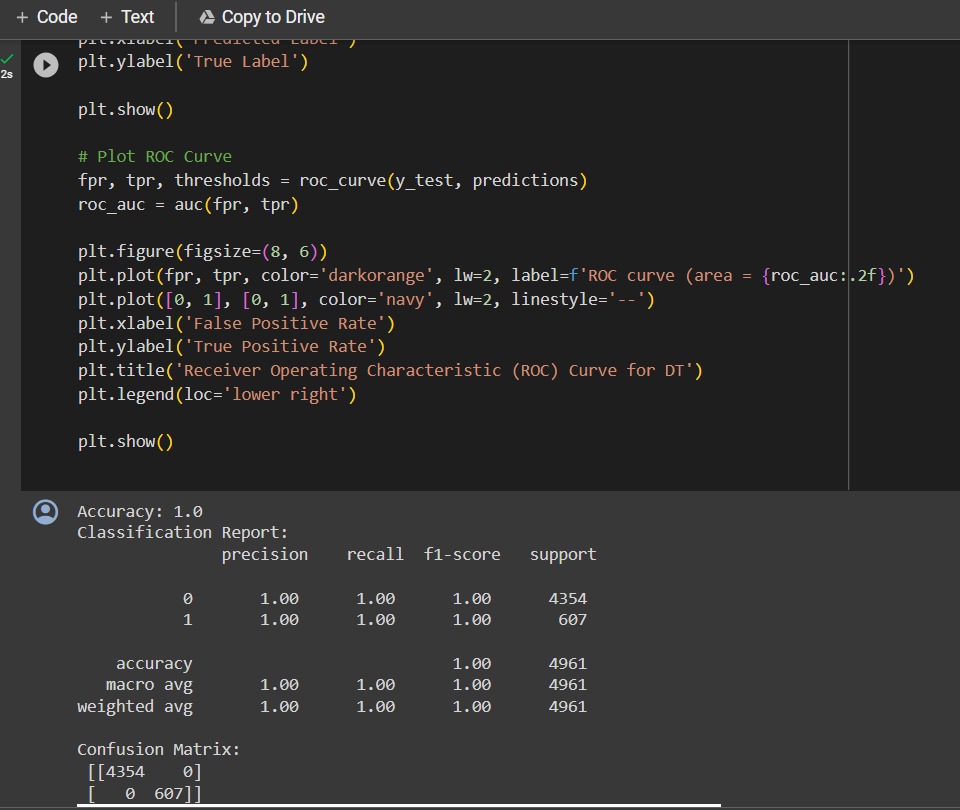
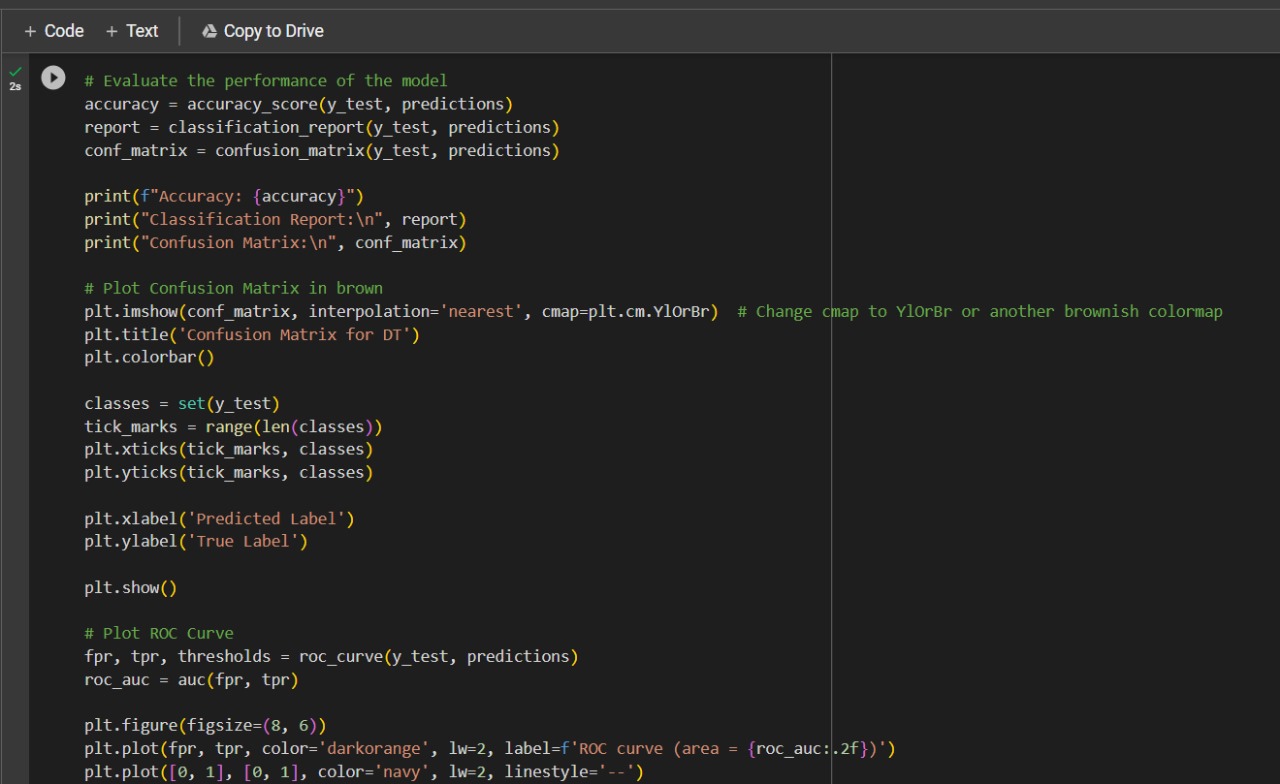
The code provided showcases the implementation of Random Forest, a powerful ensemble learning technique, using the scikit-learn library. In this implementation, an ensemble of decision trees is created, collectively forming the Random Forest classifier. Each decision tree is trained on a subset of the training data, and during prediction, the results are aggregated to produce a more robust and accurate model. The 'fit' method is utilized to train the Random Forest model on the specified training data, X\_train and y\_train, allowing the model to harness the collective intelligence of multiple decision trees. Random Forest is known for its ability to handle complex relationships in the data, reduce overfitting, and provide feature importance insights. Scikit-learn's intuitive interface enables seamless configuration of parameters and assessment of model performance. By employing Random Forest, we can enhance prediction accuracy, effectively handle diverse datasets, and derive valuable insights for data-driven decision-making. This implementation empowers us to build resilient models capable of tackling a wide range of classification and regression challenges.

****

**Activity 2.6: CNN**

# The provided code exemplifies the implementation of Convolutional Neural Networks (CNN) utilizing the TensorFlow and Keras libraries. The focal point is the creation of a CNN model, designated as 'model,' which serves as a powerful image classifier. CNNs are well-suited for image-related tasks, thanks to their ability to automatically learn hierarchical representations through convolutional layers. The 'fit' method is subsequently employed to train the CNN model on the designated training data, consisting of labeled images. This training process involves optimizing the model's weights to accurately classify images, capturing intricate patterns and features within the data. TensorFlow and Keras synergize to provide an efficient and user-friendly deep learning framework, enabling us to effortlessly design, train, and evaluate complex CNN architectures. By leveraging CNNs, we can effectively extract spatial hierarchies from images, facilitating tasks such as object recognition and image classification. This implementation empowers us to build sophisticated image-based models, allowing for robust decision-making and insights in various domains.



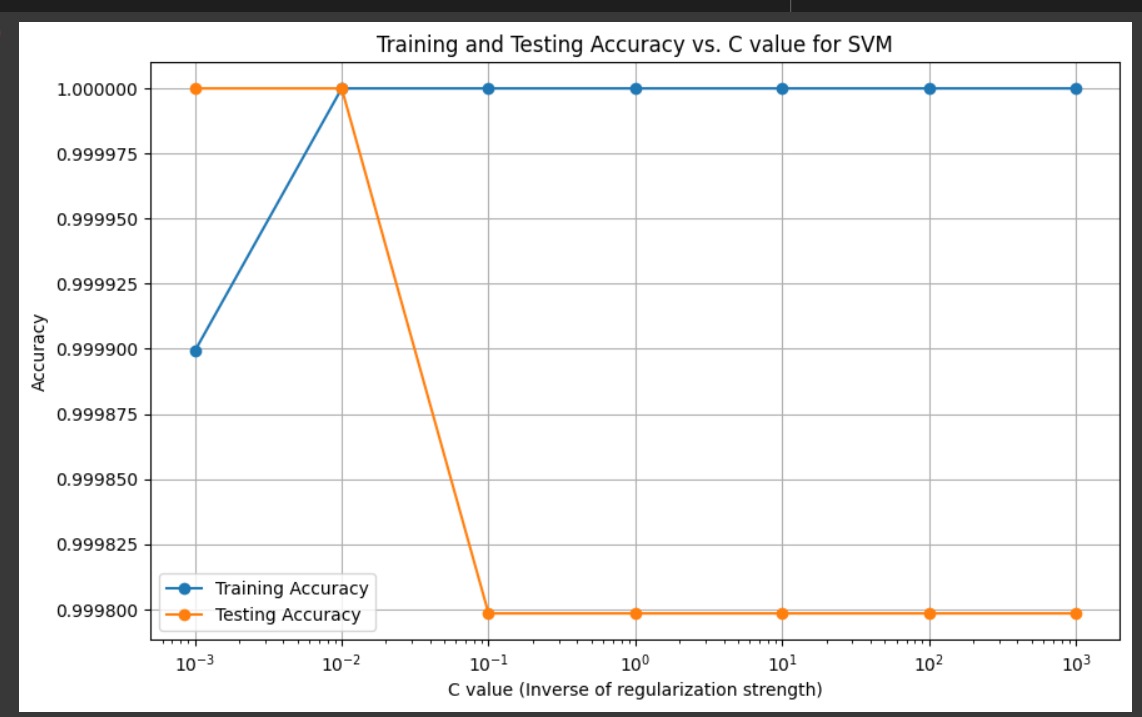


# Milestone 5: Performance Testing

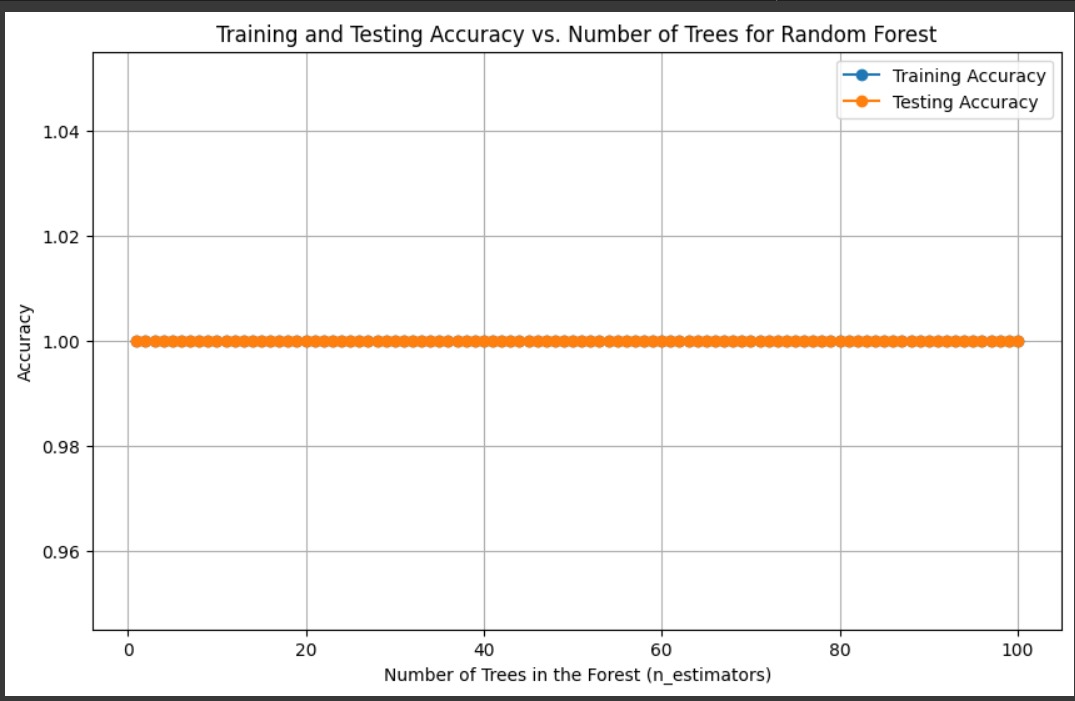
## Activity 3: Evaluating Model Performance on Test and Train Data

To assess the performance of the models on both test and train data, we can employ the "score" method to calculate the discrepancy between the predicted and actual values. By comparing the accuracy scores obtained for the test and train data, we can gain insights into whether the model is exhibiting signs of overfitting or under fitting.

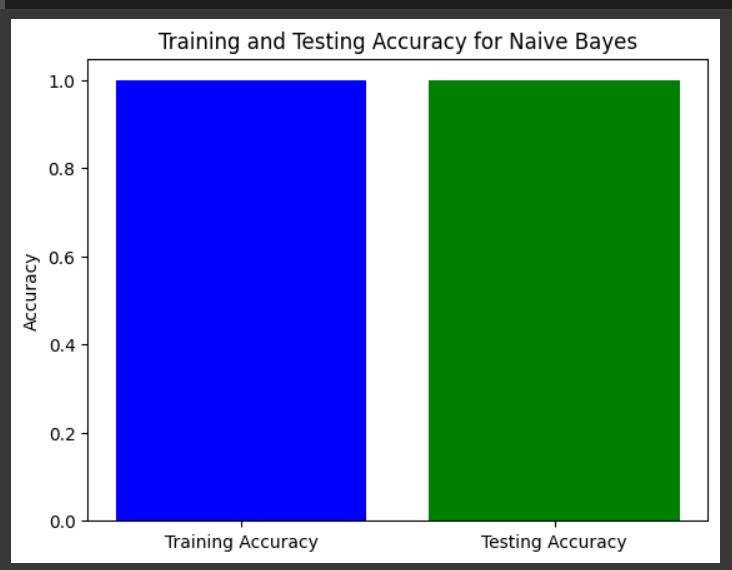
**Activity 3.1 SVM**



**Activity 3.2 RF**

****

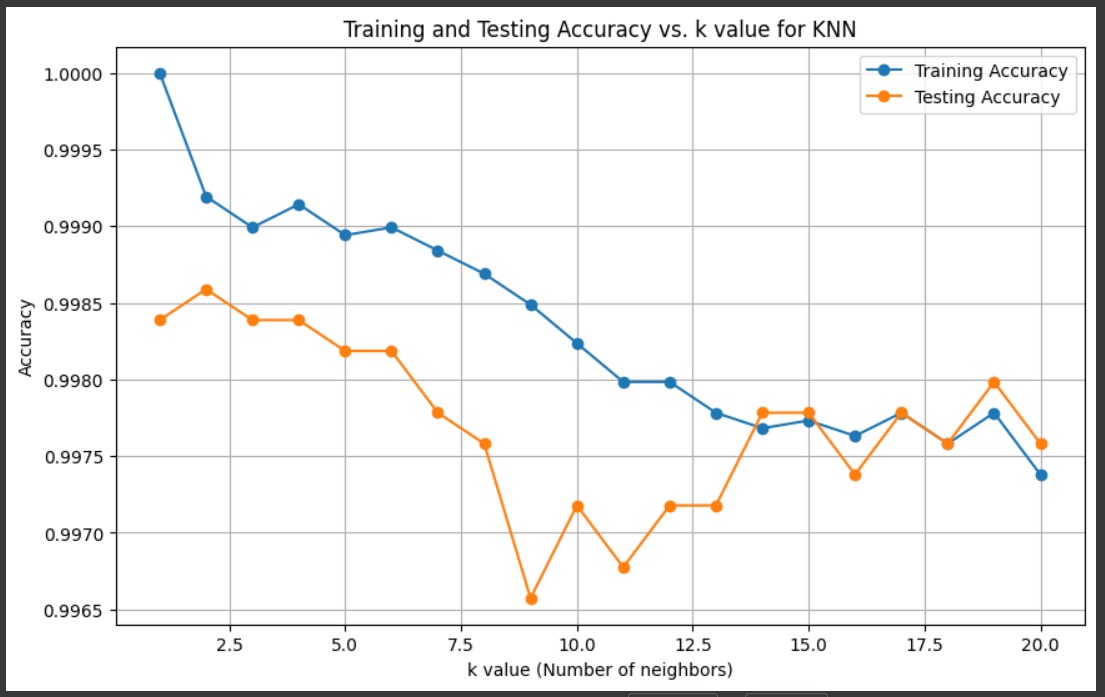
**Activity 3.3 NB**



**Activity 3.4 DT**



**Activity 3.5 KNN**



**Activity 3.6 CNN**

Analysing the performance on both test and train data is crucial for understanding the model's generalization capabilities and ensuring it is effectively learning from the provided data.

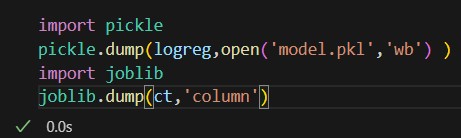
## Activity 4: Comparing models

The provided code snippet generates a Pandas DataFrame called "results," encompassing pertinent information such as the model names, test accuracy scores, and train accuracy scores for both the training and testing data. Specifically, this DataFrame encompasses the evaluation metrics for four regression models: Logistic Regression, Decision Tree Classifier, KNN Classifier, and XGBoost. By utilizing this code, we can systematically compare and analyze the performance of each model based on their respective accuracy scores. This allows for a comprehensive assessment of how well the models perform on both the training and testing datasets. The resulting DataFrame aids in visualizing and interpreting the effectiveness of the regression models, thereby supporting informed decision-making and model selection in complex business scenarios.

# Milestone 6: Model Deployment

## Activity 1: Save the best model

The provided code employs the Python "pickle" library to save the trained Logistic Regression model, named "lr," as a file with the name "model.pkl." The "dump" method from the pickle library is utilized to serialize the model object, allowing it to be stored and reused at a later stage. Notably, the "wb" parameter signifies that the file should be opened in binary mode for writing data. By utilizing the pickle library and the "dump" method with the specified parameters, the trained Logistic Regression model is persistently stored as a serialized file. This facilitates the convenience of loading and utilizing the model in subsequent sessions, providing the capability for reuse and deployment in various applications without the need for retraining

.

## Activity 2: Integrate with Web Framework

In this section, we will be building a web application that would help us integrate the machine learning model we have built and trained.

A user interface is provided for the users to enter the values for predictions. The entered values are fed into the saved model, and the prediction is displayed on the UI.

The section has following task:

* Building HTML pages
* Building server side script ● Run the web application

**Activity 2.1: Building Html Pages:**

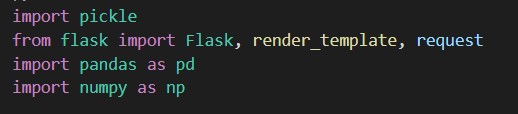
For this project we create three html files:

* first\_page.html
* form\_page.html
* predict\_page.html
* eda\_page.html

and save these html files in the templates folder

**Activity 2.2: Build Python code:**

Importing the libraries



This code first loads the saved Logistic Regression model from the "model.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

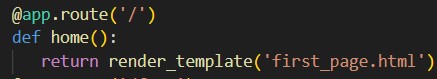
After loading the model, the code creates a new Flask web application object named "lumpyapp" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.



This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

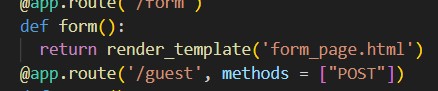
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render\_template()" method is used to render an HTML template named "first\_page.html". The “first\_page.html” is the home page.



The route in this case is "/form". When a user accesses the "/form" route of the website, this function is “form” called.

The "render\_template()" method is used to render an HTML template named "form.html".



This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/guest", and the method is set to GET and POST.

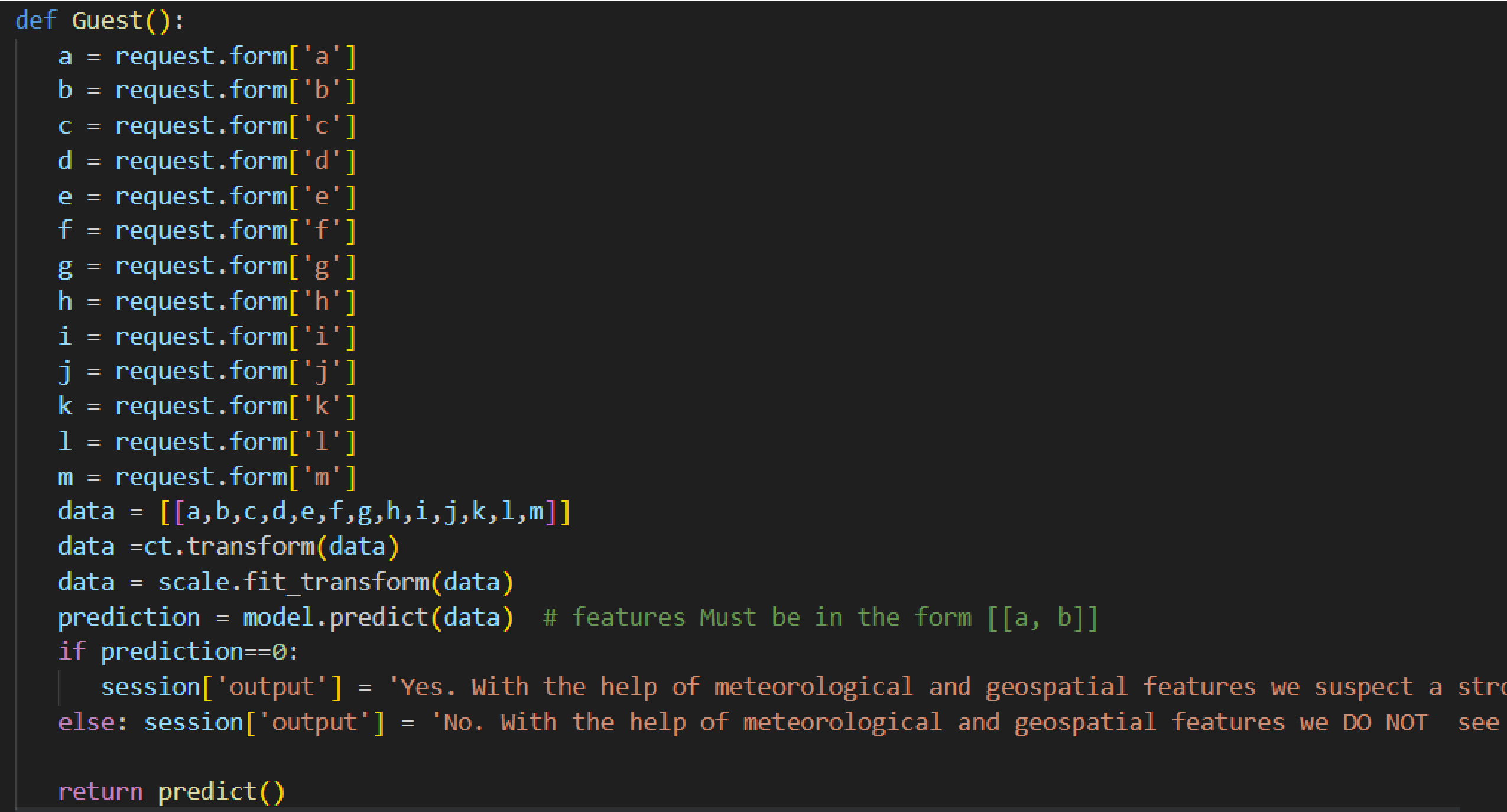
The function "form()" is then associated with this route. This function first loads the previously saved Linear Regression model using "model = pickle.load(open('model.pkl',

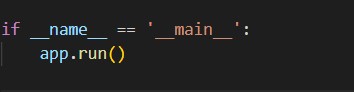
'rb'))".

Then, the function receives the user inputs for various geographical variables using "request.form['...']". The function then uses the loaded Logistic Regression model to predict the disease based on the user inputs.

Finally, the predicted body fat percentage is passed to an HTML template, where it is displayed to the user.

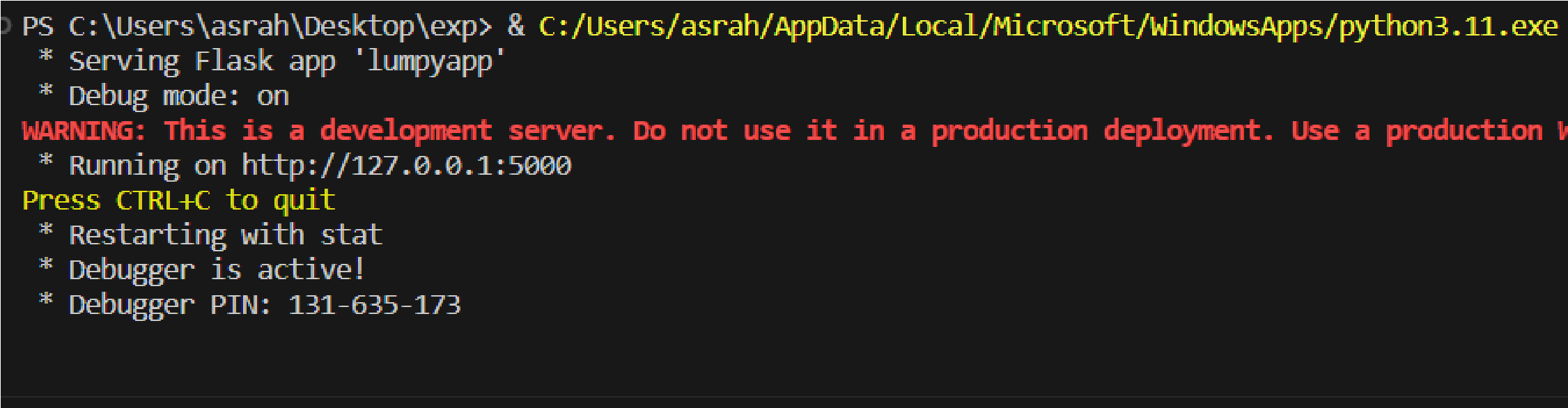
Main Function:

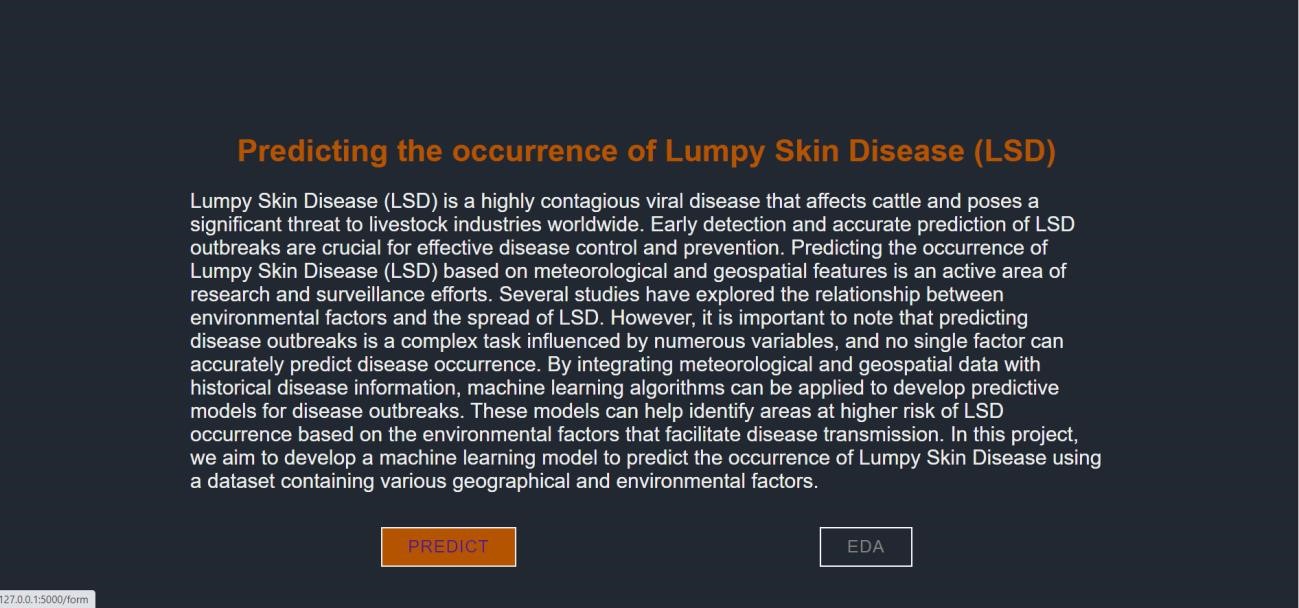
This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.



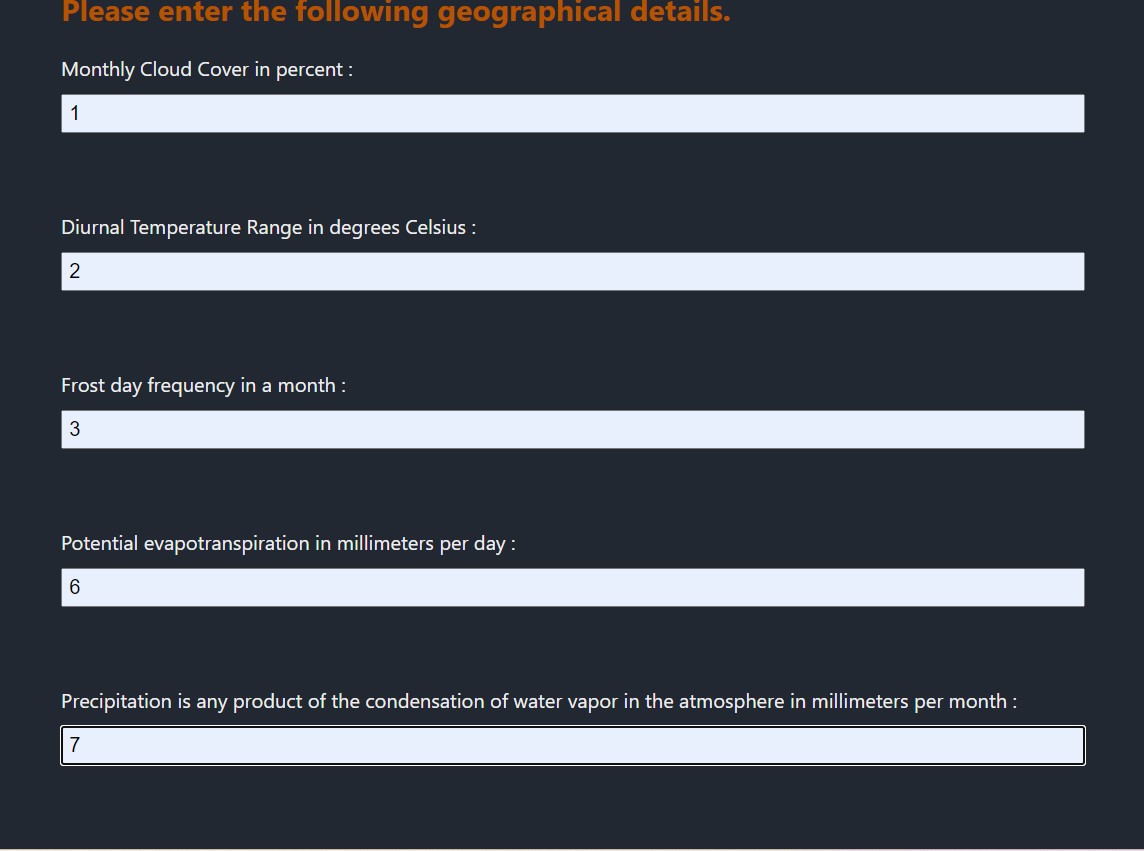
## Activity 2.3: Run the web application

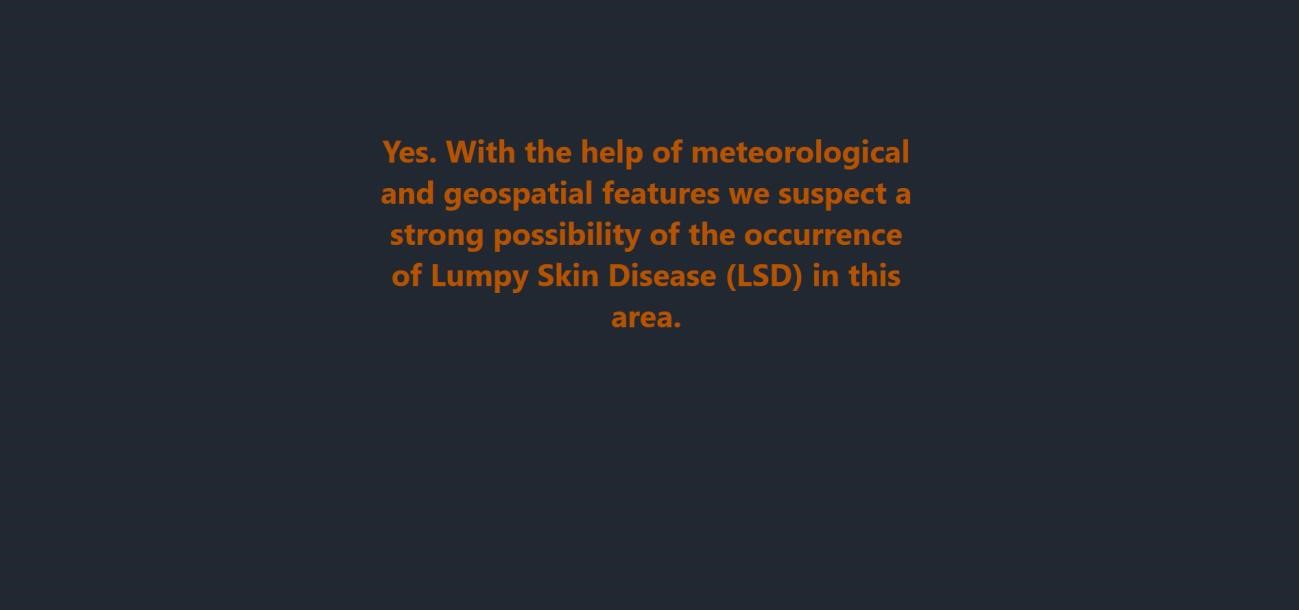
When you run the “main.py” file this window will open in the output terminal. Copy the [**http://127.0.0.1:5000**](http://127.0.0.1:5000/) and paste this link in your browser.

This is the "first\_page.html" file that appears when we paste the URL into the browser. To proceed to the next page, click the ‘Predict’ button.



On this page a user will input the following values and then click on “Predict” button to see the disease prediction.





# Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

**Activity 1: Project Documentation-Step by step project development procedure.**