

Predicting Lumpy Skin Disease

Introduction

Lumpy Skin Disease (LSD) is a highly contagious viral disease that affects livestock, posing a significant threat to livestock industries worldwide. Early detection and control of LSD outbreaks are crucial for effective disease management. In this project, we aim to develop a machine learning model to predict the occurrence of LSD 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 in a machine learning project for predicting lumpy skin disease typically revolves around identifying potential outbreaks of lumpy skin disease in animals, particularly in livestock such as cattle. Lumpy skin disease is a viral infection that affects cattle, causing characteristic skin nodules, fever, and other clinical signs. It can have significant economic implications for the agricultural industry due to the potential for reduced milk production, weight loss, and mortality in affected animals.

Activity 2: Business Requirements

Data Collection:

Source: Identify sources of relevant data, including meteorological data, geospatial information, historical disease occurrence records, and any other data that could impact the spread of lumpy skin disease.

Frequency: Determine the frequency at which data needs to be collected to maintain up-to-date models.

Data Preprocessing:

Cleaning: Implement data cleaning processes to handle missing values, outliers, and inconsistencies in the data.

Transformation: Apply necessary transformations such as scaling, encoding categorical variables, and feature engineering to prepare the data for modeling.

Feature Selection:

Identify Relevant Features: Work with domain experts to identify and select features that have a significant impact on lumpy skin disease occurrence.

Dimensionality Reduction: Explore techniques for dimensionality reduction if dealing with a large number of features.

Model Development:

Choice of Model: Select appropriate machine learning models for prediction (e.g., Support Vector Machines, Random Forests) based on the nature of the data and problem.

Training and Validation: Implement a training-validation strategy to develop and evaluate the model's performance.

Hyperparameter Tuning: Optimize model hyperparameters to improve predictive accuracy.

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.

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import missingno as msno
```

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.

```
data=pd.read_csv("/content/archive (10).zip")
data.head()
```

| | x | y | region | country | reportingDate | cld | dtr | frs | pet | pre | tmn | tmp | tmx | vap | wet | elevation | dominant_land_cover |
|---|-----------|-----------|--------|------------|---------------|------|------|-------|-----|------|-------|-------|------|------|------|-----------|---------------------|
| 0 | 90.380931 | 22.437184 | Asia | Bangladesh | 10/9/2020 | 41.6 | 12.8 | 0.00 | 2.3 | 1.7 | 12.7 | 19.1 | 25.5 | 15.7 | 0.00 | 147 | 2 |
| 1 | 87.854975 | 22.986757 | Asia | India | 20/12/2019 | 40.5 | 13.3 | 0.00 | 2.4 | 0.0 | 13.2 | 19.8 | 26.5 | 16.3 | 0.00 | 145 | 2 |
| 2 | 85.279935 | 23.610181 | Asia | India | 20/12/2019 | 27.3 | 13.6 | 0.08 | 2.3 | 0.6 | 9.4 | 16.2 | 23.0 | 13.0 | 0.98 | 158 | 2 |
| 3 | 81.564510 | 43.882221 | Asia | China | 25/10/2019 | 45.3 | 12.8 | 31.00 | 0.4 | 8.8 | -22.5 | -16.1 | -9.7 | 0.9 | 4.64 | 178 | 2 |
| 4 | 81.161057 | 43.834976 | Asia | China | 25/10/2019 | 38.8 | 13.2 | 31.00 | 0.4 | 10.5 | -20.4 | -13.8 | -7.2 | 1.2 | 1.69 | 185 | 3 |

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.

2.1 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.

2.2 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.

```

def get_continent_name(continent_code: str) -> str:
    continent_dict = {
        "NA": "North America",
        "SA": "South America",
        "AS": "Asia",
        "AF": "Africa",
        "OC": "Oceania",
        "EU": "Europe",
        "AQ": "Antarctica"
    }
    return continent_dict[continent_code]

get_continent_name("EU")

def get_continent(lat: float, lon: float) -> Tuple[str, str]:
    geolocator = Nominatim(user_agent="<username>@gmail.com", timeout=10)
    geocode = RateLimiter(geolocator.reverse, min_delay_seconds=1)

    location = geocode(f"{lat}, {lon}", language="en")

    # for cases where the location is not found, coordinates are antarctica
    if location is None:
        return "Antarctica", "Antarctica"

    # extract country code
    address = location.raw["address"]
    country_code = address["country_code"].upper()

    # get continent code from country code
    continent_code = pc.country_alpha2_to_continent_code(country_code)
    continent_name = get_continent_name(continent_code)

    return country_code, continent_name

```

Executing the aforementioned code snippet to implement the proposed solution.

```

df[["country", "region"]] = df.progress_apply(
    lambda x: get_continent(x["lat"], x["longitude"]), axis=1, result_type="expand"
)

```

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.

```

! pip install pycountry
import pycountry
def findCountry (country_name):
    try:
        return pycountry.countries.get(alpha_2= country_name).name
    except:
        return ("not found")
real_df['country']=real_df['country'].apply(findCountry)

```

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.

```
real_df.isnull().sum()
```

| | |
|---------------------------|-----|
| Unnamed: 0 | 0 |
| longitude | 0 |
| lat | 0 |
| cloud_cover | 0 |
| diurnal_temperature_range | 0 |
| frost_day_frequency | 0 |
| evapotranspiration | 0 |
| precipitation | 0 |
| tmn | 0 |
| tmp | 0 |
| tmx | 0 |
| vap | 0 |
| wet_day | 0 |
| elevation | 0 |
| dominant_land_cover | 0 |
| X5_Ct_2010_Da | 0 |
| X5_Bf_2010_Da | 0 |
| lumpy | 0 |
| country | 354 |
| region | 0 |
| dtype: int64 | |

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.


```
real_df['region'][real_df['country']=='not found'].unique()
```

```
array(['Africa', 'Europe'], dtype=object)
```

```
real_df[(real_df['region']=='Europe')&(real_df['country']=='not found')]=real_df[(real_df['region']=='Europe')&(real_df['country']=='not found')]  
real_df['country'].replace('not found', 'African Country', inplace = True)
```

Python

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.

```
ct = ColumnTransformer([('ohe', OneHotEncoder(), [12])], remainder= "passthrough")  
real_df = ct.fit_transform(real_df)  
real_df
```

```
array([[ 0. ,  1. ,  0. , ...,  0. , 147. ,  2. ],  
       [ 0. ,  1. ,  0. , ...,  0. , 145. ,  2. ],  
       [ 0. ,  1. ,  0. , ...,  0.98, 158. ,  2. ],  
       ...,  
       [ 1. ,  0. ,  0. , ..., 15.87, 178. ,  5. ],  
       [ 1. ,  0. ,  0. , ..., 16.3 , 180. ,  3. ],  
       [ 1. ,  0. ,  0. , ..., 16.67, 179. ,  3. ]])
```

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.

```
for column in real_df.columns:
    real_df[column].plot.hist()
    plt.title(f"Histogram of {column}")
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()
```

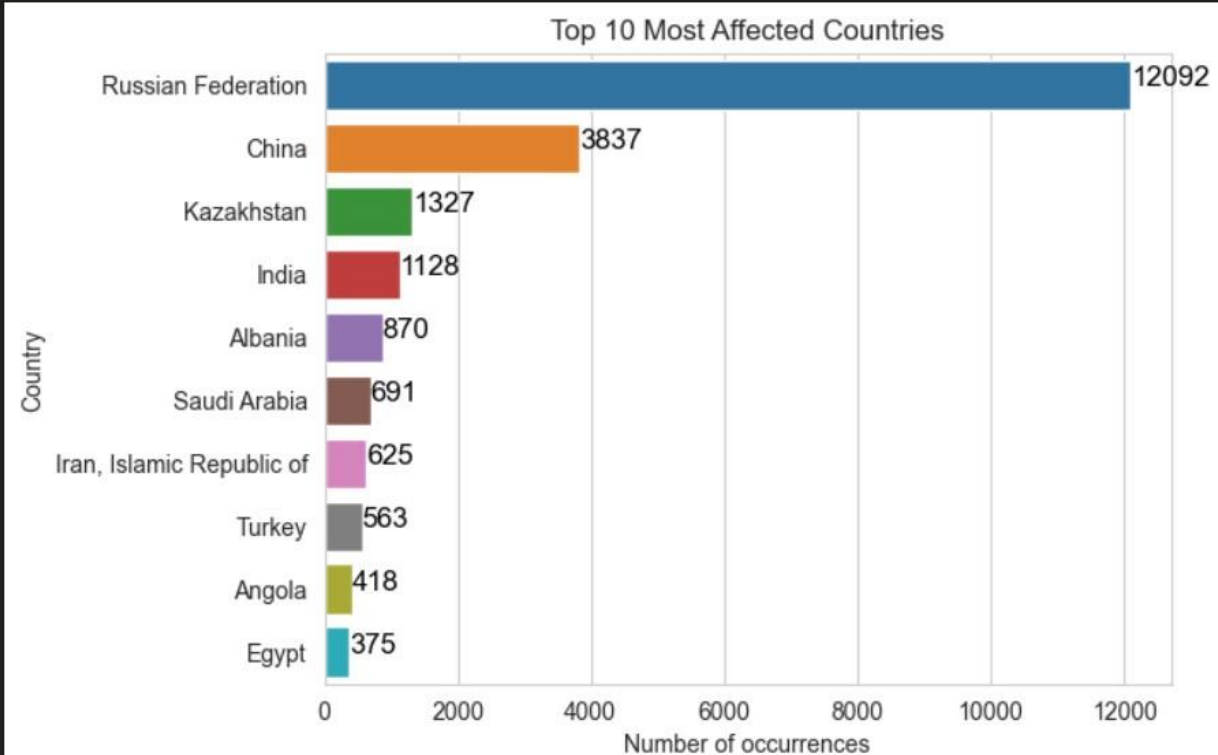
Activity 2.2: Bivariate analysis

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.

```

country_count = real_df['country'].value_counts()
country_count = country_count[:10]
sns.set_style("whitegrid")
sns.barplot(x=country_count.values, y=country_count.index)
plt.title('Top 10 Most Affected Countries')
plt.xlabel('Number of occurrences')
plt.ylabel('Country')
for i, v in enumerate(country_count.values):
    plt.text(v, i, str(v), color='black', fontsize=12)

```

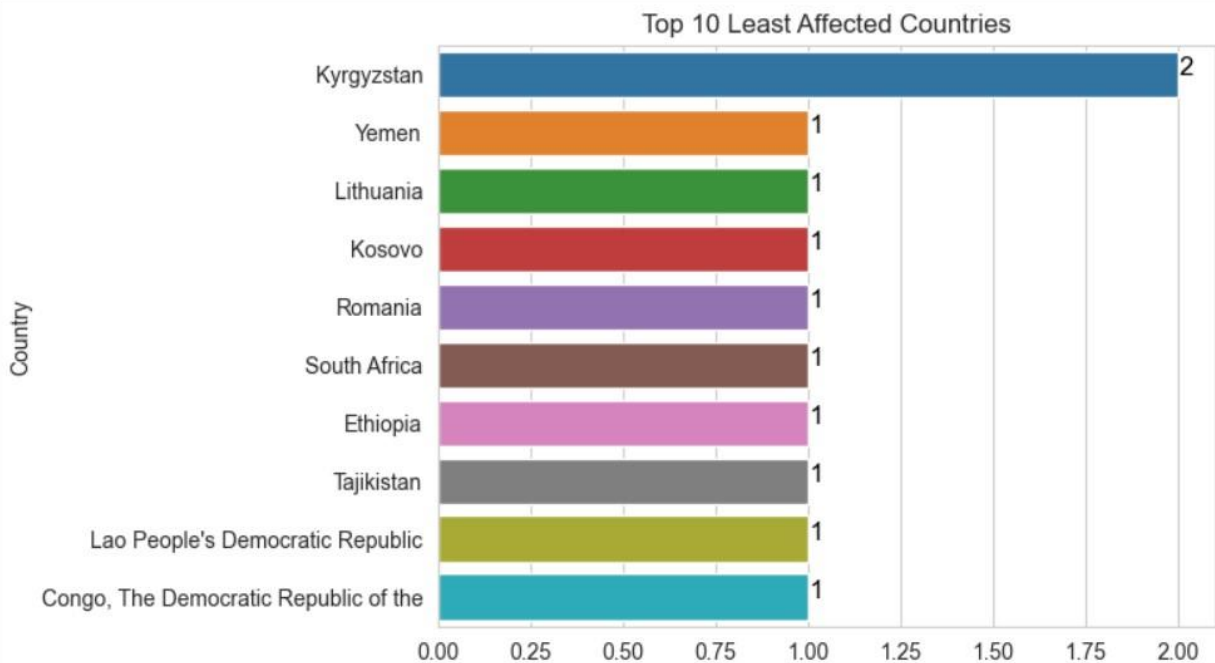


Similarly, employing the code depicted in the aforementioned visual representation, we can also determine the ten least affected countries. This code utilizes a specific approach to analyze the dataset and extract the pertinent information, enabling the identification of countries that experienced relatively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

```

country_count = real_df['country'].value_counts()
country_count = country_count[-10:]
sns.set_style("whitegrid")
sns.barplot(x=country_count.values, y=country_count.index)
plt.title('Top 10 Least Affected Countries')
plt.xlabel('Number of occurrences')
plt.ylabel('Country')
for i, v in enumerate(country_count.values):
    plt.text(v, i, str(v), color='black', fontsize=12)

```

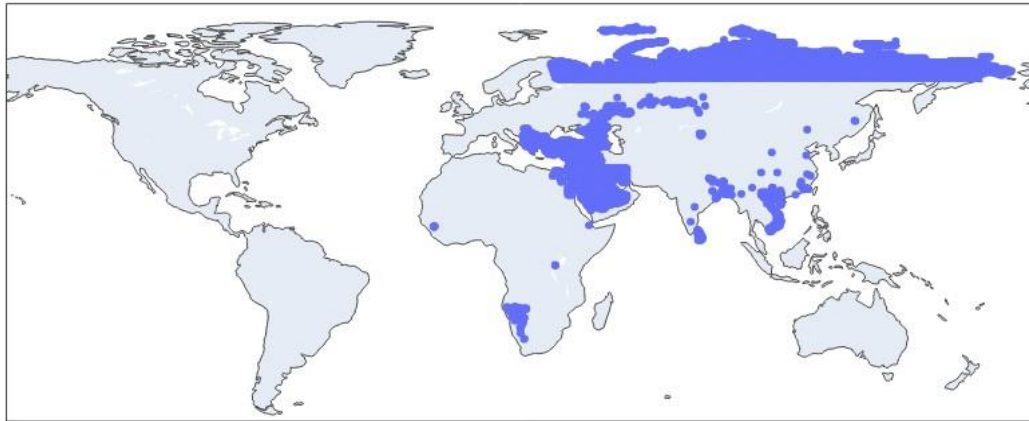


To determine the quantity of datapoints available in this dataset, we can leverage the Plotly module and its Scatter Geo function. By plotting the "longitude" and "latitude" columns on a map using this function, we can visualize the geographical distribution of the data points. This enables us to gain 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.

```

fig = px.scatter_geo(real_df, lat='lat', lon='longitude')
fig.update_layout(title = 'Lumpy Skin Disease Area', title_x=0.5)
fig.show()

```



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.

```
import plotly.express as px

# Assuming 'df' is the DataFrame containing longitude, latitude, and target columns
fig = px.scatter_mapbox(real_df, lat='lat', lon='longitude', hover_name='country',
                        color='lumpy', color_continuous_scale='algae',
                        mapbox_style='open-street-map', zoom=10)

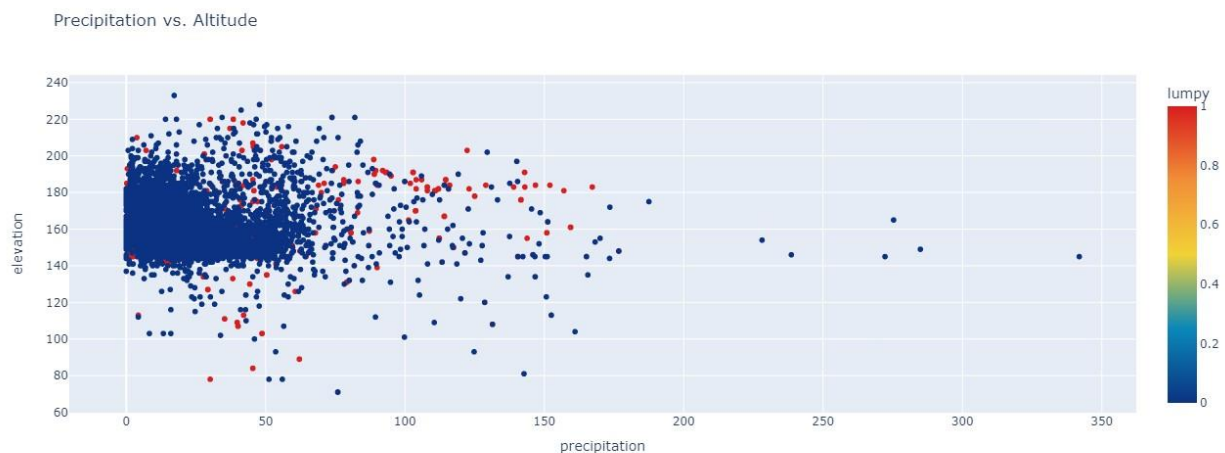
fig.update_layout(title='Geographic Locations with Target Hue',
                  margin=dict(l=0, r=0, t=50, b=0),
                  mapbox=dict(center=dict(lat=real_df['lat'].mean(), lon=real_df['longitude'].mean()))

fig.show()
```



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.

```
fig = px.scatter(real_df, x='precipitation', y='elevation', color='lumpy',  
| | | | | hover_data=['country'], title='Precipitation vs. Altitude')  
fig.show()
```

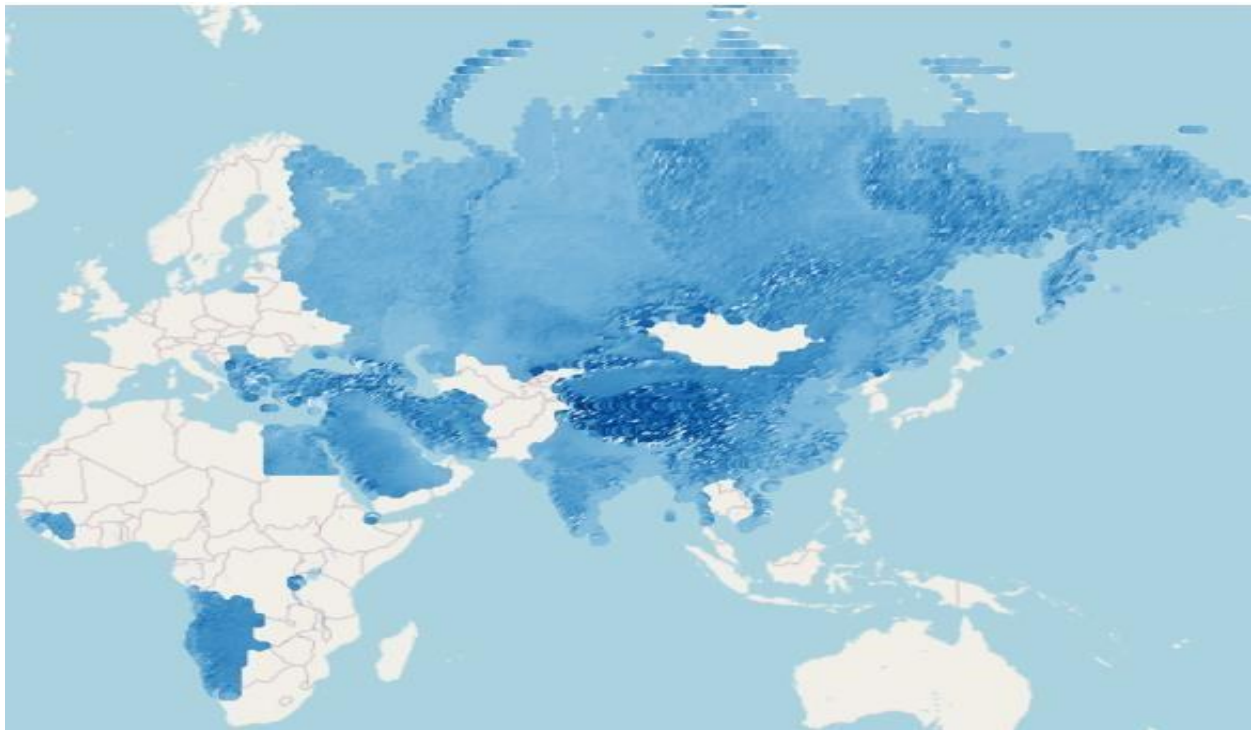


To ascertain the interplay between elevation and geographical coordinates, we employ the following code snippet as part of our analytical methodology.


```
fig = px.scatter_mapbox(real_df, lat='lat', lon='longitude', hover_name='country',
                        color='elevation', color_continuous_scale='blues',
                        mapbox_style='open-street-map', zoom=10)

fig.update_layout(title='Geographic Locations with Target Hue',
                  margin=dict(l=0, r=0, t=50, b=0),
                  mapbox=dict(center=dict(lat=real_df['lat'].mean(), lon=real_df['longitude'].mean()))

fig.show()
```



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.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1,stratify=y)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(21353, 16)
(21353,)
(2373, 16)
(2373,)
```

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: Logistic Regression Model.

The provided code leverages the scikit-learn library to conduct logistic regression modeling. This process involves utilizing regression algorithms and techniques to analyze the relationship between variables and make predictions based on the linear relationship observed in the data. By employing the scikit-learn library, we can effectively apply the principles of logistic regression to model and interpret the data, aiding in making informed decisions and drawing meaningful insights.

```
from sklearn.linear_model import LogisticRegression
# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()
logreg.fit(rescaledX_train, yTrain)
```

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,

```
test_acc=[]
```


named "dtt", 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.

```
# import numpy
import numpy as np
error_rate = []
# searching k value upto 40
for i in range(1,40):
    # knn algorithm
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(rescaledX_train, yTrain)
    # testing the model
    y_pred_test_knn = knn.predict(rescaledX_test)
    #training the model
    y_pred_train_knn = knn.predict(rescaledX_train)

    error_rate.append(np.mean(y_pred_test_knn != yTest))
# Configure and plot error rate over k values
plt.figure(figsize=(10,4))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker='o', markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K-Values')
plt.xlabel('K-Values')
plt.ylabel('Error Rate')
```

Milestone 5: Performance Testing

Activity 1: 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 underfitting.

Analyzing 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 1.1: Logistic Regression Model.

```
print("Test: Accuracy = ", logreg.score(rescaledX_test,yTest))
print("Train: Accuracy = ", logreg.score(rescaledX_train,yTrain))

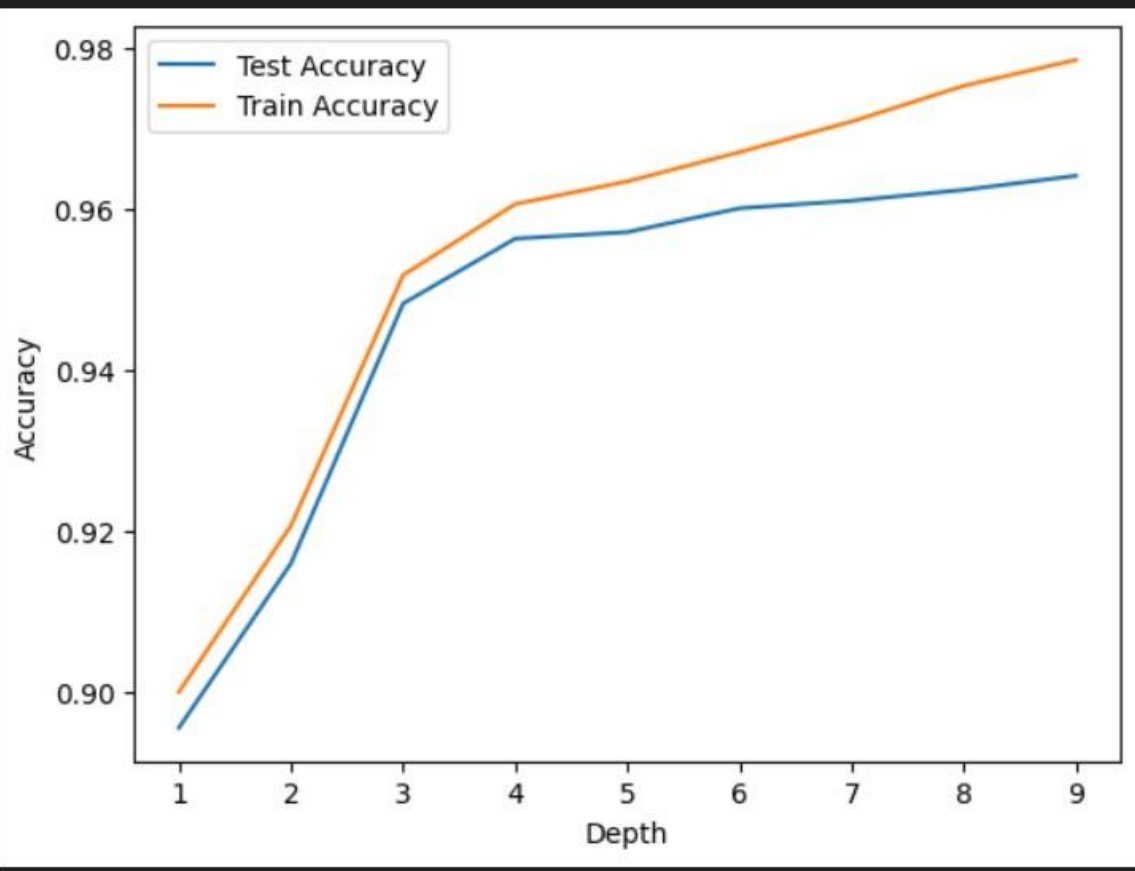
confusion_matrix = metrics.confusion_matrix(yTest,y_pred_test_logreg)
print(confusion_matrix)
```

```
Test: Accuracy = 0.9461093938986695
Train: Accuracy = 0.9540375532772722
[[6445  74]
 [ 327 595]]
```

Activity 1.2: Decision Tree Classifier Model.

```
df2 = DataFrame (list_score,columns=['Depth','Train Accuracy','Test Accuracy'])
plt.plot(df2['Depth'],df2['Test Accuracy'],label='Test Accuracy')
plt.plot(df2['Depth'],df2['Train Accuracy'],label='Train Accuracy')
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend()
```

matplotlib.legend.Legend at 0x7fdfab173580>



Activity 1.3: K Nearest Neighbours Classifier Model.

```
test_acc_knn = accuracy_score(yTest, y_pred_test_knn)
train_acc_knn = accuracy_score(yTrain,y_pred_train_knn)
print('Train score:',train_acc_knn,'Test score:',test_acc_knn)
print(i,'Train score:',train_acc_knn,'Test score:',test_acc_knn)
```

```
Train score: 0.9629650961870753 Test score: 0.9587421045558393
39 Train score: 0.9629650961870753 Test score: 0.9587421045558393
```

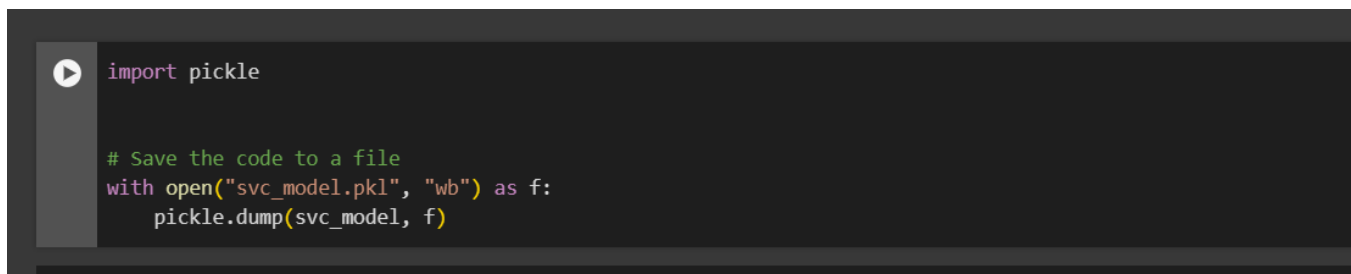
Activity 2: Comparing models

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. 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.

A code editor snippet with a dark background. On the left, there is a vertical toolbar with a play button icon. The code is written in a light-colored font. It starts with 'import pickle' on the first line. The second line is a comment: '# Save the code to a file'. The third line is 'with open("svc_model.pkl", "wb") as f:', and the fourth line is 'pickle.dump(svc_model, f)'.

```
import pickle

# Save the code to a file
with open("svc_model.pkl", "wb") as f:
    pickle.dump(svc_model, f)
```

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

```
import pickle
from flask import Flask, render_template, request
import pandas as pd
import numpy as np
```

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.

```
model = pickle.load(open("model.pkl", 'rb'))
scale = pickle.load(open("scale.pkl", 'rb'))
ct = joblib.load('column')
```

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.

```
@app.route('/')
def home():
    return render_template('first_page.html')
```

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".

```
@app.route('/form')
def form():
    return render_template('form_page.html')
@app.route('/guest', methods = ["POST"])
```

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'))".

```
def Guest():
    a = request.form['a']
    b = request.form['b']
    c = request.form['c']
    d = request.form['d']
    e = request.form['e']
    f = request.form['f']
    g = request.form['g']
    h = request.form['h']
    i = request.form['i']
    j = request.form['j']
    k = request.form['k']
    l = request.form['l']
    m = request.form['m']
    data = [[a,b,c,d,e,f,g,h,i,j,k,l,m]]
    data = ct.transform(data)
    data = scale.fit_transform(data)
    prediction = model.predict(data) # features Must be in the form [[a, b]]
    if prediction==0:
        session['output'] = 'Yes. With the help of meteorological and geospatial features we suspect a stroke'
    else: session['output'] = 'No. With the help of meteorological and geospatial features we DO NOT see a stroke'
    return predict()
```

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.

```
if __name__ == '__main__':
    app.run()
```

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> and paste this link in your browser.

```
PS C:\Users\asrah\Desktop\exp> & C:/Users/asrah/AppData/Local/Microsoft/WindowsApps/python3.11.exe  
* Serving Flask app 'lumpyapp'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production W  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 131-635-173
```

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

Lumpy Skin Disease Prediction

Monthly Cloud Cover (%):

Diurnal Temperature Range (°C):

Frost Day Frequency (per month):

Potential Evapotranspiration (mm/day):

Precipitation (mm/month):

Predict

Yes. With the help of meteorological and geospatial features we suspect a strong possibility of the occurrence of Lumpy Skin Disease (LSD) in this area.

