

Topic - Rabies positive cases prediction

The dataset contains rabies laboratory submission data for New York State (excluding NYC), collected by the Wadsworth Center Rabies Laboratory since May 2007. The data is updated quarterly and used for health monitoring, funding decisions, and vaccination strategies.

Link to DataSet :

https://health.data.ny.gov/Health/Rabies-Laboratory-Submissions-Beginning-May-2007/q25r-zbis/about_data

Dataset Overview:

This dataset consists of 33,895 entries across 10 columns, covering time, location, animal classification, and test results.

Types of Data Present :

Time-Based Data

Received Year (Text): The year the rabies sample was received.

Received Month (Number): The month the sample was received.

Report Date (Text): The date when the data was officially reported.

Geographical Data

County Name (Text): The county where the specimen was collected, the bite occurred, or the test was processed.

Latitude (Text): The approximate center point latitude of the county.

Longitude (Text): The approximate center point longitude of the county.

Geocode (Point): Not included due to privacy concerns.

Categorical Data

Animal Classification (Text): Groups the tested animals into categories such as:

Cattle, Bats, Cat, Dog, Raccoon, Skunk, Other Domestic (e.g., horses, donkeys, ferrets), Other Wild (e.g., deer, coyotes, otters, etc.), Rodents/Lagomorphs (e.g., rats, squirrels, rabbits, etc.)

Numerical Data

Number of Samples (Number): The total number of rabies test samples received.

Positive Samples (Number): The number of samples that tested positive for rabies.

Statistical summary of the numerical columns

: Provides insight into trends, distribution, and variability within the dataset.

```
1 df.describe()
```

	Received Year	Received Month	Number of Samples	Positive Samples	Latitude Center Point	Longitude Center Point
count	33895.000000	33895.000000	33895.000000	33895.000000	33895.000000	33895.000000
mean	2015.660481	6.589792	3.173595	0.181797	42.667271	-75.545364
std	4.922545	3.174228	8.113104	0.504999	0.900112	1.808321
min	2007.000000	1.000000	1.000000	0.000000	40.746070	-79.501690
25%	2011.000000	4.000000	1.000000	0.000000	42.249420	-76.875560
50%	2016.000000	7.000000	1.000000	0.000000	42.740220	-75.230000
75%	2020.000000	9.000000	3.000000	0.000000	43.061690	-73.794900
max	2024.000000	12.000000	366.000000	9.000000	44.848880	-72.662140

Checking NULL Values

```
1 #checking missing value:
2 print(df.isnull().sum())
```

Received Year	0
Received Month	0
Report Date	0
County Name	0
Animal Classification	0
Number of Samples	0
Positive Samples	0
Latitude Center Point	0
Longitude Center Point	0
Geocode	0

dtype: int64

In the dataset we do not have any NULL values :)

Duplicate Value Removal:

```
Checking Duplicates

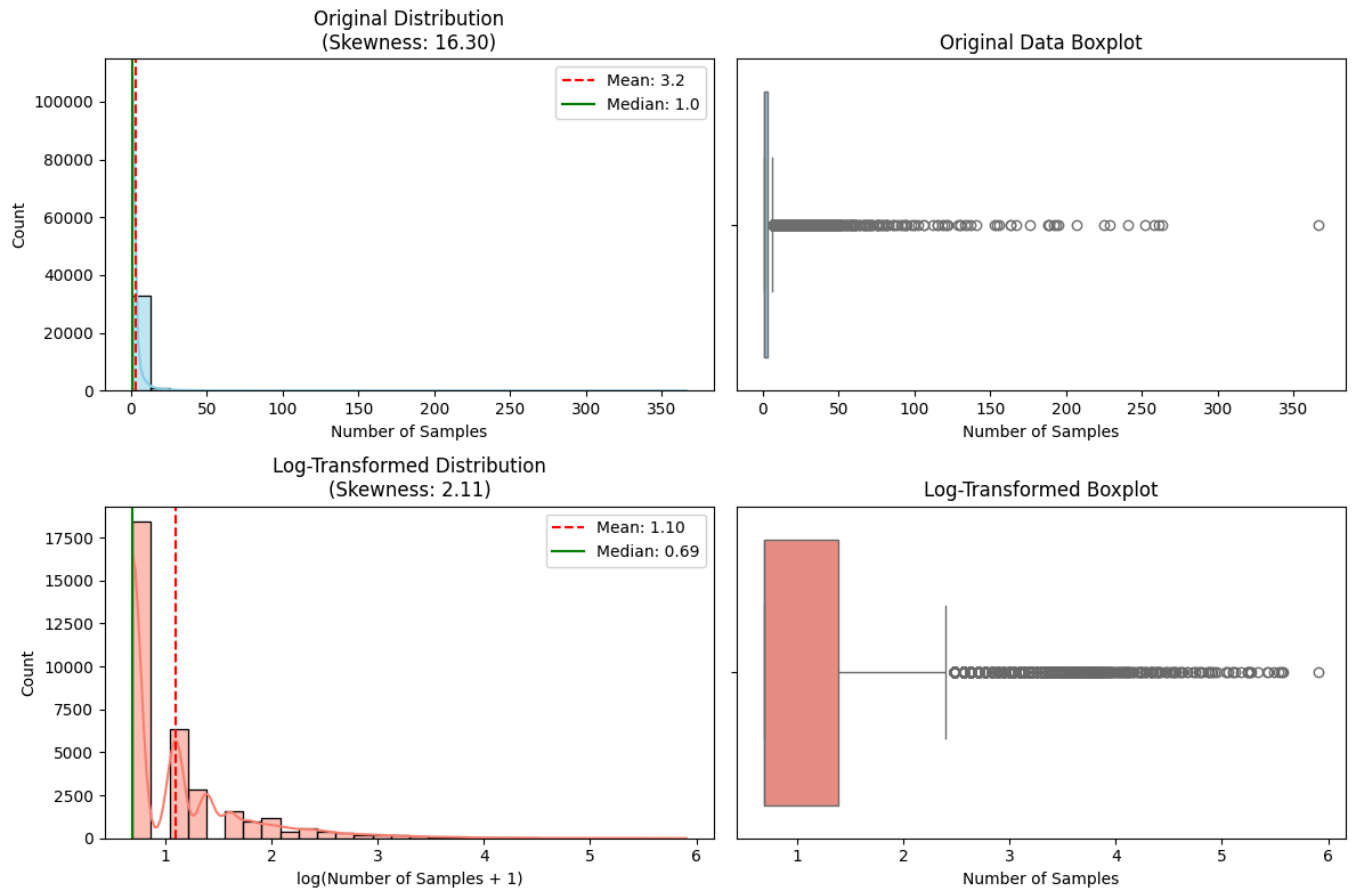
1 duplicates = df[df.duplicated()]
2 print("Duplicate Rows:\n", duplicates)
```

Duplicate Rows:
Empty DataFrame
Columns: [Received Year, Received Month, Report Date, County Name, Animal Classification, Number of Samples, Positive Samples, Latitude Center Point, Longitude Center Point, Geocode]
Index: []

There are no duplicate values in the dataset as well.

Outlier Treatment:

Since the Number of Samples Column has outliers, we need to treat it.



Since data is skewed, applying *log transform* on the Number of Samples Column.

Transforming Categorical Features to apply One Hot Encoding:

There are 57 counties, but generating categorical encoding for each of them would be challenging. As a result, regions of counties were created and one hot encoding was applied.



```
region_mapping = {
    'Western NY': ['Cattaraugus', 'Chautauqua', 'Erie', 'Niagara',
                   'Orleans', 'Genesee', 'Wyoming', 'Allegany'],
    'Finger Lakes': ['Monroe', 'Livingston', 'Ontario', 'Wayne', 'Yates',
                     'Seneca', 'Steuben', 'Chemung', 'Schuyler', 'Tompkins'],
```

```

    'Central NY': ['Onondaga', 'Cayuga', 'Cortland', 'Madison', 'Oswego',
'Oneida', 'Chenango', 'Otsego', 'Broome', 'Tioga', 'Herkimer'],
    'North Country': ['Clinton', 'Franklin', 'Essex', 'Hamilton', 'Lewis',
'St. Lawrence', 'Jefferson'],
    'Capital Region': ['Albany', 'Schenectady', 'Rensselaer', 'Saratoga',
'Warren', 'Washington', 'Fulton', 'Montgomery', 'Schoharie', 'Greene',
'Columbia'],
    'Hudson Valley': ['Westchester', 'Rockland', 'Putnam', 'Orange',
'Dutchess', 'Ulster', 'Sullivan', 'Delaware'],
    'Long Island': ['Nassau', 'Suffolk']
}

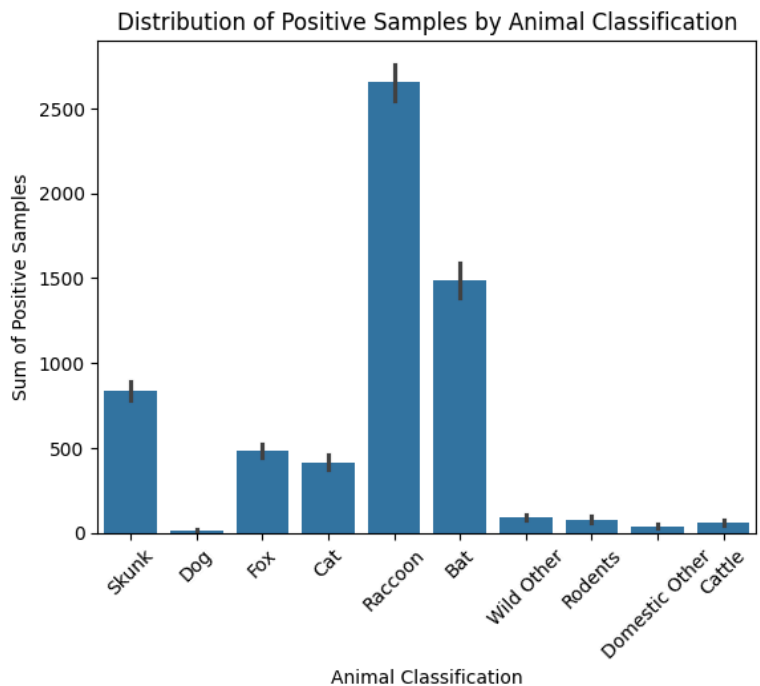
```

```
display(df[['County Name', 'Geographic Region']].head())
```

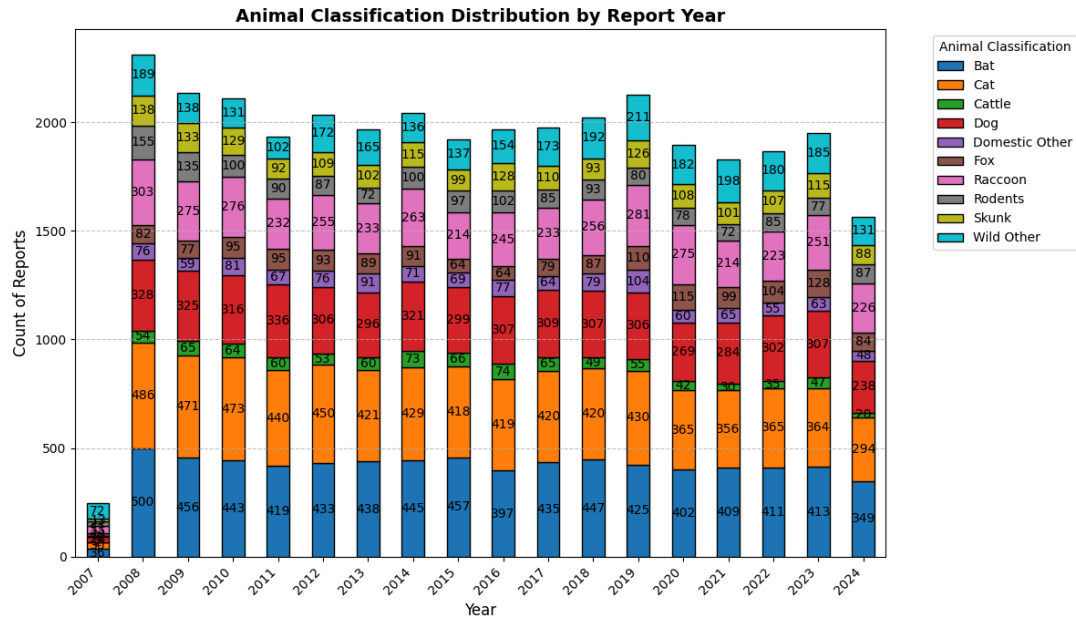
County Name	Geographic Region	
Cattaraugus	Western NY	
St. Lawrence	North Country	
Orange	Hudson Valley	
Washington	Capital Region	
Jefferson	North Country	

Visualization Graphs

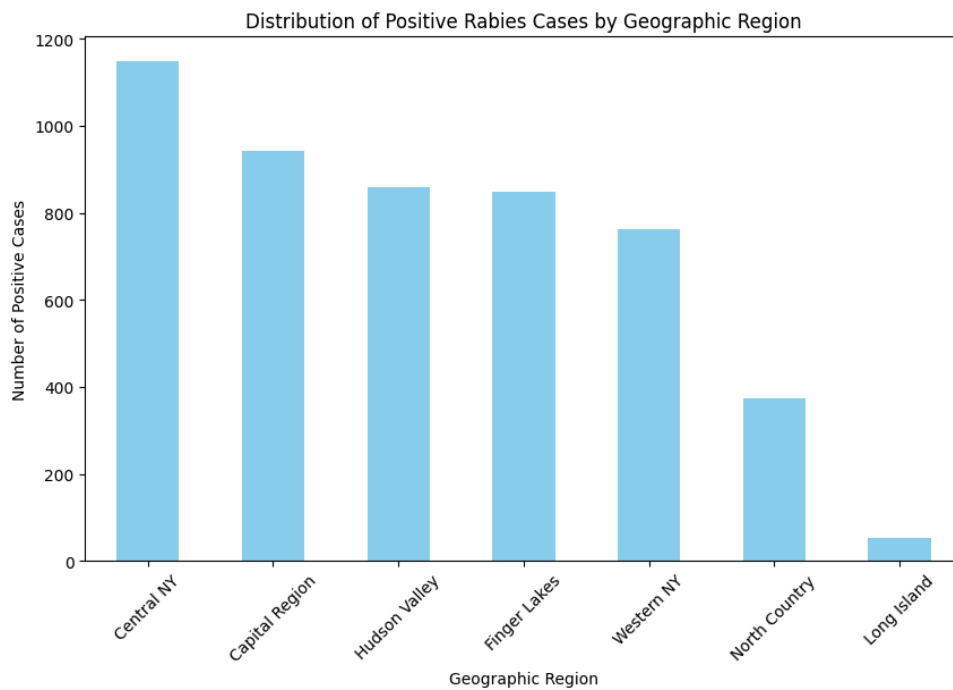
This graph displays the count of positive rabies samples in the dataset, categorized by the animal responsible for transmitting the rabies.



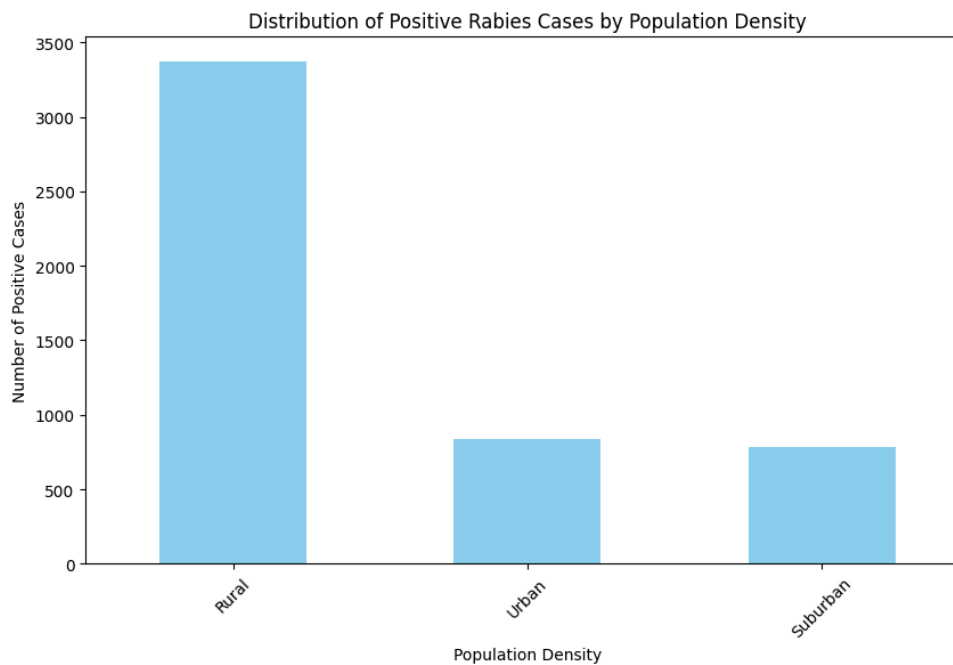
This graph shows the number of animal samples in the dataset, grouped by year.



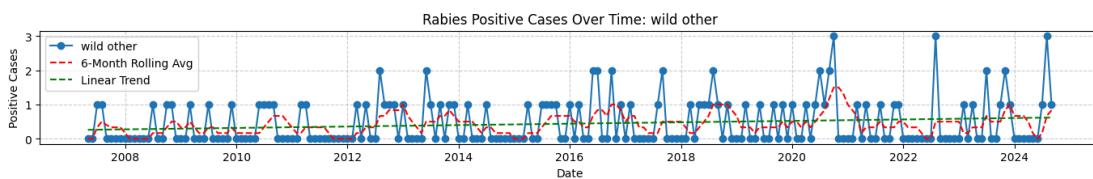
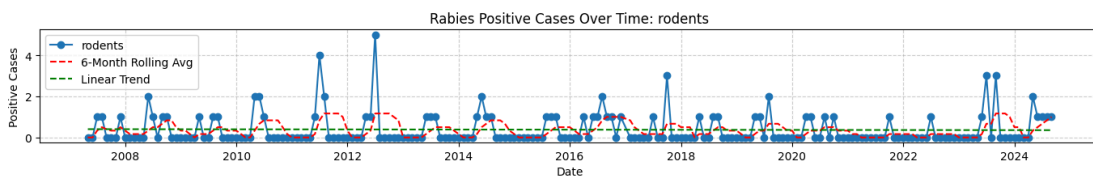
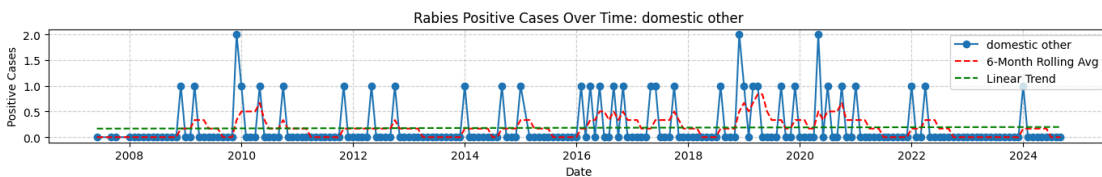
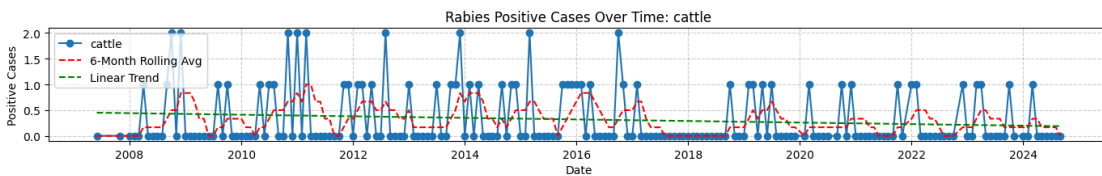
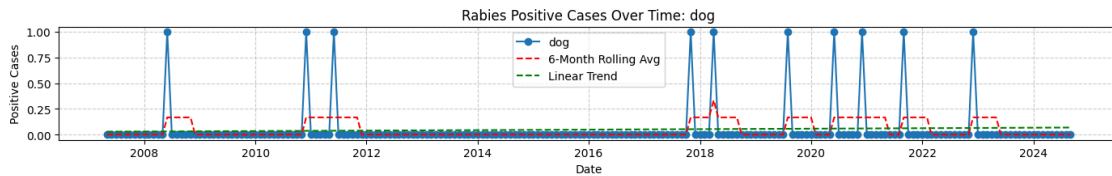
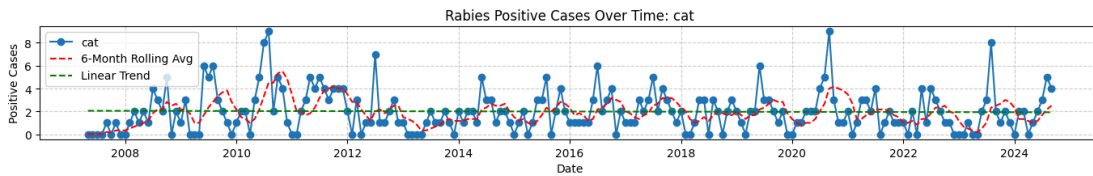
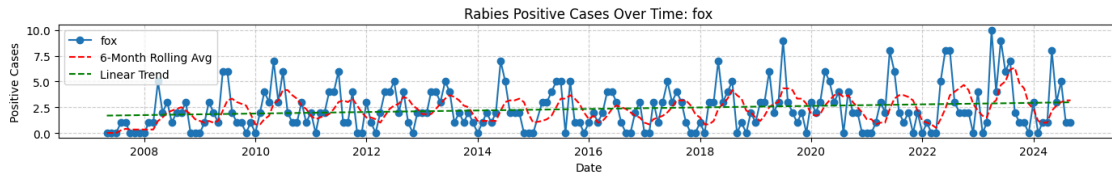
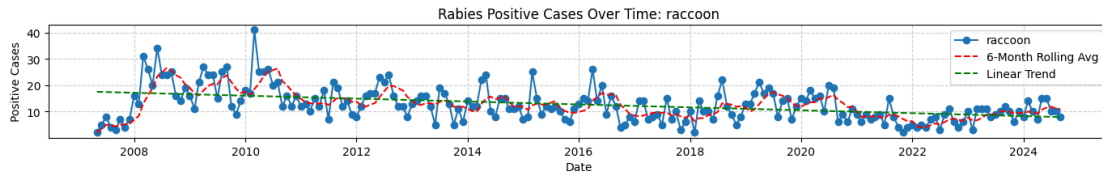
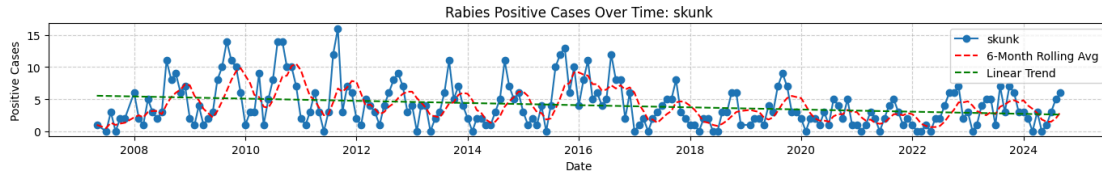
There are 57 counties, but generating categorical encoding for each of them would be challenging. As a result, regions of counties were created. This graph shows the number of positive rabies cases by geographical region.

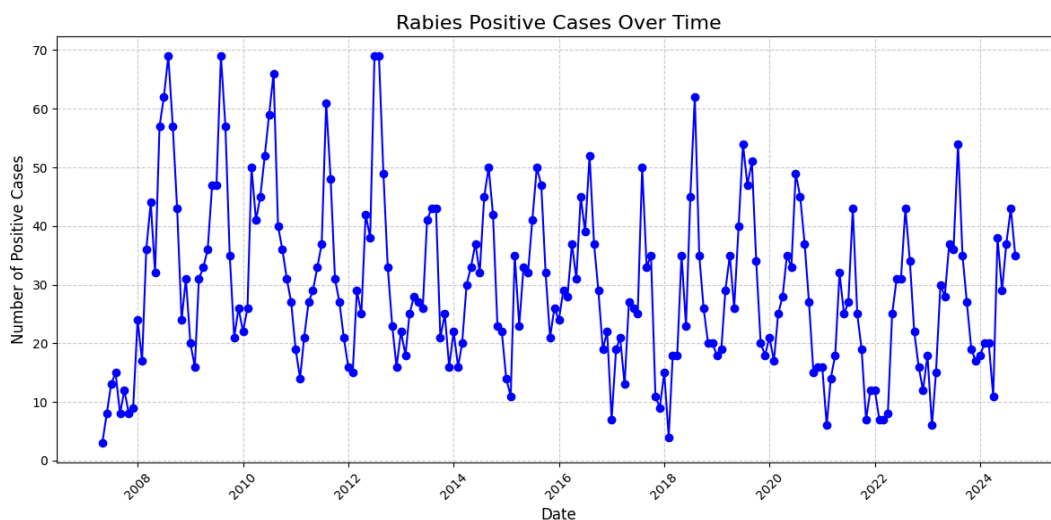


This graph shows the number of positive rabies cases by population density.

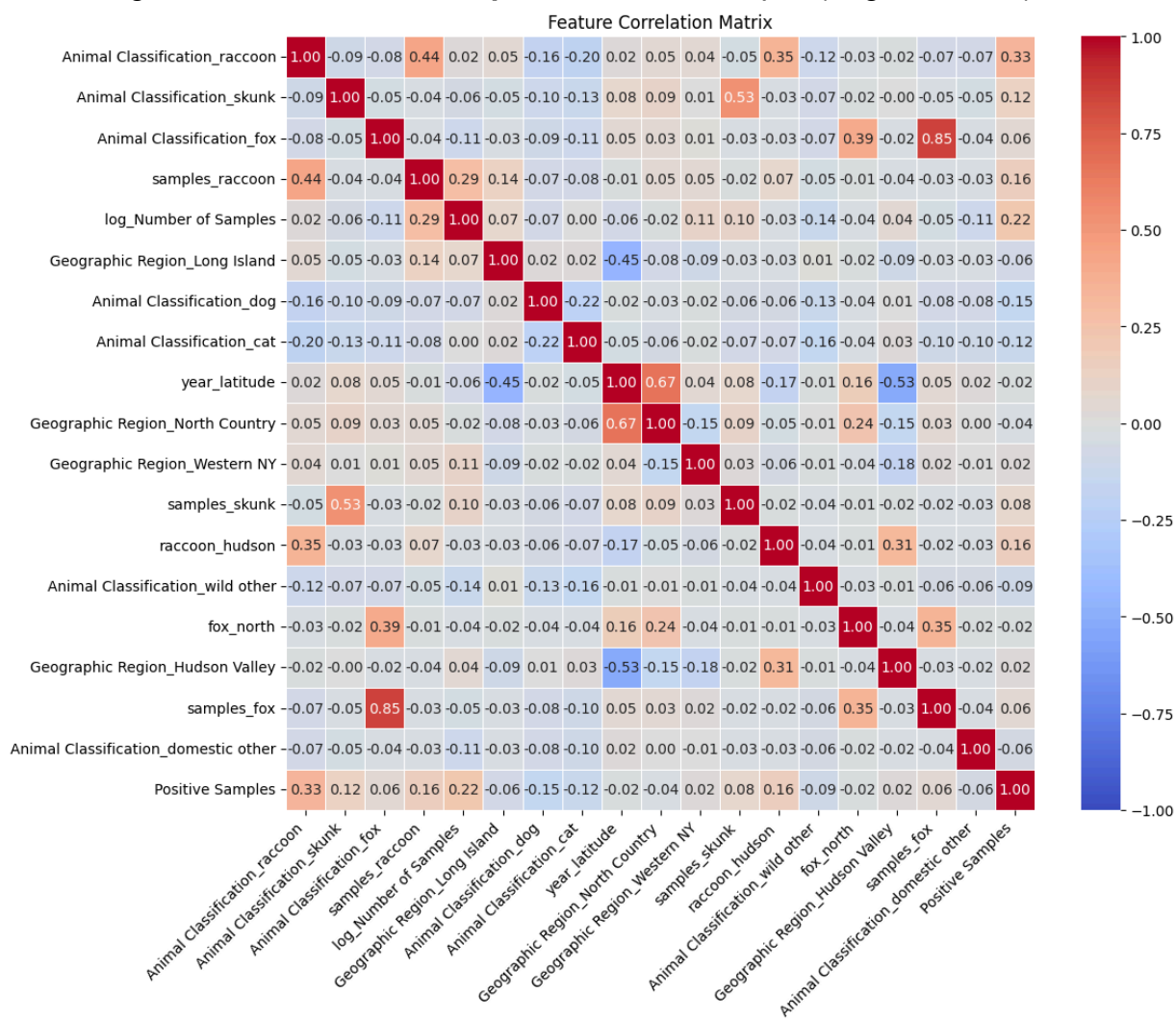


This graph shows the number of positive rabies cases by animals over time.





Performing the **Correlation Heatmap** with Positive Samples(target variable):



Algorithms Training Report:

Logistic Regression:

Mathematical Representation:

$$P(y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1x_1+\beta_2x_2+\dots+\beta_nx_n)}}$$

The logistic regression model estimates the probability that a given input X belongs to the positive class (1).

Key Features:

- Simple and interpretable model for binary classification.
- The output is probabilistic, providing both class prediction and confidence levels.
- Uses a sigmoid function to model probabilities.

Advantages:

- Easy to implement and interpret.
- Works well for linearly separable data.
- Can handle imbalanced datasets with the `class_weight='balanced'` option.

Disadvantages:

- Assumes linearity between the features and the log-odds of the target.
- Struggles with complex patterns or non-linear relationships in data.

For this algorithm we got a validation accuracy of 78.67% and test accuracy of 78.13%.

```
Validation Performance:
Accuracy: 0.786688798560472

Classification Report:
      precision    recall  f1-score   support

     0       0.96       0.78       0.86       4626
     1       0.39       0.81       0.53        798

 accuracy       0.68       0.80       0.79       5424
 macro avg       0.68       0.80       0.69       5424
weighted avg       0.88       0.79       0.81       5424

Confusion Matrix:
[[3622 1004]
 [ 153  645]]

Validation Log Loss: 0.478175676006386

Test Performance:
Accuracy: 0.7813836849092787

Classification Report:
      precision    recall  f1-score   support

     0       0.96       0.78       0.86       5782
     1       0.38       0.80       0.52        997

 accuracy       0.67       0.79       0.78       6779
 macro avg       0.67       0.79       0.69       6779
weighted avg       0.87       0.78       0.81       6779

Confusion Matrix:
[[4501 1281]
 [ 201  796]]

Test Log Loss: 0.4755153352930148
```

SVM:

Mathematical Representation:

$$f(X) = \sum_{i=1}^n \alpha_i y_i K(X_i, X) + b$$

- Where α_i are Lagrange multipliers, $K(X_i, X)$ is the kernel function, and b is the bias term.

Key Features:

- SVM attempts to find the hyperplane that best separates classes in high-dimensional space.
- Handles non-linear classification via the kernel trick (e.g., radial basis function kernel).

Advantages:

- Very effective in high-dimensional spaces.
- Works well for both linear and non-linear classification.

Disadvantages:

- Computationally expensive for large datasets.
- Sensitive to noise and outliers.

For this algorithm we got a validation accuracy of 81.31% and test accuracy of 81.38%.

```
Validation Metrics:
Accuracy: 0.8131
Log Loss: 0.3027
      precision    recall  f1-score   support

     0       0.96      0.82      0.88       4626
     1       0.43      0.79      0.56        798

   accuracy          0.81       5424
  macro avg       0.69      0.81      0.72       5424
 weighted avg       0.88      0.81      0.83       5424

[[3776  850]
 [ 164  634]]

Test Metrics:
Accuracy: 0.8138
Log Loss: 0.6327
      precision    recall  f1-score   support

     0       0.96      0.82      0.88       5782
     1       0.43      0.79      0.56        997

   accuracy          0.81       6779
  macro avg       0.69      0.80      0.72       6779
 weighted avg       0.88      0.81      0.83       6779

[[4730 1052]
 [ 210  787]]
```

XG Boost:

Mathematical Representation:

$$F(x)^{(t)} = F(x)^{(t-1)} + \eta \sum_{i=1}^n \gamma_i h_i(x)$$

- Where $F(x)^{(t)}$ is the model prediction at step t , γ_i is the weight, and $h_i(x)$ is the weak learner at step t .

Key Features:

- XGBoost is a boosting algorithm that builds multiple weak learners (decision trees) in a sequential manner, correcting errors made by the previous ones.
- It handles missing data and works well with imbalanced datasets.

Advantages:

- Excellent performance for both regression and classification tasks.
- Can be fine-tuned with several hyperparameters for optimal performance.
- Handles large datasets efficiently.

Disadvantages:

- Prone to overfitting if hyperparameters are not tuned properly.
- Requires careful parameter tuning to achieve good performance.

For this algorithm we got a validation accuracy of 84.53% and test accuracy of 84.63%.

```
Validation Performance:
Accuracy: 0.8453171091445427

Classification Report:
      precision    recall  f1-score   support

     0       0.95      0.87      0.91      4626
     1       0.48      0.72      0.58       798

 accuracy      0.85      0.85      0.85      5424
  macro avg       0.71      0.79      0.74      5424
 weighted avg       0.88      0.85      0.86      5424

Confusion Matrix:
[[4011  615]
 [ 224  574]]

Validation Log Loss: 0.4537847869222347

Test Performance:
Accuracy: 0.8462900132762944

Classification Report:
      precision    recall  f1-score   support

     0       0.95      0.87      0.91      5782
     1       0.48      0.71      0.58       997

 accuracy      0.85      0.85      0.85      6779
  macro avg       0.72      0.79      0.74      6779
 weighted avg       0.88      0.85      0.86      6779

Confusion Matrix:
[[5027  755]
 [ 287  710]]

Test Log Loss: 0.44481715524695997
```

Neural Network Model :

Mathematical Representation:

$$y = \sigma(WX + b)$$

- Where W is the weight matrix, X is the input, b is the bias, and σ is the activation function (e.g., ReLU or Sigmoid).

Key Features:

- Neural networks consist of layers of interconnected nodes (neurons), and they are capable of learning complex patterns in large datasets.
- The network learns weights during training via backpropagation and optimization algorithms (e.g., Adam optimizer).

Advantages:

- Highly flexible and can model complex relationships in the data.
- Suitable for tasks with large amounts of data and intricate patterns (e.g., image, speech recognition).

Disadvantages:

- Requires a large amount of data to train effectively.
- Can be computationally intensive and prone to overfitting without proper regularization or tuning.

For this algorithm we got a validation accuracy of 87.02% and test accuracy of 87.21%.

```
Validation Performance:
Accuracy: 0.8702064896755162

Classification Report:
      precision    recall  f1-score   support

     0       0.91      0.95      0.93      4626
     1       0.58      0.43      0.49       798

 accuracy      0.87      0.87      0.87      5424
 macro avg      0.74      0.69      0.71      5424
 weighted avg      0.86      0.87      0.86      5424

Confusion Matrix:
[[4376  250]
 [ 454  344]]

Validation Log Loss: 0.2931181996127205

Test Performance:
Accuracy: 0.8721050302404485

Classification Report:
      precision    recall  f1-score   support

     0       0.91      0.95      0.93      5782
     1       0.59      0.43      0.50       997

 accuracy      0.87      0.87      0.87      6779
 macro avg      0.75      0.69      0.71      6779
 weighted avg      0.86      0.87      0.86      6779

Confusion Matrix:
[[5482  300]
 [ 567  430]]

Test Log Loss: 0.28834175014128843
```

Neural Network Model Details :

Input Layer : The first layer (Dense) has *10 neurons* and uses the *ReLU (Rectified Linear Unit)* activation function. It expects input data with the shape corresponding to the number of features in `X_train` (i.e., the number of columns in the feature matrix).

Hidden Layer : The second layer (Dense) has *20 neurons* and also uses the *ReLU* activation function. ReLU helps the model learn non-linear patterns in the data.

Output Layer : The final layer (Dense) consists of a *single neuron* with a *sigmoid activation function*. This setup is used for binary classification, where the output is a probability value between 0 and 1.

Model Compilation:

Loss Function : The model uses *binary_crossentropy* as the loss function, suitable for binary classification problems.

Optimizer: The *Adam* optimizer is employed, which is a popular optimizer for training deep learning models. It adapts the learning rate during training and is generally efficient.

Metrics : The model is trained for *100 epochs* with a *batch size of 15*.

Analysis of the Results

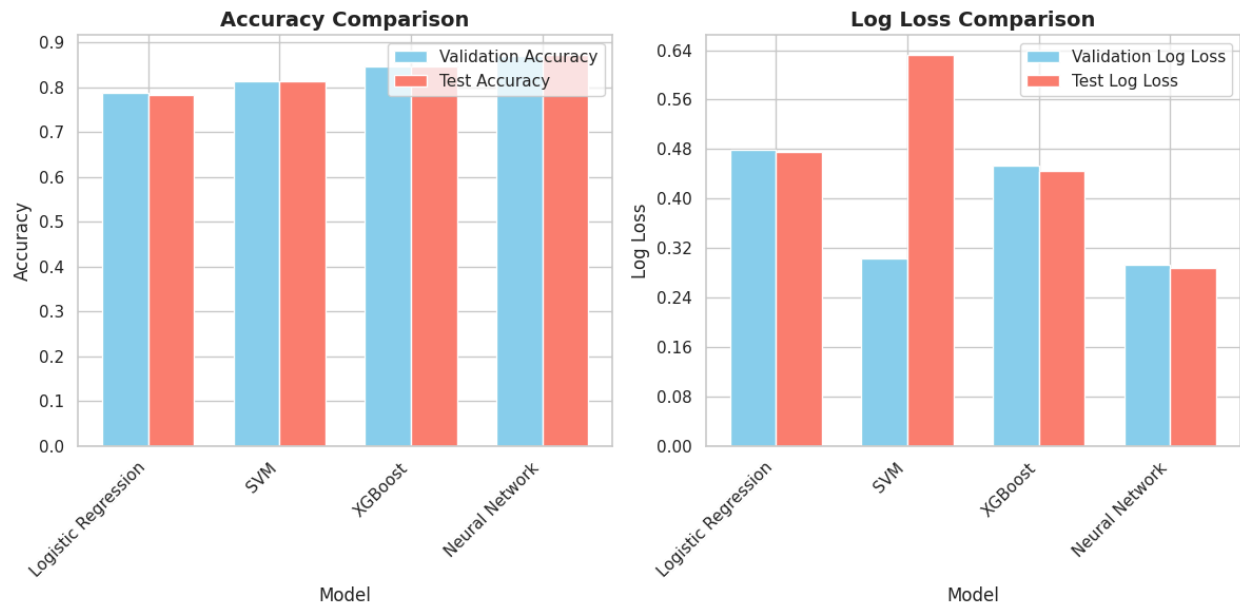
Model	Validation Accuracy
Logistic Regression	0.787
SVM	0.813
XGBoost	0.846
Neural Network	0.870

Model	Test Accuracy
Logistic Regression	0.781
SVM	0.813
XGBoost	0.846
Neural Network	0.872

The accuracy is quite close between validation and test sets, indicating that the model generalizes fairly well without significant overfitting or underfitting.

Model	Validation Log Loss
Logistic Regression	0.4782
SVM	0.3027
XGBoost	0.4538
Neural Network	0.2931

Model	Test Log Loss
Logistic Regression	0.4755
SVM	0.6327
XGBoost	0.4448
Neural Network	0.2883



Model Comparison:

Generalization: All models exhibit good generalization as evidenced by the close alignment between validation and test accuracy/log loss scores.

XGBoost and Neural Network outperform SVM and Logistic Regression in both accuracy and log loss on both the validation and test sets. Neural Network stands out as the best performer, with the highest accuracy and lowest log loss, suggesting it is the most accurate and reliable model for this task.

Conclusion

- **Neural Network performs best overall** in accuracy and confidence but has lower recall.
- **XGBoost is a close second** with **better recall but slightly lower accuracy**.
- **SVM is a strong alternative** to Logistic Regression, performing better in all aspects.
- **Logistic Regression is fast but lacks predictive power** compared to the other models.