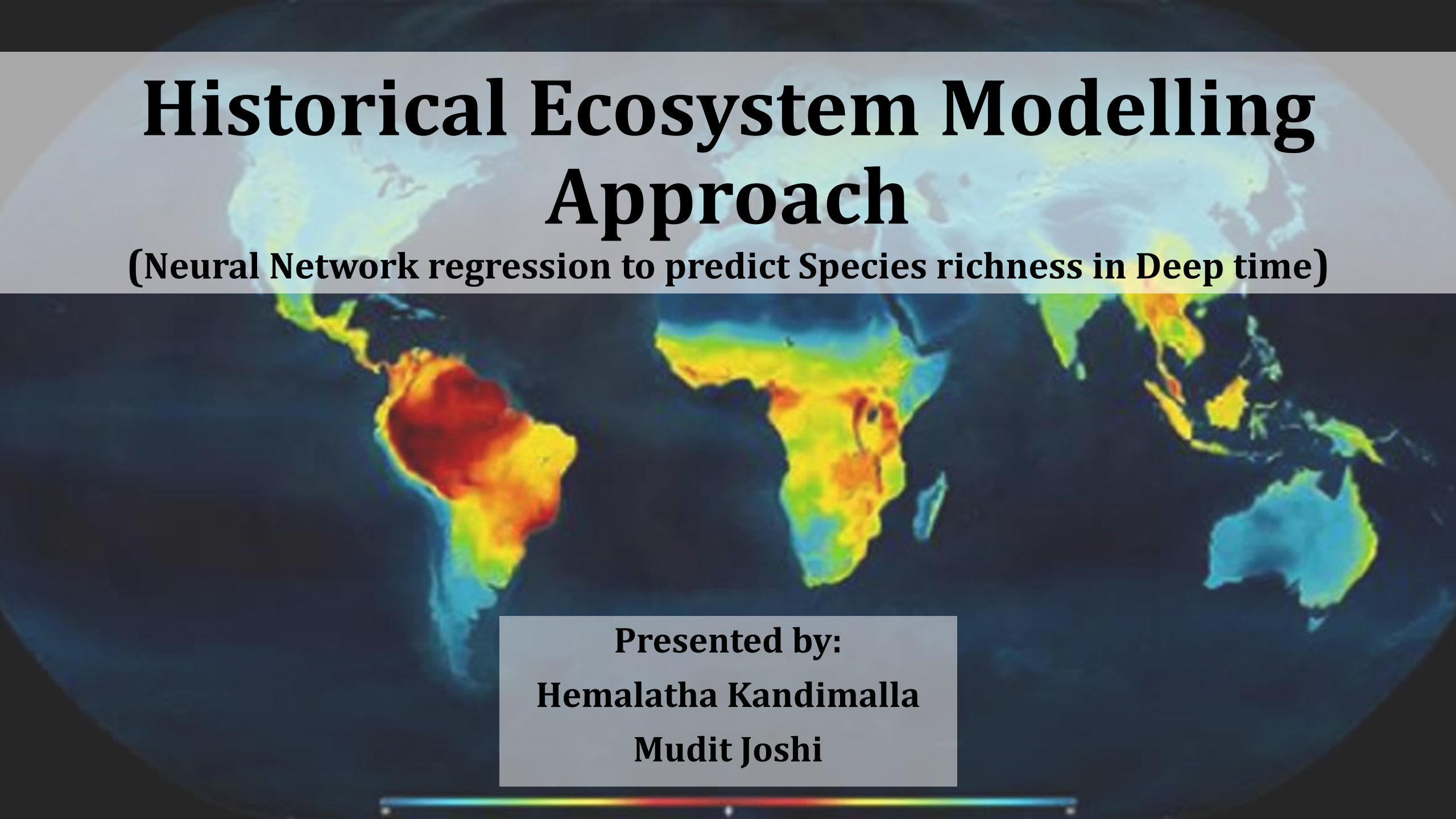
Historical Ecosystem Modelling Approach

(Neural Network regression to predict Species richness in Deep time)



Presented by:

Hemalatha Kandimalla

Mudit Joshi

Theoretical introduction

- The evolution of Earth's climate on geological timescales is largely driven by variations in the magnitude of **total solar irradiance (TSI)** and **changes in the greenhouse gas content** of the atmosphere. [1]
- Evolutionary Speed Hypothesis ESH predicts that **environmental kinetic energy** shapes variation in speciation rates through temperature- or life history-dependent rates of evolution. [2]

Terms to be used:

i. **Age of the Landscape**: in million years
ii. **Elevation**: Maximum elevation of earth, dynamic due to continent formations.
iii. **Global avg temperature**: average temperature of Earth's atmosphere over the entire surface, taking into account variations across regions and seasons.
iv. **pCO2** : partial pressure of carbon dioxide, is a measure of the concentration of carbon dioxide in Earth's atmosphere, representing the partial pressure contributed by this greenhouse gas.
v. **Species richness**: The count of distinct species in a particular ecosystem or geographical area.

1. Froster et. al 2017 https://doi.org/10.1038/ncomms14845
2. Skeels et. al. 2021 https://doi.org/10.1093/sysbio/syac048

Research question

To train a Neural network on a simulation dataset and using it to predict species richness in different hypothetical climate scenarios:

I – Maximum probability pCO2 (*Froster et. al 2017*) ---------- **Used for model training**

II – Median pCO2 constantly over time

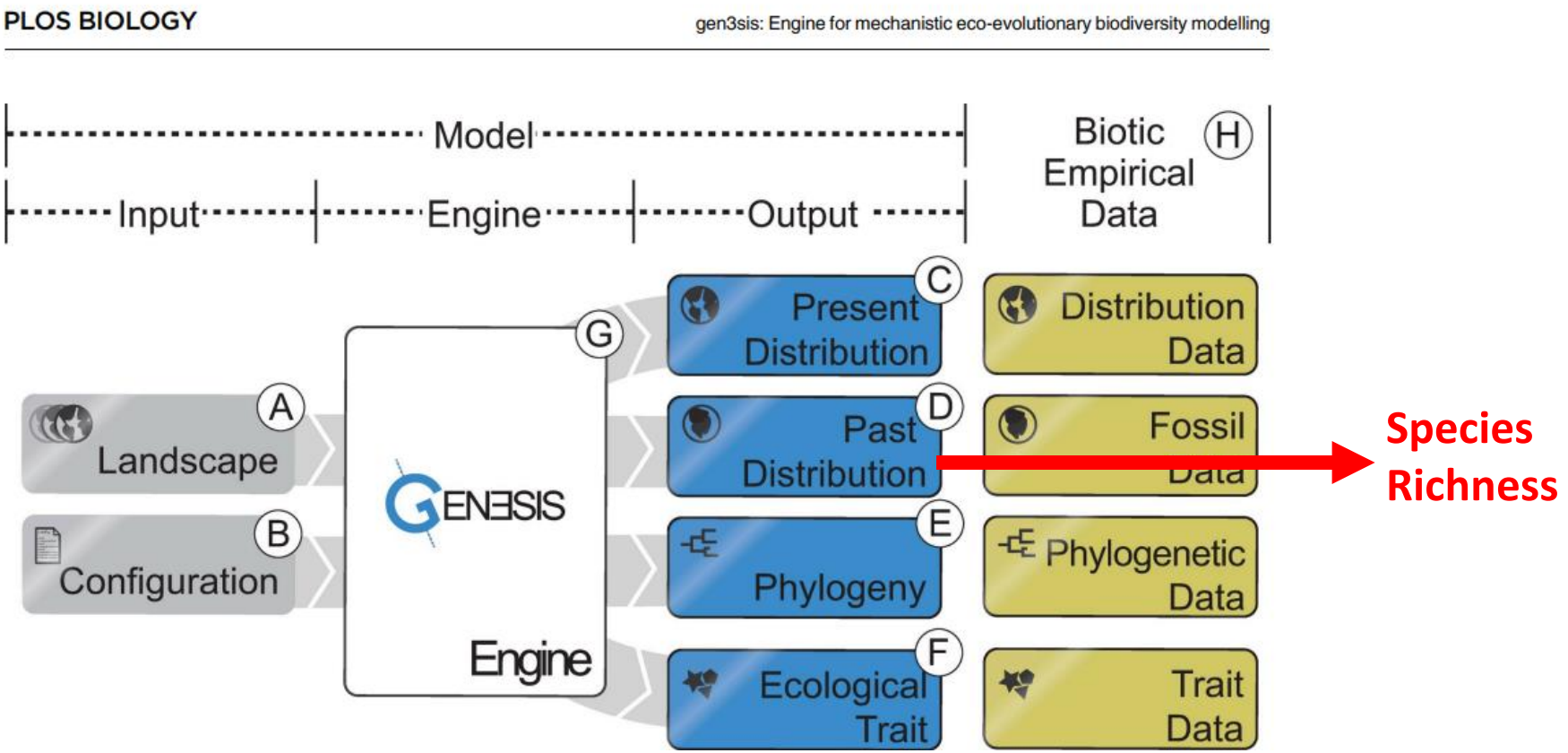
III – Mass extinction scenarios, with extreme peaks of pCO2

Objectives:

- Data import and preparation
- Visualization of Data, Selection of Features and Target Variable
- Making Baseline model (Regression Neural Network)
- Tuning Hyperparameters to build Robust model architecture.
- Using Model to predict Species Richness in new scenarios.

Data source

- It is our own data, generated as part of a masters project.
- It is simulated data generated using an R-based mechanistic model “GEN3SIS”.
- Complex model, to track the eco-evolution of configurated species over a given landscape reconstruction.



Hagen et al. 2021: https://doi.org/10.1371/journal.pbio.3001340

Data Characteristics

```
...  Maximum probability climate scenario
        Age        pCO2  Mean_Temperature    elevation  SR_total
     0  0.0  276.007628          2.125152  4913.789062      7021
     1  0.5  275.733739          2.204318  4869.667969      7055
     2  1.0  287.210267          2.298681  4861.992188      7490
     3  1.5  290.034080          2.365668  4854.316406      7326
     4  2.0  295.986206          2.436605  4846.640625      7781
     Median climate scenario
        Age        pCO2  Mean_Temperature    elevation
     0  0.0  738.907001          8.747786  4913.789062
     1  0.5  738.907001          8.714482  4869.667969
     2  1.0  738.907001          8.710369  4861.992188
     3  1.5  738.907001          8.685812  4854.316406
     4  2.0  738.907001          8.663997  4846.640625
     Climate scenario with Mass extinctions
        Age        pCO2  Mean_Temperature    elevation
     0  0.0  276.007628          2.154842  4913.789062
     1  0.5  275.733739          2.235443  4869.667969
     2  1.0  287.210267          2.333039  4861.992188
     3  1.5  290.034080          2.404725  4854.316406
     4  2.0  295.986206          2.481470  4846.640625
```

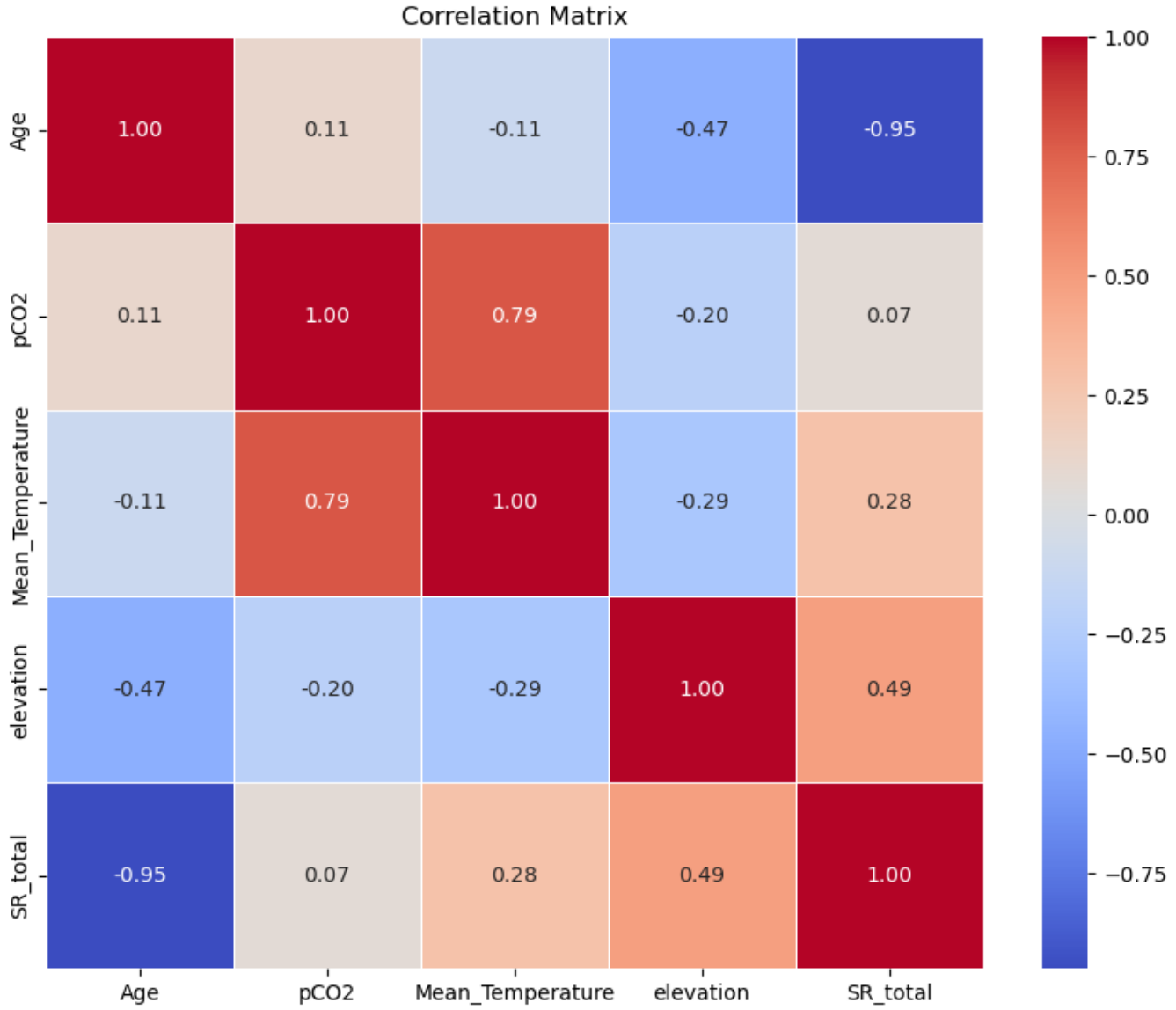
.

.

.

. Upto 801 rows,

Representing 400 myrs (at a timestep of 0.5myr)



Data Characteristics

```
...   Maximum probability climate scenario
      Age        pCO2  Mean_Temperature    elevation  SR_total
   0  0.0  276.007628          2.125152  4913.789062      7021
   1  0.5  275.733739          2.204318  4869.667969      7055
   2  1.0  287.210267          2.298681  4861.992188      7490
   3  1.5  290.034080          2.365668  4854.316406      7326
   4  2.0  295.986206          2.436605  4846.640625      7781
   Median climate scenario
      Age        pCO2  Mean_Temperature    elevation
   0  0.0  738.907001          8.747786  4913.789062
   1  0.5  738.907001          8.714482  4869.667969
   2  1.0  738.907001          8.710369  4861.992188
   3  1.5  738.907001          8.685812  4854.316406
   4  2.0  738.907001          8.663997  4846.640625
   Climate scenario with Mass extinctions
      Age        pCO2  Mean_Temperature    elevation
   0  0.0  276.007628          2.154842  4913.789062
   1  0.5  275.733739          2.235443  4869.667969
   2  1.0  287.210267          2.333039  4861.992188
   3  1.5  290.034080          2.404725  4854.316406
   4  2.0  295.986206          2.481470  4846.640625
```

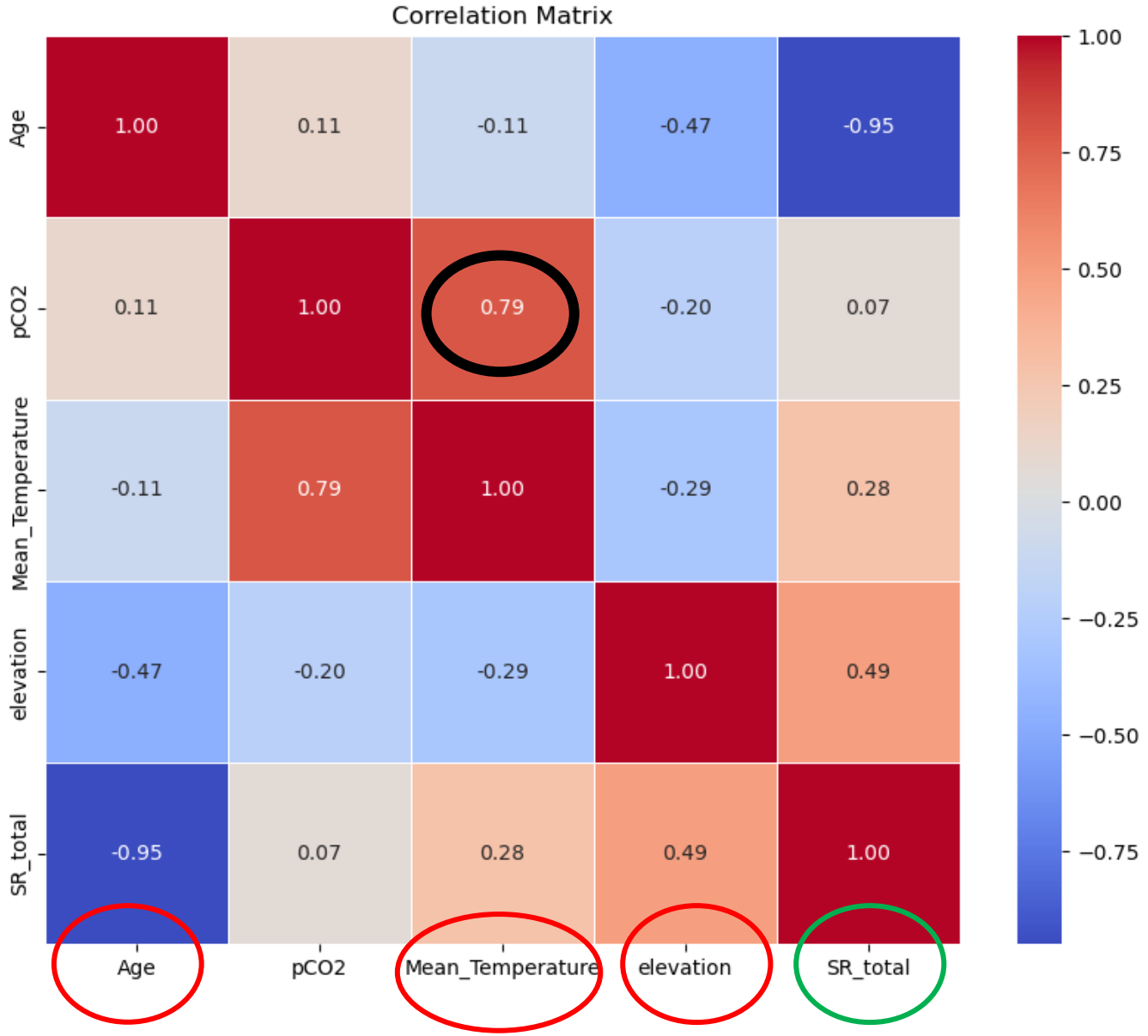
.

.

.

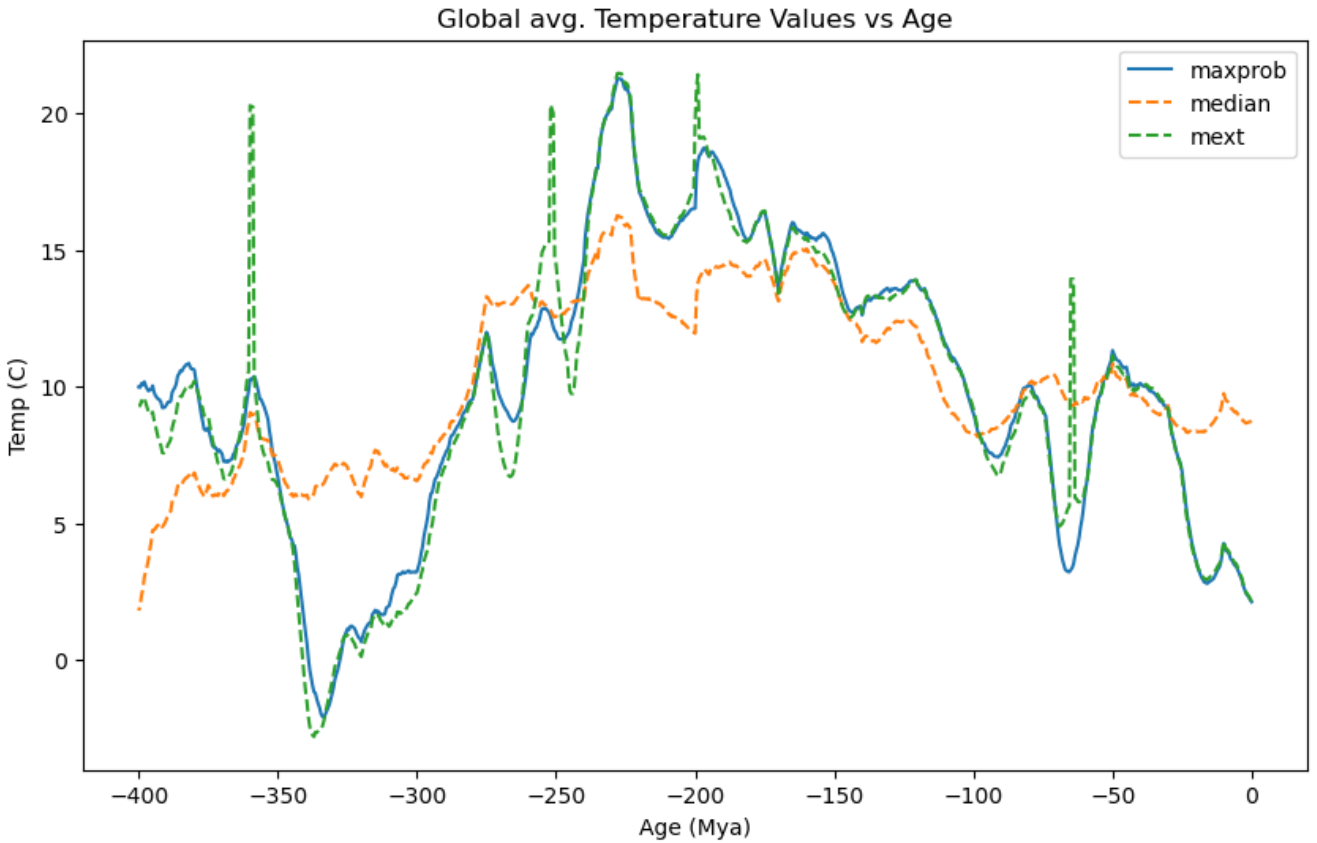
. Upto 801 rows,

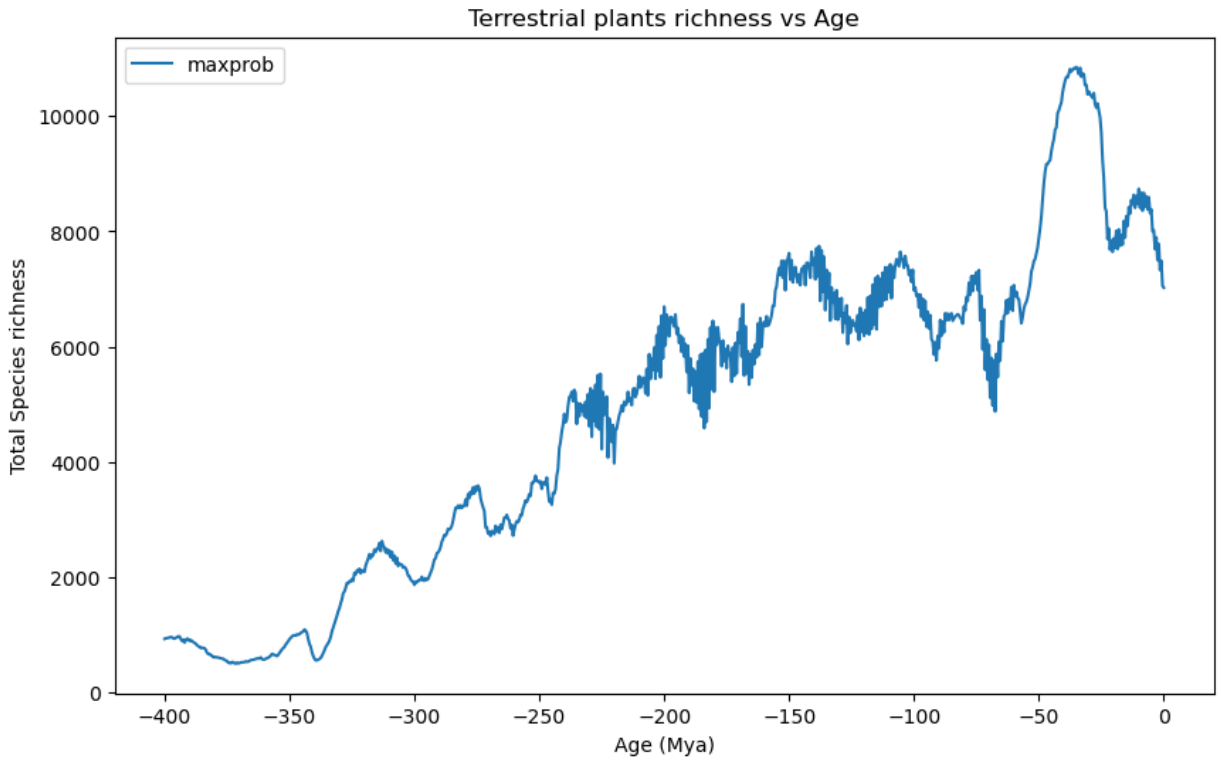
Representing 400 myrs (at a timestep of 0.5myr)



Target: Species richness

Features: Mean temp, Elevation & Age of landscape


Global avg. Temperature Values vs Age
maxprob
median
mext
Temp (C)
Age (Mya)
20
15
10
5
0
−400
−350
−300
−250
−200
−150
−100
−50
0


Terrestrial plants richness vs Age
maxprob
Total Species richness
Age (Mya)
10000
8000
6000
4000
2000
0
−400
−350
−300
−250
−200
−150
−100
−50
0

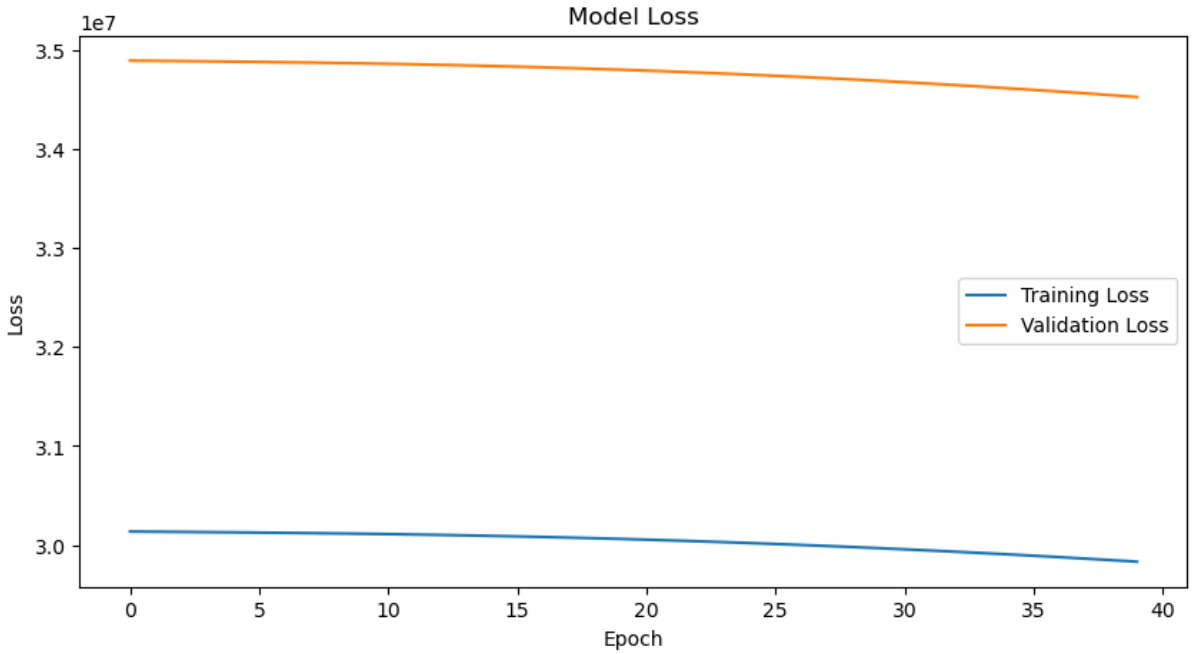
Baseline model

```
# Feature selection
#
#Setting seed (to avoid randomness in the model convergence)
SEED = 42
# Set seed for NumPy
np.random.seed(SEED)
# Set seed for TensorFlow
tf.random.set_seed(SEED)


# Setting Features and target variable (Age, pCO2, temp... are used to predict SR values)
features = maxprob[['Age', 'elevation', 'Mean_Temperature']]
target = maxprob['SR_total']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

#Normalize the data (to bring them into comparable numerical values)
def normalize(data):
    return (data - data.mean()) / data.std()

X_train = normalize(X_train)
X_test = normalize(X_test)
```

Baseline model

```
model_baseline = tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(1)  # Output layer with one neuron for species richness prediction
])
```

```
# model compilation
def r_squared(y_true, y_pred):
    SS_res = tf.reduce_sum(tf.square(y_true - y_pred))
    SS_tot = tf.reduce_sum(tf.square(y_true - tf.reduce_mean(y_true)))
    return 1 - (SS_res / (SS_tot + tf.keras.backend.epsilon()))

model_baseline.compile(optimizer='adam', loss='mean_squared_error', metrics=[r_squared])
```



Baseline model R-squared:
-2.5281901423494784

Loss function: Mean Squared Error (MSE)

- Differentiability: MSE is a differentiable function, which is crucial for the optimization algorithm to perform gradient descent during training.
- Optimization: Minimizing MSE during training corresponds to minimizing the squared differences between predicted and actual values. Which is a common optimization objective in regression tasks.

Evaluation metric: R-squared

- As our primary goal is to have a model that explains and predict a certain percentage of the variance in the target variable.
- It represents the proportion of the dependent variable's variance captured by the model.

While MSE guides the training process by specifying the loss function to be minimized, R-squared gives a more interpretable measure of model performance.

Model improvement: Adjusting learning rate of optimizer

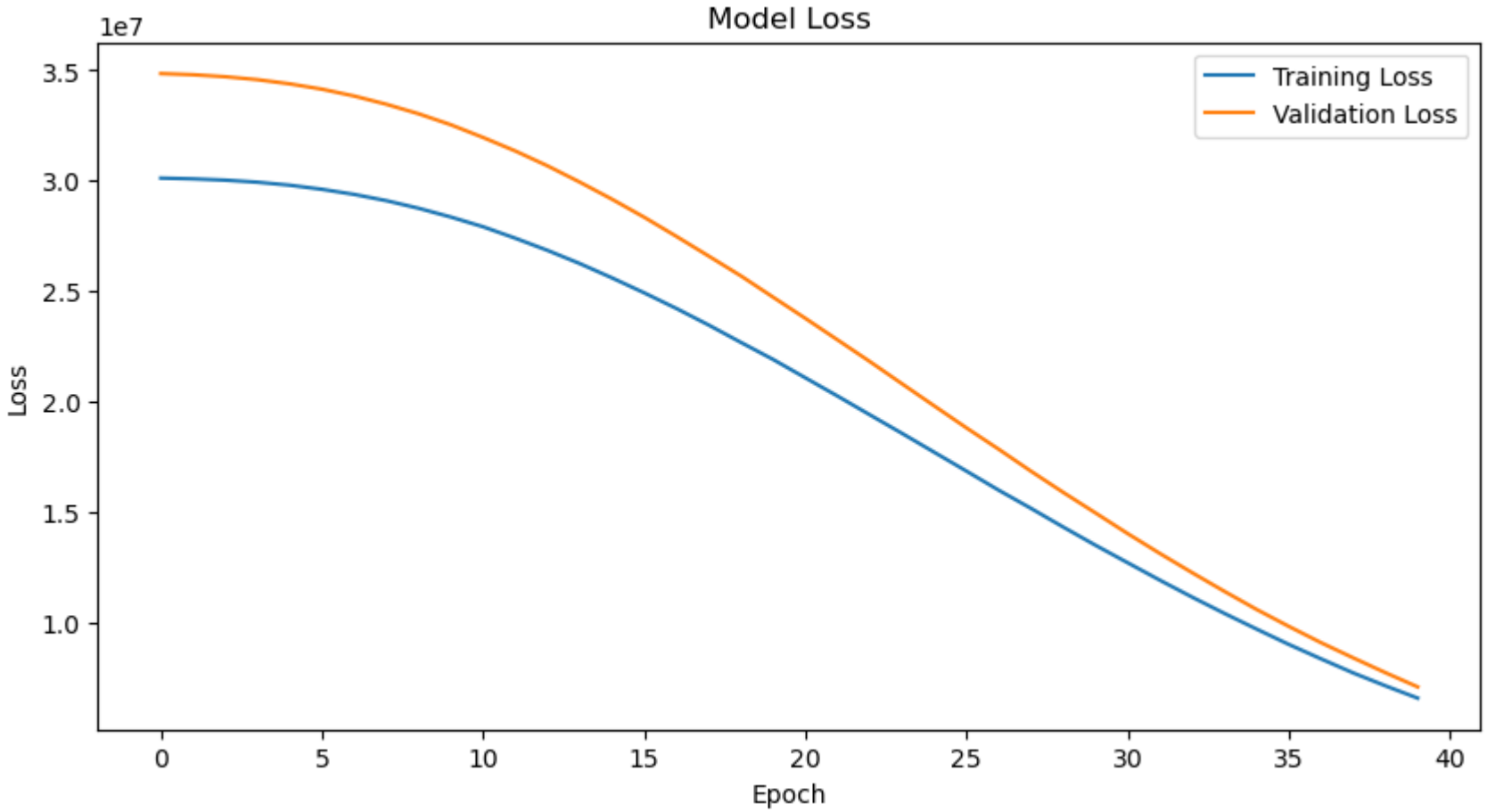
```python
model_baseline = tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(1)  # Output layer with one neuron for species richness prediction
])

# model compilation
def r_squared(y_true, y_pred):
    SS_res = tf.reduce_sum(tf.square(y_true - y_pred))
    SS_tot = tf.reduce_sum(tf.square(y_true - tf.reduce_mean(y_true)))
    return 1 - (SS_res / (SS_tot + tf.keras.backend.epsilon()))

#model_baseline.compile(optimizer='adam', loss='mean_squared_error', metrics=[r_squared])
model_baseline.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='mean_squared_error', metrics=[r_squared])

# Train the model
history = model_baseline.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.2, verbose=1)
```

- Learning rate determines the step size at each iteration during the optimization process.
  default (Adam optimizer) = 0.001.

- **higher learning rate can help the model better navigate turbulent gradients**

- It enables the model to jump over narrow valleys and move **towards more promising regions of the parameter space.**

- **Increases the exploration capability** of the optimization process.



Baseline model R-squared: 0.24428386560298

Model improvement: Adding more hidden layers

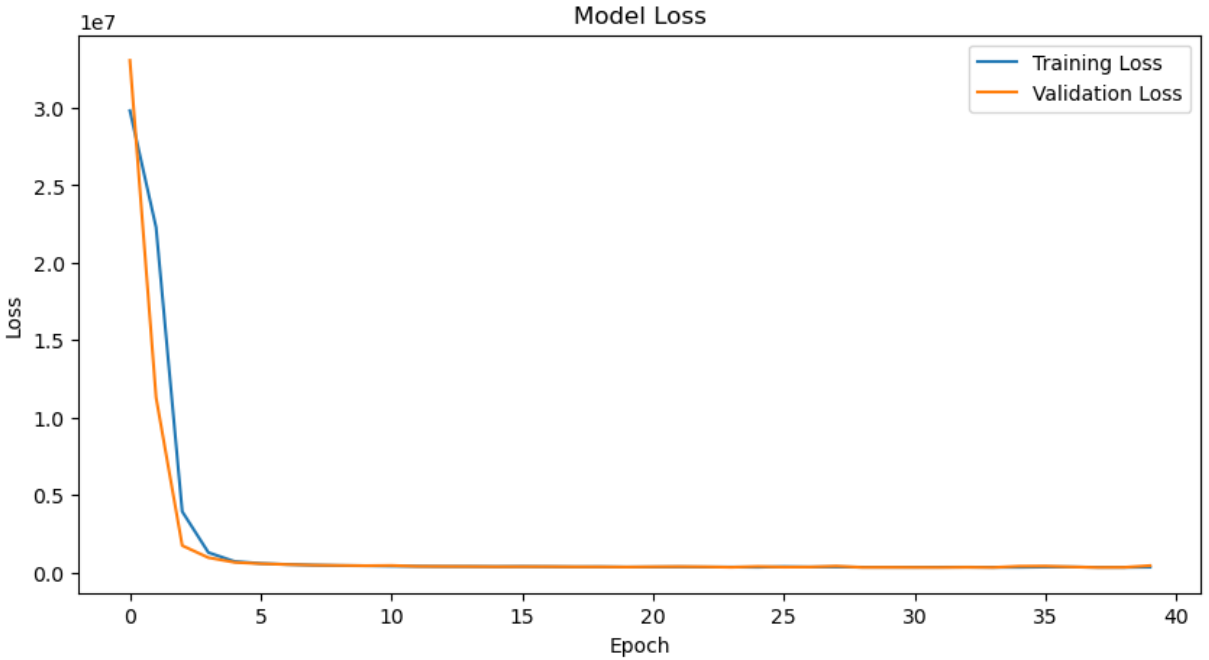
```python
# model 1
m1 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer with one neuron for species richness prediction
])

# Compile the model
m1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='mean_squared_error', metrics=[r_squared])

# Train the model
history = m1.fit(X_train, y_train, epochs=40, batch_size=32, validation_split=0.2, verbose=1)
```

Increased Model Capacity:

By adding more layers increases its capacity to capture complex patterns and relationships in the data. A more complex model can better represent the underlying structure of your regression problem.

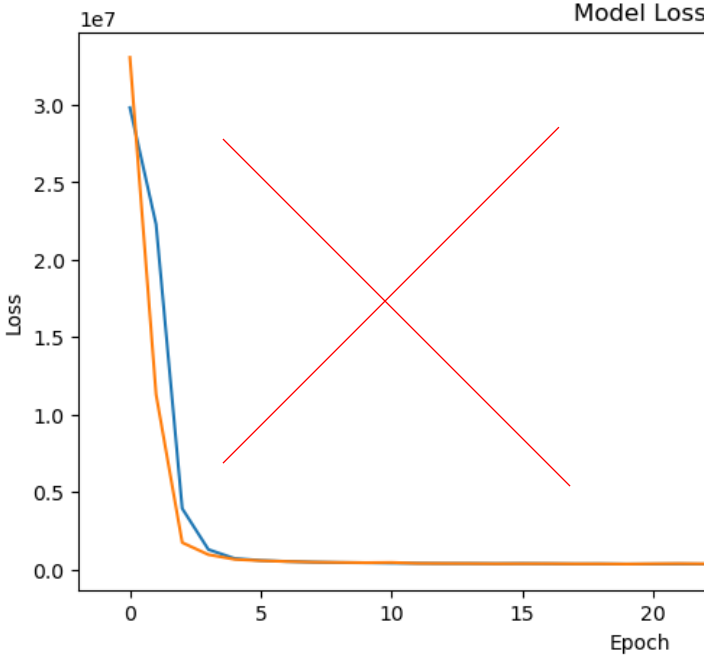
Non-Linearity and Improved Representation:

Introducing ReLU activation functions between layers allows the model to learn non-linear mappings, leading to better approximation of the target variable.

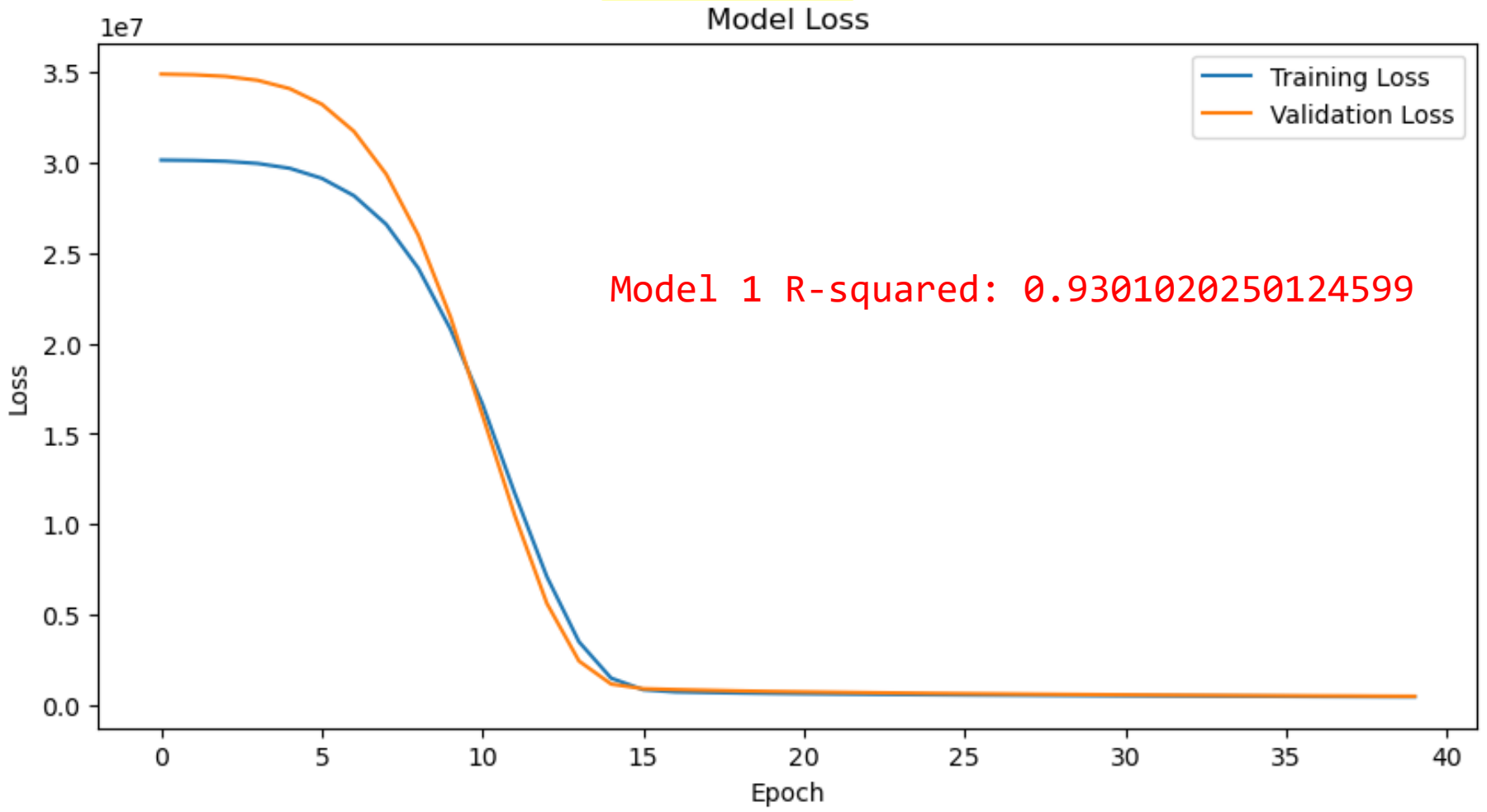


Model 1 R-squared: 0.9463048665325831

```python
# model 1
m1 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer with one neuron for species richness prediction
])

# Compile the model
m1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mean_squared_error', metrics=[r_squared])
```



Model 1 R-squared: 0.94636



Model 1 R-squared: 0.9301020250124599

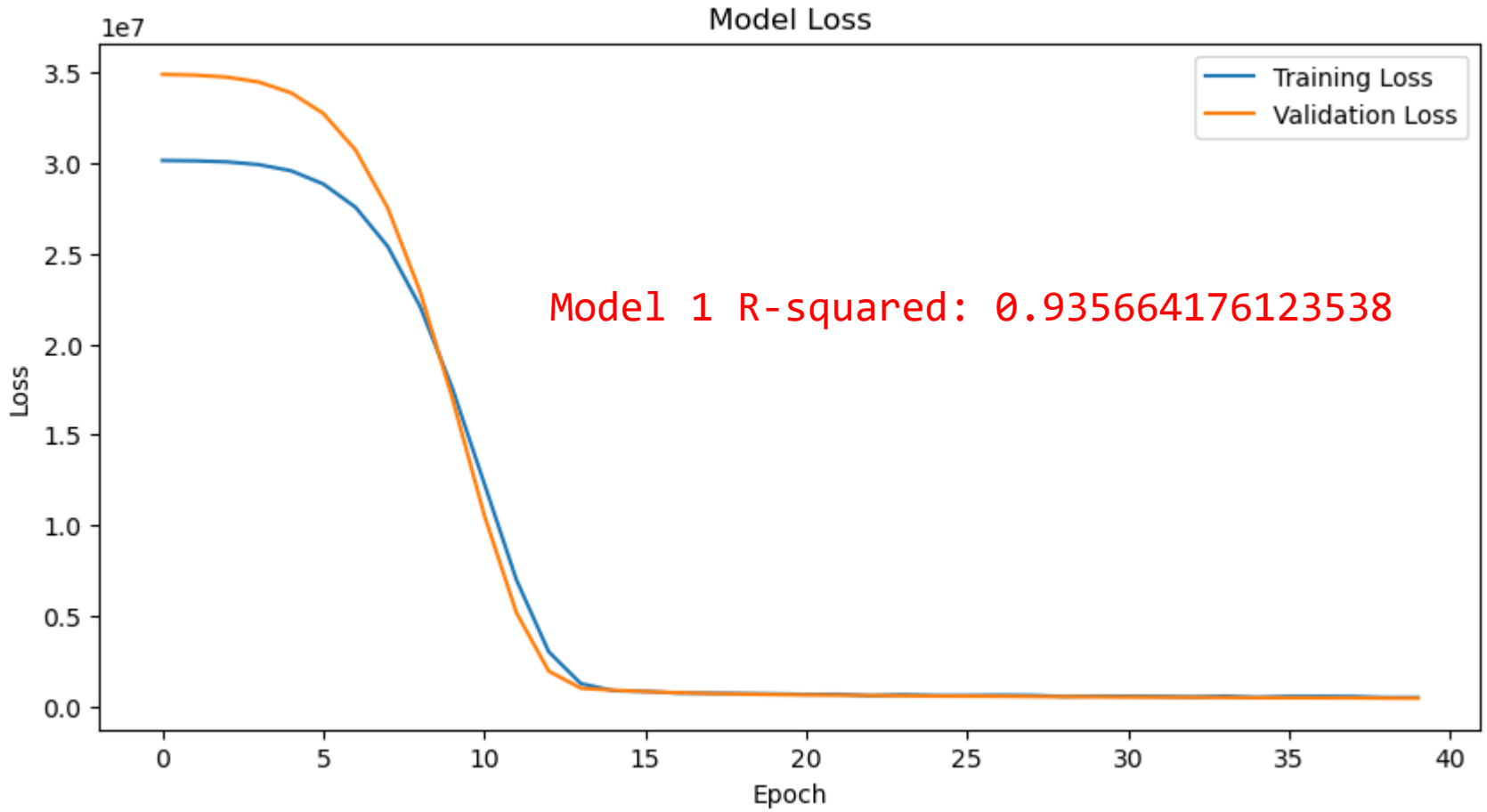
Model improvement: Adding Dropout layers ?

```python
# model 2
m2 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(3,)),
    tf.keras.layers.Dropout(0.2),  # Add dropout with a dropout rate (e.g., 0.2)
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer with one neuron for species richness prediction
])
```

- Dropout is a regularization technique that randomly drops (sets to zero) a fraction of input units during training.

- Helps prevent co-adaptation of neurons and **encourages the network to learn more robust** features. By randomly dropping units during training, **dropout introduces noise and reduces the risk of overfitting to the training data**.

  Dropout is often more beneficial when you have a **limited amount of training data**.

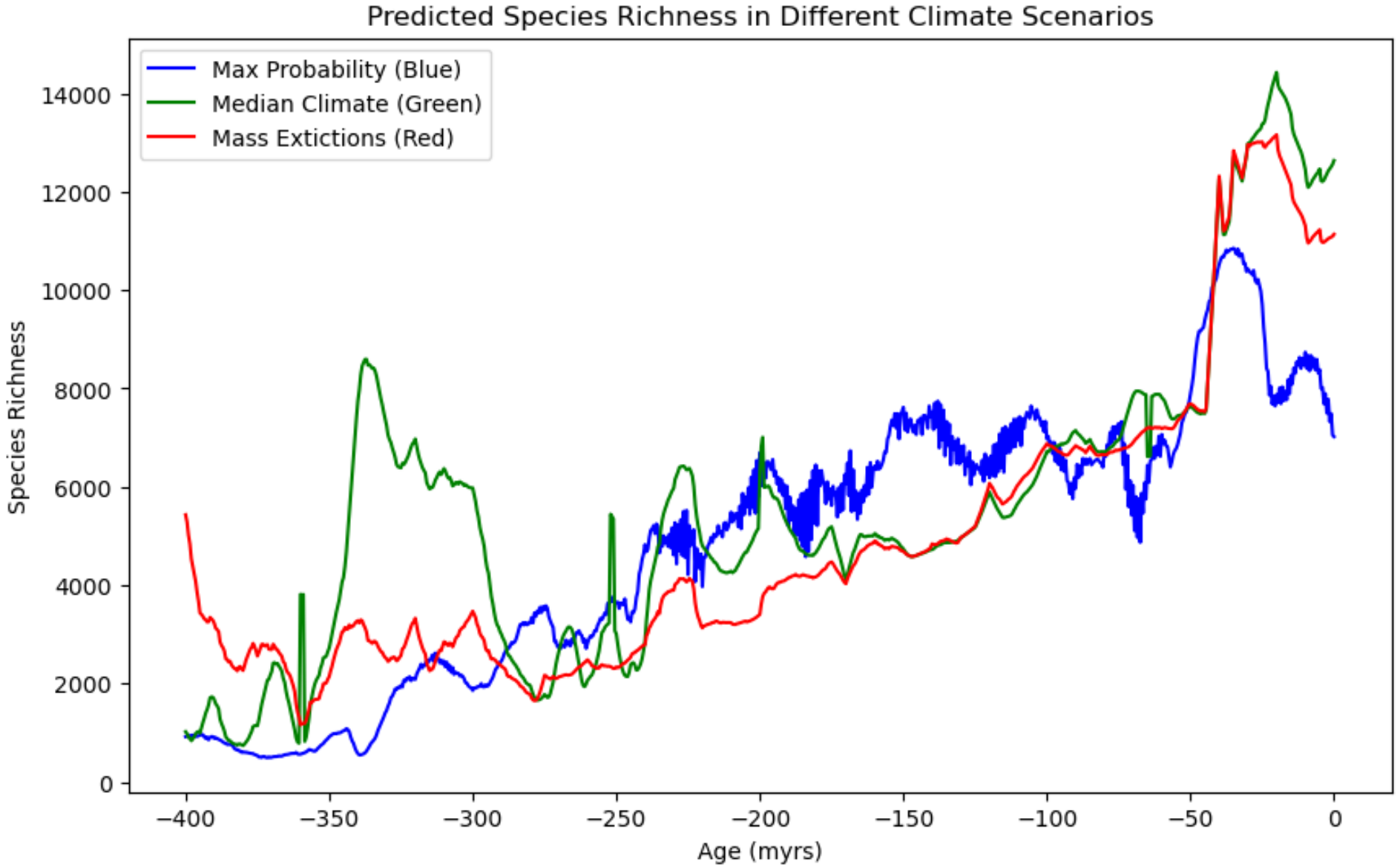


- Dropout tends to be **more effective in larger and more complex models**.
- As the regression model is relatively simple, the regularization provided by dropout **did not have a substantial impact**.

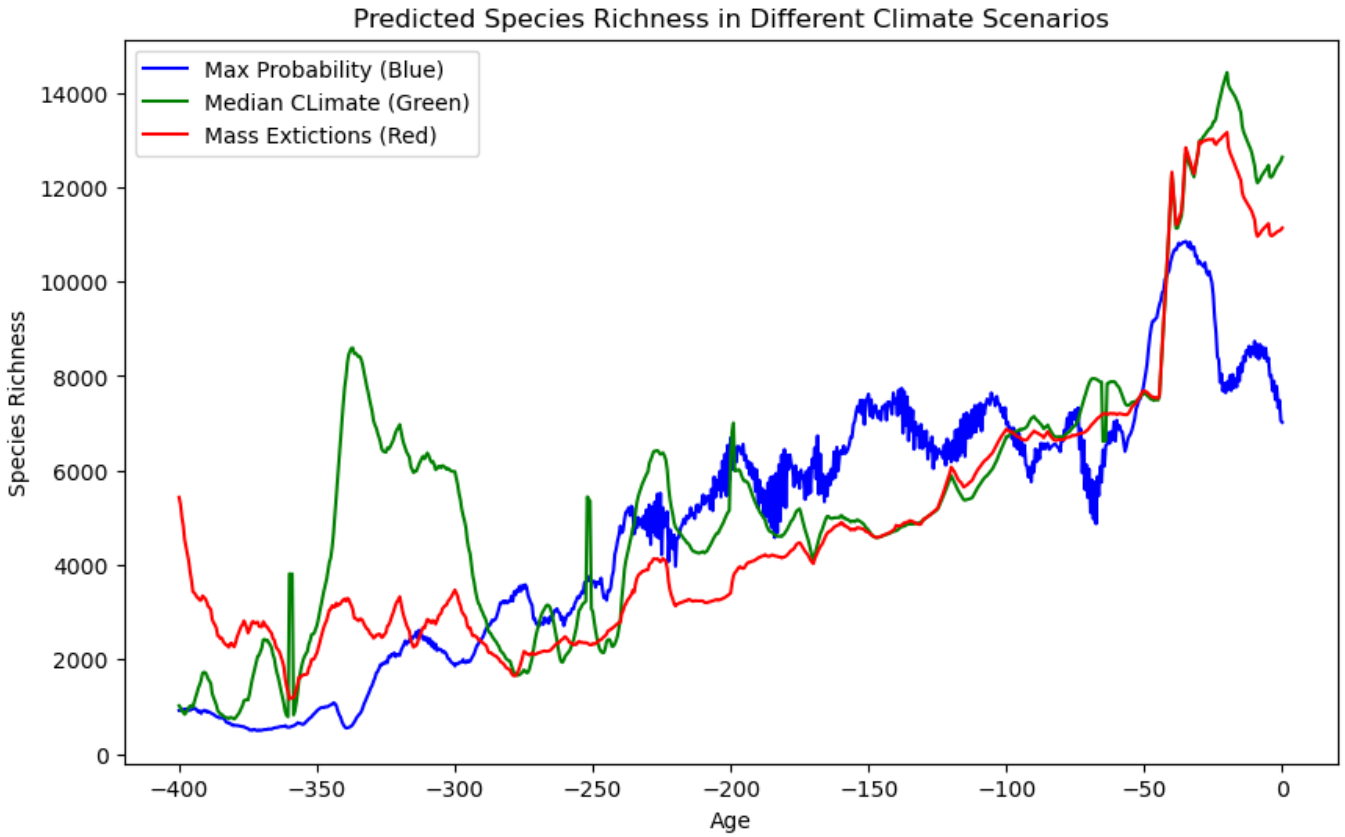
Making Predictions

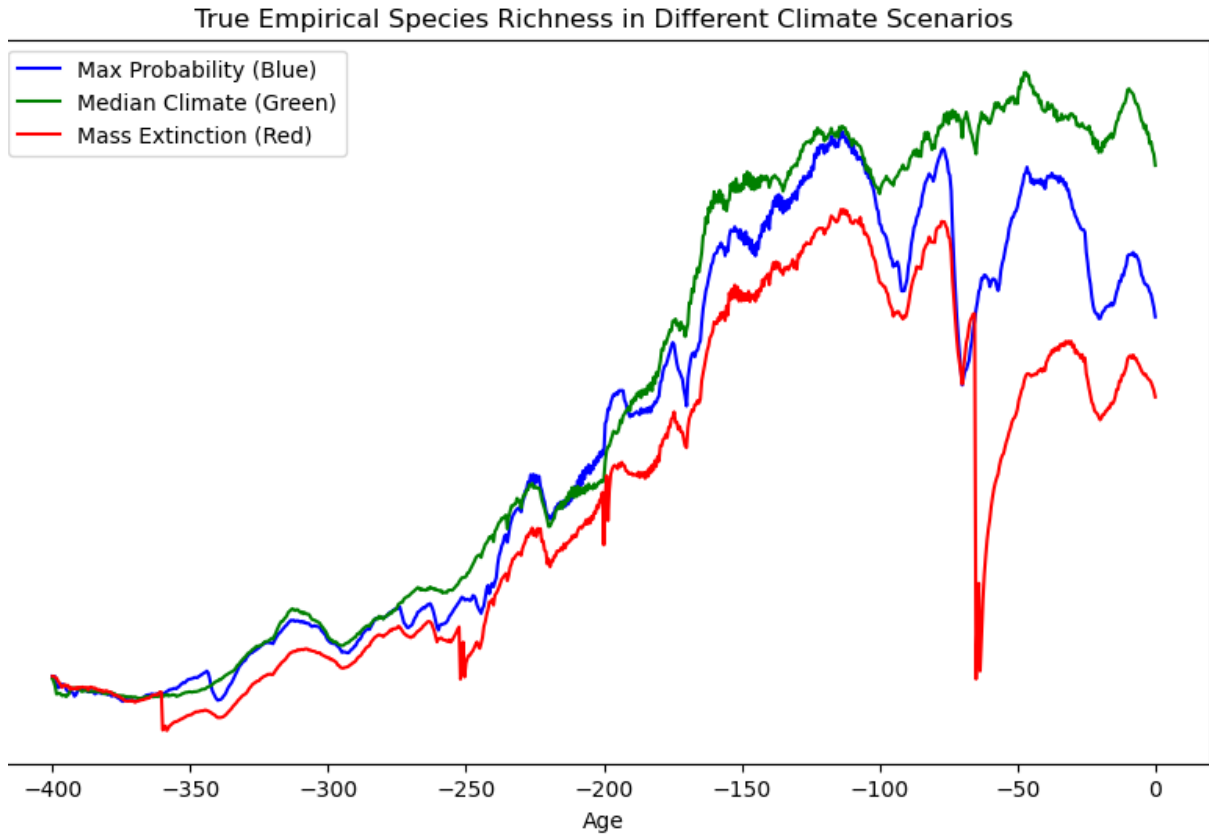
```
# making predictions for median and mext dataset
# features used during training
features_median = median[['Age', 'Mean_Temperature', 'elevation']]
features_mext = mext[['Age', 'Mean_Temperature', 'elevation']]

# Standardize the features using the same scaler used during training
scaler = StandardScaler()
features_median_scaled = scaler.fit_transform(features_median)
features_mext_scaled = scaler.transform(features_mext)

# Using trained model to predict species richness for each scenario
predictions_median = m2.predict(features_median_scaled)
predictions_mext = m2.predict(features_mext_scaled)
```

Making Predictions



Comparison with simulated data





Reason: Nature of data > Amount of Data

Learnings

- Lower epoch size: If you have a small dataset, training for too many epochs might lead to overfitting. We started with 70 epochs, finally used 40.

- it's not a strict rule that increasing the learning rate is only useful for basic models. The optimal learning rate depends on the specific characteristics of your model, dataset, and task. Experimenting is always a good idea!

- In a relatively simple regression model, with fewer parameters, the regularization provided by dropout, may have a less substantial impact compared to larger and more complex models.

Future prospects

- Input data which represent more complex parameters to represent the process of evolution more holistically, like speciation rate, size of lineage.
- More amount of data, a much shorter timestep than 0.5myrs will capture more bio-dynamic patterns of evolution of earth and it biodiversity.
- Use time-series analysis, to identify any specific trends during the course of evolution.
- Developing a neural network model to accurately answer such research question, can potentially save a lot of computing time usually taken by other modelling methods.

Thank you for your time!
(Questions?)