```
!pip install scikeras
!pip install spektral
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from spektral) (2.31.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from spektral) (1.5.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from spektral) (1.11.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from spektral) (4.66.4)
Requirement already satisfied: tensorflow>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from spektral) (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (24
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0-
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (0.2
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (0.2.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (24.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/pyth
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (1.14
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (1.64
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0->spektral) (
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow>=2.2.0-
Collecting keras<2.16,>=2.15.0 (from tensorflow>=2.2.0->spektral)
  Downloading keras-2.15.0-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 27.0 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->spektral) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->spektral) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->spektral) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->spektral) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->spektral) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->spektral) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->spektral) (2024.6.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->spektral) (3.5.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow>=2.2
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorf
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow>=2
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow>=2
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorbo
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboa
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5-
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-aut
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-
Installing collected packages: keras, spektral
  Attempting uninstall: keras
    Found existing installation: keras 3.3.3
    Uninstalling keras-3.3.3:
      Successfully uninstalled keras-3.3.3
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sourc
scikeras 0.13.0 requires keras>=3.2.0, but you have keras 2.15.0 which is incompatible.
Successfully installed keras-2.15.0 spektral-1.3.1
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
import xgboost as xgb
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout, GRU, SimpleRNN, Conv1D, MaxPooling1D, Flatten
from keras.layers import Input, Attention
from scikeras.wrappers import KerasRegressor
from sklearn.metrics import mean_squared_error
import tensorflow_probability as tfp
import networkx as nx
from spektral.layers import GCNConv
from keras.models import Model
import tensorflow as tf
import time



data = pd.read_csv("/content/sample_data/output.csv")

# Preprocess the dataset
data = data.drop(columns=["Name"])
data_encoded = pd.get_dummies(data, columns=["Initial Continent", "Initial Climate", "Final Continent", "Final Climate"])

X = data_encoded.drop(columns=["Impact"])
y = data_encoded[["Impact"]]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

data.head()
```

	Skin Tone	Initial Continent	Initial Climate	Initial Radiations	Initial Oxygen	Initial Nitrogen	Final Continent	Final Climate	Final Radiations	Final Oxygen	Final Nitrogen	Duration	Impact
0	1	Asia	Cold	1	0.79	0.2	Europe	Cold	1	0.79	0.2	10	0
1	2	Asia	Cold	1	0.79	0.2	Europe	Cold	1	0.79	0.2	6	0
2	3	Asia	Cold	1	0.79	0.2	Europe	Cold	1	0.79	0.2	9	0
3	4	Asia	Cold	1	0.79	0.2	Europe	Temperate	1	0.79	0.2	3	0
4	5	Asia	Cold	1	0.79	0.2	Europe	Temperate	1	0.79	0.2	5	0

```
    def linear_regression(X_train, y_train, X_test):
        model = LinearRegression()
        model.fit(X_train, y_train)
        return model.predict(X_test), None

    def polynomial_regression(X_train, y_train, X_test, degree=2):
        poly = PolynomialFeatures(degree=degree)
        X_train_poly = poly.fit_transform(X_train)
        X_test_poly = poly.transform(X_test)
        model = LinearRegression()
        model.fit(X_train_poly, y_train)
        return model.predict(X_test_poly), None

    def svr(X_train, y_train, X_test):
        model = SVR(kernel='rbf')
        model.fit(X_train, y_train.values.ravel())
        return model.predict(X_test), None

    def decision_tree(X_train, y_train, X_test):
        model = DecisionTreeRegressor(random_state=42)
        model.fit(X_train, y_train)
        return model.predict(X_test), None

    def random_forest(X_train, y_train, X_test):
        model = RandomForestRegressor(random_state=42)
        model.fit(X_train, y_train.values.ravel())
        return model.predict(X_test), None  # Return predictions and None for history

    def gradient_boosting(X_train, y_train, X_test):
        model = GradientBoostingRegressor(n_estimators=100, random_state=42)
        model.fit(X_train, y_train.values.ravel())
        return model.predict(X_test), None

    def xgboost(X_train, y_train, X_test):
        model = xgb.XGBRegressor(n_estimators=100, random_state=42)
        model.fit(X_train, y_train.values.ravel())
        return model.predict(X_test), None

    def adaboost(X_train, y_train, X_test):
        model = AdaBoostRegressor(n_estimators=100, random_state=42)
        model.fit(X_train, y_train.values.ravel())
        return model.predict(X_test), None
```

```
def lstm(X_train, y_train, X_test):
    X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
    X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(1, X_train_scaled.shape[1])))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))  # Output layer with 1 neuron for Impact
    model.compile(optimizer='adam', loss='mean_squared_error')
    history = model.fit(X_train_reshaped, y_train, validation_data=(X_test_reshaped, y_test), epochs=100, batch_size=32, verbose=1)
    return model.predict(X_test_reshaped), history


def nn(X_train, y_train, X_test):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        Dense(64, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, batch_size=32, verbose=1)
    return model.predict(X_test), history



def build_cnn():
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(X_train_scaled.shape[1], 1)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model



def rnn(X_train, y_train, X_test):
    # Reshape the input data to be 3D: (samples, timesteps, features)
    X_train_reshaped = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
    X_test_reshaped = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

    model = Sequential([
        SimpleRNN(64, input_shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2]), activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    history = model.fit(X_train_reshaped, y_train, validation_data=(X_test_reshaped, y_test), epochs=100, batch_size=32, verbose=1)
    return model.predict(X_test_reshaped), history



def gru(X_train, y_train, X_test):
    print("X_train_scaled shape:", X_train.shape)
    print("X_test_scaled shape:", X_test.shape)
    # Build the GRU model
    model = Sequential()
    model.add(GRU(units=50, activation='relu', input_shape=(X_train_reshaped.shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(Dense(units=1))  # Output layer with 1 neuron for Impact

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Train the model
    history = model.fit(X_train_reshaped, y_train, epochs=300, batch_size=32, validation_data=(X_test_reshaped, y_test), verbose=1)
    return model.predict(X_test), history




def build_autoencoder():
    input_dim = X_train_scaled.shape[1]
```

```
    encoding_dim = 32

    input_layer = Input(shape=(input_dim,))
    encoded = Dense(encoding_dim, activation='relu')(input_layer)
    decoded = Dense(input_dim, activation='sigmoid')(encoded)

    autoencoder = Model(input_layer, decoded)
    autoencoder.compile(optimizer='adam', loss='mean_squared_error')
    return autoencoder


def autoencoder(X_train, y_train, X_test):
    autoencoder = build_autoencoder()
    history = autoencoder.fit(X_train, X_train, epochs=100, batch_size=32, verbose=1, validation_data=(X_test, X_test))
    encoder = Model(autoencoder.input, autoencoder.layers[-2].output)
    X_train_encoded = encoder.predict(X_train)
    X_test_encoded = encoder.predict(X_test)
    model = LinearRegression()
    model.fit(X_train_encoded, y_train)
    return model.predict(X_test_encoded), history


def build_gnn():
    input_shape = X_train_scaled.shape[1]
    X_in = Input(shape=(input_shape,))
    A_in = Input(shape=(None,))
    X_1 = GCNConv(32, activation='relu')([X_in, A_in])
    X_2 = GCNConv(1)([X_1, A_in])
    model = Model(inputs=[X_in, A_in], outputs=X_2)
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model


def gnn(X_train, y_train, X_test):
    A = np.eye(X_train.shape[1])
    model = build_gnn()
    history = model.fit([X_train_scaled, A], y_train, epochs=100, batch_size=32, verbose=1, validation_data=([X_test_scaled, A], y_test))
    return model.predict([X_test_scaled, A]), history


def build_attention():
    input_shape = X_train_scaled.shape[1]
    input_layer = Input(shape=(input_shape,))
    attention_data = Attention()([input_layer, input_layer])
    dense_layer = Dense(64, activation='relu')(attention_data)
    output_layer = Dense(1)(dense_layer)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model


def attention(X_train, y_train, X_test):
    model = KerasRegressor(build_fn=build_attention, epochs=100, batch_size=32, verbose=1)
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test))
    return model.predict(X_test), history


# Function to save results incrementally
def save_results(results):
    results_df = pd.DataFrame(results)
    results_df.to_csv("/content/sample_data/model_results.csv", index=False)
    print("Results saved to model_results.csv")
```

```
# Loop through each model and save results incrementally
results = []

models = {
    'Linear Regression': linear_regression,
    'Polynomial Regression': lambda X_train, y_train, X_test: polynomial_regression(X_train, y_train, X_test, degree=2),
    'Support Vector Regression': svr,
    'Decision Tree': decision_tree,
    'Random Forest': random_forest,
    'Gradient Boosting': gradient_boosting,
    'XGBoost': xgboost,
    'AdaBoost': adaboost,
    'LSTM': lstm,
    'Neural Network': nn,
    'Recurrent Neural Network': rnn,
    'GRU': gru,
    'Autoencoder': autoencoder,
    'Graph Neural Network': gnn,
    'Attention Mechanism': attention
}

for name, model_func in models.items():
    print(f"Training {name}...")
    start_time = time.time()
    predictions, history = model_func(X_train_scaled, y_train, X_test_scaled)
    mse = mean_squared_error(y_test, predictions)
    end_time = time.time()
    result = {'Model': name, 'Test Loss (MSE)': mse, 'Training Time (s)': end_time - start_time}
    if history:
        result['Training Loss'] = history.history['loss'][-1]
        result['Validation Loss'] = history.history['val_loss'][-1]
    results.append(result)
    save_results(results)

print("All models have been trained and results saved incrementally to model_results.csv.")
```

```
Training Linear Regression...
Results saved to model_results.csv
Training Polynomial Regression...
Results saved to model_results.csv
Training Support Vector Regression...
Results saved to model_results.csv
Training Decision Tree...
Results saved to model_results.csv
Training Random Forest...
Results saved to model_results.csv
Training Gradient Boosting...
Results saved to model_results.csv
Training XGBoost...
Results saved to model_results.csv
Training AdaBoost...
Results saved to model_results.csv
Training LSTM...
Epoch 1/100
25/25 [==============================] - 15s 59ms/step - loss: 1.8545 - val_loss: 1.8488
Epoch 2/100
25/25 [==============================] - 0s 6ms/step - loss: 1.4358 - val_loss: 1.0712
Epoch 3/100
25/25 [==============================] - 0s 6ms/step - loss: 0.6417 - val_loss: 0.5747
Epoch 4/100
25/25 [==============================] - 0s 7ms/step - loss: 0.3996 - val_loss: 0.4626
Epoch 5/100
25/25 [==============================] - 0s 6ms/step - loss: 0.3388 - val_loss: 0.4018
Epoch 6/100
25/25 [==============================] - 0s 7ms/step - loss: 0.2974 - val_loss: 0.3632
Epoch 7/100
25/25 [==============================] - 0s 7ms/step - loss: 0.2718 - val_loss: 0.3328
Epoch 8/100
25/25 [==============================] - 0s 7ms/step - loss: 0.2435 - val_loss: 0.3089
Epoch 9/100
25/25 [==============================] - 0s 7ms/step - loss: 0.2257 - val_loss: 0.2892
Epoch 10/100
25/25 [==============================] - 0s 7ms/step - loss: 0.2144 - val_loss: 0.2696
Epoch 11/100
25/25 [==============================] - 0s 7ms/step - loss: 0.2077 - val_loss: 0.2650
Epoch 12/100
25/25 [==============================] - 0s 7ms/step - loss: 0.1857 - val_loss: 0.2523
Epoch 13/100
25/25 [==============================] - 0s 7ms/step - loss: 0.1748 - val_loss: 0.2384
Epoch 14/100
```

```
25/25 [==============================] - 0s 6ms/step - loss: 0.1608 - val_loss: 0.2347
Epoch 15/100
25/25 [==============================] - 0s 7ms/step - loss: 0.1622 - val_loss: 0.2322
Epoch 16/100
25/25 [==============================] - 0s 6ms/step - loss: 0.1753 - val_loss: 0.2212
Epoch 17/100
25/25 [==============================] - 0s 6ms/step - loss: 0.1635 - val_loss: 0.2104
Epoch 18/100
25/25 [==============================] - 0s 7ms/step - loss: 0.1537 - val_loss: 0.2204
Epoch 19/100
25/25 [==============================] - 0s 7ms/step - loss: 0.1541 - val_loss: 0.2050
Epoch 20/100
25/25 [==============================] - 0s 7ms/step - loss: 0.1358 - val_loss: 0.2104
Epoch 21/100
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential, Model
from keras.layers import LSTM, Dense, Dropout, GRU, SimpleRNN, Bidirectional, Input, Dot, Activation, Reshape
from keras.optimizers import Adam
import xgboost as xgb
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
# Add this import at the top of your 'ipython-input-1-4efe37c45b5f' file
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor

# Preprocessing function
def preprocess_data(data, target_column, categorical_columns):
    data = data.drop(columns=["Name"])  # Drop unnecessary columns like Name
    data_encoded = pd.get_dummies(data, columns=categorical_columns)
    X = data_encoded.drop(columns=[target_column])
    y = data_encoded[[target_column]]
    return X, y

# Splitting and scaling function
def split_and_scale(X, y, test_size=0.2, random_state=42):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    return X_train_scaled, X_test_scaled, y_train, y_test

# Machine Learning Models
def linear_regression_model(X_train, X_test, y_train, y_test):
    model = LinearRegression()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"Linear Regression MSE: {mse}")
    return model, predictions

def svr_model(X_train, X_test, y_train, y_test):
    model = SVR()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"SVR MSE: {mse}")
    return model, predictions

def decision_tree_model(X_train, X_test, y_train, y_test):
    model = DecisionTreeRegressor()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"Decision Tree Regressor MSE: {mse}")
    return model, predictions

def random_forest_model(X_train, X_test, y_train, y_test):
    model = RandomForestRegressor()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"Random Forest Regressor MSE: {mse}")
    return model, predictions

def xgboost_model(X_train, X_test, y_train, y_test):
    model = xgb.XGBRegressor()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"XGBoost Regressor MSE: {mse}")
    return model, predictions

def adaboost_model(X_train, X_test, y_train, y_test):
    model = AdaBoostRegressor()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
```

```
        print(f"AdaBoost Regressor MSE: {mse}")
        return model, predictions

    def gradient_boosting_model(X_train, X_test, y_train, y_test):
        model = GradientBoostingRegressor()
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print(f"Gradient Boosting Regressor MSE: {mse}")
        return model, predictions

    # Deep Learning Models
    def lstm_model(X_train, X_test, y_train, y_test):
        model = Sequential()
        model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
        model.add(Dropout(0.2))
        model.add(LSTM(units=50))
        model.add(Dropout(0.2))
        model.add(Dense(1))
        model.compile(optimizer=Adam(), loss='mean_squared_error')
        model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print(f"LSTM Model MSE: {mse}")
        return model, predictions

    def bi_lstm_model(X_train, X_test, y_train, y_test):
        model = Sequential()
        model.add(Bidirectional(LSTM(units=50, return_sequences=True), input_shape=(X_train.shape[1], X_train.shape[2])))
        model.add(Dropout(0.2))
        model.add(Bidirectional(LSTM(units=50)))
        model.add(Dropout(0.2))
        model.add(Dense(1))
        model.compile(optimizer=Adam(), loss='mean_squared_error')
        model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print(f"Bi-LSTM Model MSE: {mse}")
        return model, predictions

    def gru_model(X_train, X_test, y_train, y_test):
        model = Sequential()
        model.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
        model.add(Dropout(0.2))
        model.add(GRU(units=50))
        model.add(Dropout(0.2))
        model.add(Dense(1))
        model.compile(optimizer=Adam(), loss='mean_squared_error')
        model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print(f"GRU Model MSE: {mse}")
        return model, predictions

    def rnn_model(X_train, X_test, y_train, y_test):
        model = Sequential()
        model.add(SimpleRNN(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
        model.add(Dropout(0.2))
        model.add(SimpleRNN(units=50))
        model.add(Dropout(0.2))
        model.add(Dense(1))
        model.compile(optimizer=Adam(), loss='mean_squared_error')
        model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
        predictions = model.predict(X_test)
        mse = mean_squared_error(y_test, predictions)
        print(f"RNN Model MSE: {mse}")
        return model, predictions

    def attention_model(X_train, X_test, y_train, y_test):
        input_layer = Input(shape=(X_train.shape[1], X_train.shape[2]))
        lstm_out = LSTM(50, return_sequences=True)(input_layer)
        attention_score = Dense(1, activation='softmax')(lstm_out)
        attention_mul = Dot(axes=1)([attention_score, lstm_out])
        attention_output = Reshape((X_train.shape[2],))(attention_mul)
        output_layer = Dense(1)(attention_output)
        model = Model(inputs=input_layer, outputs=output_layer)
        model.compile(optimizer=Adam(), loss='mean_squared_error')
```

```
    model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"Attention Model MSE: {mse}")
    return model, predictions



# Load data
data = pd.read_csv("/content/sample_data/output.csv")

# Preprocess data
categorical_columns = ["Initial Continent", "Initial Climate", "Final Continent", "Final Climate"]
X, y = preprocess_data(data, target_column="Impact", categorical_columns=categorical_columns)

# Split and scale data
X_train_scaled, X_test_scaled, y_train, y_test = split_and_scale(X, y)

# Train and evaluate models
models = {
    "Linear Regression": linear_regression_model,
    "SVR": svr_model,
    "Decision Tree": decision_tree_model,
    "Random Forest": random_forest_model,
    "XGBoost": xgboost_model,
    "AdaBoost": adaboost_model,
    "Gradient Boosting": gradient_boosting_model
}

# Continue with the model training and evaluation
for model_name, model_func in models.items():
    print(f"Training {model_name}...")
    model, predictions = model_func(X_train_scaled, X_test_scaled, y_train, y_test)
    print(f"{model_name} training complete.\n")
```

```
Training Linear Regression...
Linear Regression MSE: 0.4897251729873994
Linear Regression training complete.

Training SVR...
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
<ipython-input-7-edd4b26f5247>:59: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the s
  model.fit(X_train, y_train)
SVR MSE: 0.25172846899649765
SVR training complete.

Training Decision Tree...
Decision Tree Regressor MSE: 0.02
Decision Tree training complete.

Training Random Forest...
Random Forest Regressor MSE: 0.0148815
Random Forest training complete.

Training XGBoost...
XGBoost Regressor MSE: 0.0063367193039440715
XGBoost training complete.

Training AdaBoost...
AdaBoost Regressor MSE: 0.3113021773200784
AdaBoost training complete.

Training Gradient Boosting...
Gradient Boosting Regressor MSE: 0.10948036012641058
Gradient Boosting training complete.

Training LSTM...
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_gb.py:437: DataConversionWarning: A column-vector y was passed when a 1d array
  y = column_or_1d(y, warn=True)
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-8-4efe37c45b5f> in <cell line: 28>()
     28 for model_name, model_func in models.items():
     29     print(f"Training {model_name}...")
---> 30     model, predictions = model_func(X_train_scaled, X_test_scaled, y_train, y_test)
     31     print(f"{model_name} training complete.\n")

<ipython-input-7-edd4b26f5247> in lstm_model(X_train, X_test, y_train, y_test)
     90 def lstm_model(X_train, X_test, y_train, y_test):
     91     model = Sequential()
---> 92     model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
     93     model.add(Dropout(0.2))
     94     model.add(LSTM(units=50))

IndexError: tuple index out of range
```

```
from keras.models import Sequential, Model
from keras.layers import LSTM, Dense, Dropout, GRU, SimpleRNN, Bidirectional, Input, Dot, Activation, Reshape
from keras.optimizers import Adam
from sklearn.metrics import mean_squared_error

# Ensure input shape is correctly specified for each model
def lstm_model(X_train, X_test, y_train, y_test):
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"LSTM Model MSE: {mse}")
    return model, predictions

def bi_lstm_model(X_train, X_test, y_train, y_test):
    model = Sequential()
    model.add(Bidirectional(LSTM(units=50, return_sequences=True), input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(units=50)))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"Bi-LSTM Model MSE: {mse}")
    return model, predictions

def gru_model(X_train, X_test, y_train, y_test):
    model = Sequential()
    model.add(GRU(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    model.add(GRU(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"GRU Model MSE: {mse}")
    return model, predictions

def rnn_model(X_train, X_test, y_train, y_test):
    model = Sequential()
    model.add(SimpleRNN(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    model.add(SimpleRNN(units=50))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=0)
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    print(f"RNN Model MSE: {mse}")
    return model, predictions



# Load and preprocess the data
data = pd.read_csv("/content/sample_data/output.csv")
categorical_columns = ["Initial Continent", "Initial Climate", "Final Continent", "Final Climate"]
X, y = preprocess_data(data, target_column="Impact", categorical_columns=categorical_columns)
X_train_scaled, X_test_scaled, y_train, y_test = split_and_scale(X, y)

# Train and evaluate DL models
dl_models = {
    "LSTM": lstm_model,
    "Bi-LSTM": bi_lstm_model,
    "GRU": gru_model,
    "RNN": rnn_model
}
```

```
from keras.models import Sequential, Model
from keras.layers import LSTM, Dense, Dropout, GRU, SimpleRNN, Bidirectional, Input, Dot, Activation, Reshape
from keras.optimizers import Adam
from sklearn.metrics import mean_squared_error


batch_size = 32
epochs = 100
unit = 50

# Ensure input shape is correctly specified for each model
def lstm_model(X_train, X_test, y_train, y_test, unit, batch, epochs):
    # Reshape data for LSTM [samples, time steps, features]
    X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
    X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

    model = Sequential()
    model.add(LSTM(units=unit, return_sequences=True, input_shape=(1, X_train.shape[1])))
    model.add(Dropout(0.2))
    model.add(LSTM(units=unit))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    model.fit(X_train_reshaped, y_train, epochs=epochs, batch_size=batch, validation_data=(X_test_reshaped, y_test), verbose=0)
    predictions = model.predict(X_test_reshaped)
    mse = mean_squared_error(y_test, predictions)
    print(f"LSTM Model MSE: {mse}")
    return model, predictions

def bi_lstm_model(X_train, X_test, y_train, y_test, unit, batch, epochs):
    # Reshape data for Bi-LSTM [samples, time steps, features]
    X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
    X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```