1.Imports

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error, classification_report, confusion_matrix, roc_auc_score, f1
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor

from keras.models import Sequential
from keras.layers import Dense, LSTM
```

2.Upload File

```
from google.colab import files
uploaded = files.upload()
```

Choose Files preprocess…ST_data.csv

- **preprocessed_LST_data.csv**(text/csv) - 81878404 bytes, last modified: 6/21/2025 - 100% done

Saving preprocessed_LST_data.csv to preprocessed_LST_data.csv

3.Defined Parameters and Read in Chunks

```
chunk_size = 100000
csv_path = '/content/preprocessed_LST_data.csv'

model_rf = RandomForestRegressor()
model_svm = SVR()
model_ann = MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=500)

model_lstm = None
input_dim = None
scaler_lstm = StandardScaler()

encoder = LabelEncoder()
metrics_list = []

last_chunk_df = None
```

4.Train All Models on Chunks

```
chunks = pd.read_csv(csv_path, chunksize=chunk_size)

for i, chunk in enumerate(chunks):
    print(f"🔁 Training on Chunk {i+1}")

    chunk.dropna(inplace=True)
    chunk['Region'] = encoder.fit_transform(chunk['Region'])
    chunk['Source'] = encoder.fit_transform(chunk['Source'])

    X = chunk.drop(columns=['LST'])
    y = chunk['LST']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Random Forest
    model_rf.fit(X_train_scaled, y_train)
    pred_rf = model_rf.predict(X_test_scaled)

    # SVM
    model_svm.fit(X_train_scaled, y_train)
    pred_svm = model_svm.predict(X_test_scaled)
```

```
    # ANN
    model_ann.fit(X_train_scaled, y_train)
    pred_ann = model_ann.predict(X_test_scaled)

    # LSTM
    if input_dim is None:
        input_dim = X_train.shape[1]
        model_lstm = Sequential([
            LSTM(64, input_shape=(1, input_dim)),
            Dense(1)
        ])
        model_lstm.compile(optimizer='adam', loss='mse')

    X_train_lstm = scaler_lstm.fit_transform(X_train)
    X_test_lstm = scaler_lstm.transform(X_test)

    X_train_lstm = X_train_lstm.reshape((X_train_lstm.shape[0], 1, input_dim))
    X_test_lstm = X_test_lstm.reshape((X_test_lstm.shape[0], 1, input_dim))

    model_lstm.fit(X_train_lstm, y_train, epochs=5, batch_size=64, verbose=0)
    pred_lstm = model_lstm.predict(X_test_lstm).flatten()

    # Save last chunk for test visuals
    if i == 0:
        final_X_test = X_test
        final_y_test = y_test
        preds = {
            'RF': pred_rf,
            'SVM': pred_svm,
            'ANN': pred_ann,
            'LSTM': pred_lstm
        }

    # Optional: Stop early for demo
    if i == 1:
        break
```

```
🔁 Training on Chunk 1
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
938/938 ━━━━━━━━━━━━━━━━━━━━ 1s 891us/step
🔁 Training on Chunk 2
938/938 ━━━━━━━━━━━━━━━━━━━━ 1s 780us/step
```

5.Evaluation Function

```
def print_metrics(name, y_true, y_pred):
    print(f"🔍 {name}")
    print(f"R² Score: {r2_score(y_true, y_pred):.4f}")
    print(f"MAE: {mean_absolute_error(y_true, y_pred):.4f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_true, y_pred)):.4f}")
    print(f"STD: {np.std(y_pred):.4f}")
    print("="*30)

    # Convert LST to bins for classification-style metrics
    y_class = pd.qcut(y_true, 4, labels=False)
    pred_class = pd.qcut(y_pred, 4, labels=False)

    print(f"Accuracy: {accuracy_score(y_class, pred_class):.4f}")
    print(f"Precision: {precision_score(y_class, pred_class, average='weighted'):.4f}")
    print(f"Recall: {recall_score(y_class, pred_class, average='weighted'):.4f}")
    print(f"F1 Score: {f1_score(y_class, pred_class, average='weighted'):.4f}")
    try:
        print(f"ROC-AUC: {roc_auc_score(y_class, pred_class, multi_class='ovo'):.4f}")
    except:
        print("ROC-AUC: Not applicable (use binary or prob estimates)")
    print("="*50)
```

6.Evaluate All Models

Double-click (or enter) to edit

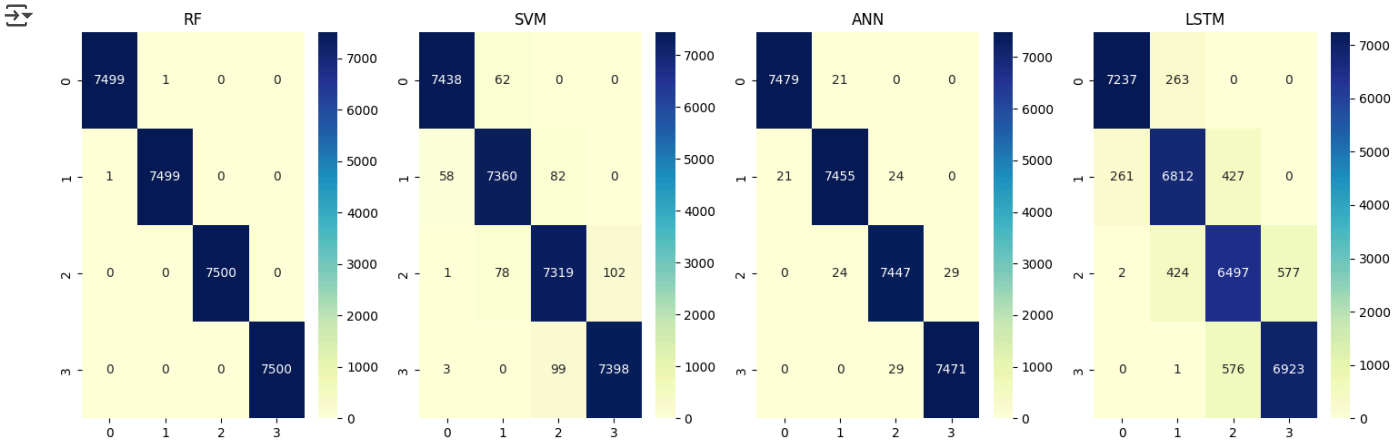
```
for name, pred in preds.items():
    print_metrics(name, final_y_test, pred)
```

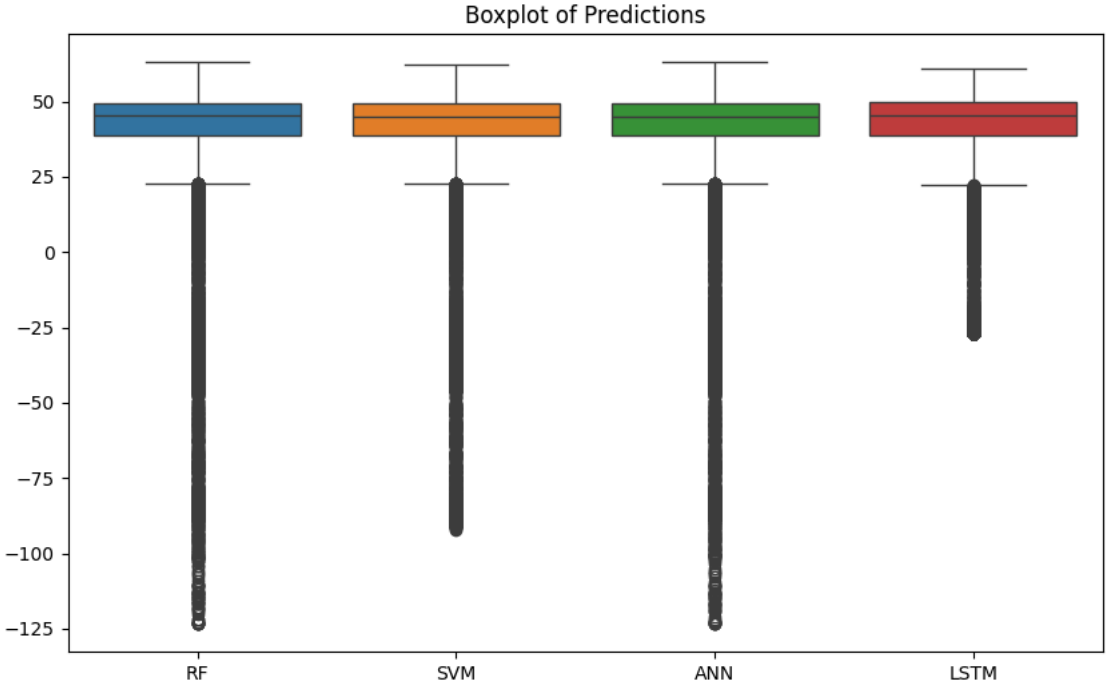
```
🔍 RF
R² Score: 1.0000
MAE: 0.0020
RMSE: 0.0133
STD: 20.5308
==============================
Accuracy: 0.9999
Precision: 0.9999
Recall: 0.9999
F1 Score: 0.9999
ROC-AUC: Not applicable (use binary or prob estimates)
==================================================
🔍 SVM
R² Score: 0.9901
MAE: 0.3083
RMSE: 2.0400
STD: 19.7853
==============================
Accuracy: 0.9838
Precision: 0.9838
Recall: 0.9838
F1 Score: 0.9838
ROC-AUC: Not applicable (use binary or prob estimates)
==================================================
🔍 ANN
R² Score: 1.0000
MAE: 0.0551
RMSE: 0.0787
STD: 20.5016
==============================
Accuracy: 0.9951
Precision: 0.9951
Recall: 0.9951
F1 Score: 0.9951
ROC-AUC: Not applicable (use binary or prob estimates)
==================================================
🔍 LSTM
R² Score: 0.8912
MAE: 1.5980
RMSE: 6.7730
STD: 16.2439
==============================
Accuracy: 0.9156
Precision: 0.9156
Recall: 0.9156
F1 Score: 0.9156
ROC-AUC: Not applicable (use binary or prob estimates)
==================================================
```

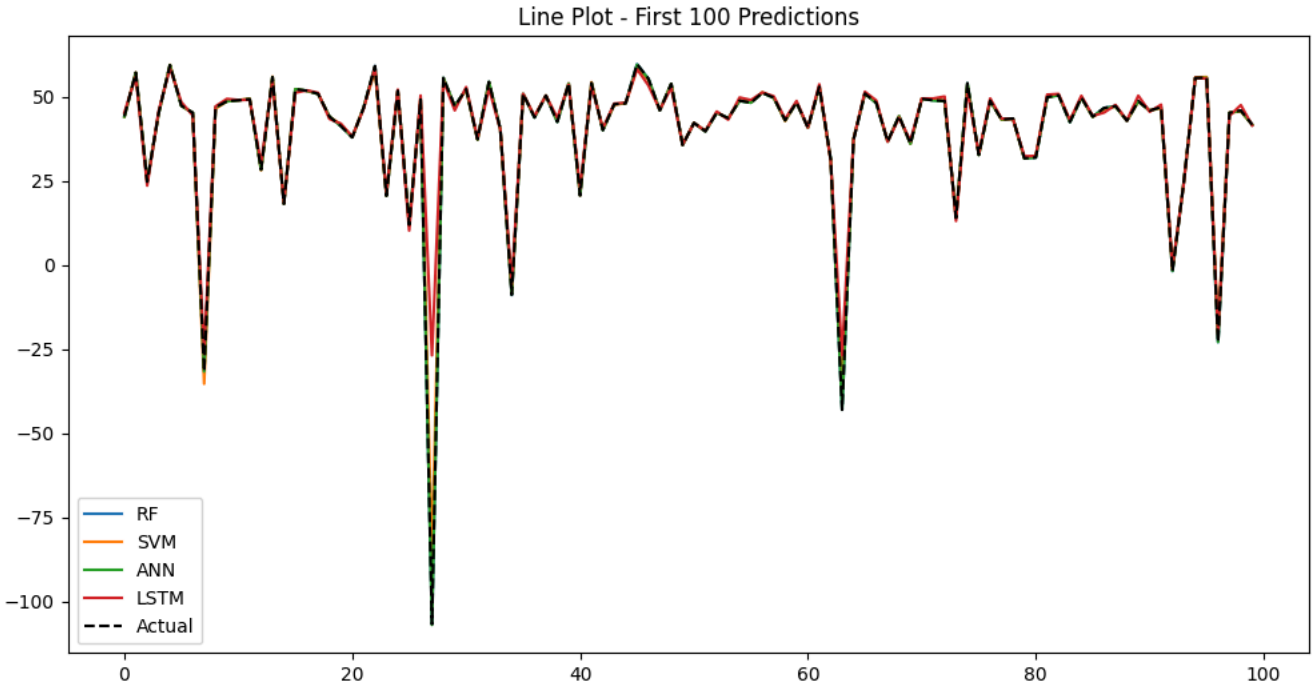
7.Visualizations

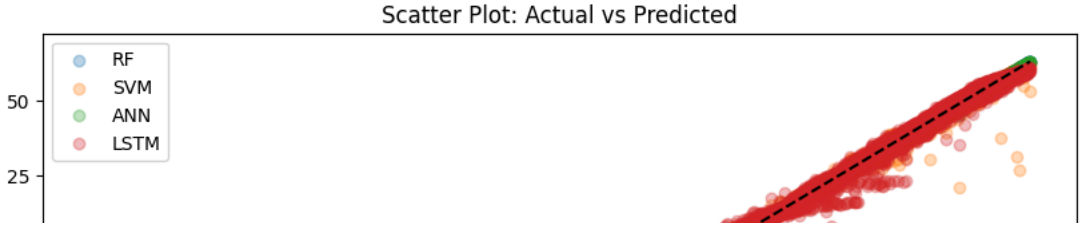
```
# Confusion matrix heatmap (on bins)
y_class = pd.qcut(final_y_test, 4, labels=False)

plt.figure(figsize=(15,5))
for i, (name, pred) in enumerate(preds.items()):
    pred_class = pd.qcut(pred, 4, labels=False)
    cm = confusion_matrix(y_class, pred_class)
    plt.subplot(1, 4, i+1)
    sns.heatmap(cm, annot=True, fmt="d", cmap="YlGnBu")
    plt.title(name)

plt.tight_layout()
plt.show()

# Boxplot
plt.figure(figsize=(10,6))
sns.boxplot(data=[preds['RF'], preds['SVM'], preds['ANN'], preds['LSTM']])
plt.xticks([0, 1, 2, 3], ['RF', 'SVM', 'ANN', 'LSTM'])
plt.title("Boxplot of Predictions")
plt.show()

# Line Plot
plt.figure(figsize=(12, 6))
for name, pred in preds.items():
    plt.plot(pred[:100], label=name)
plt.plot(final_y_test.values[:100], label="Actual", linestyle="--", color="black")
plt.title("Line Plot - First 100 Predictions")
plt.legend()
plt.show()

# Scatter Plot
```

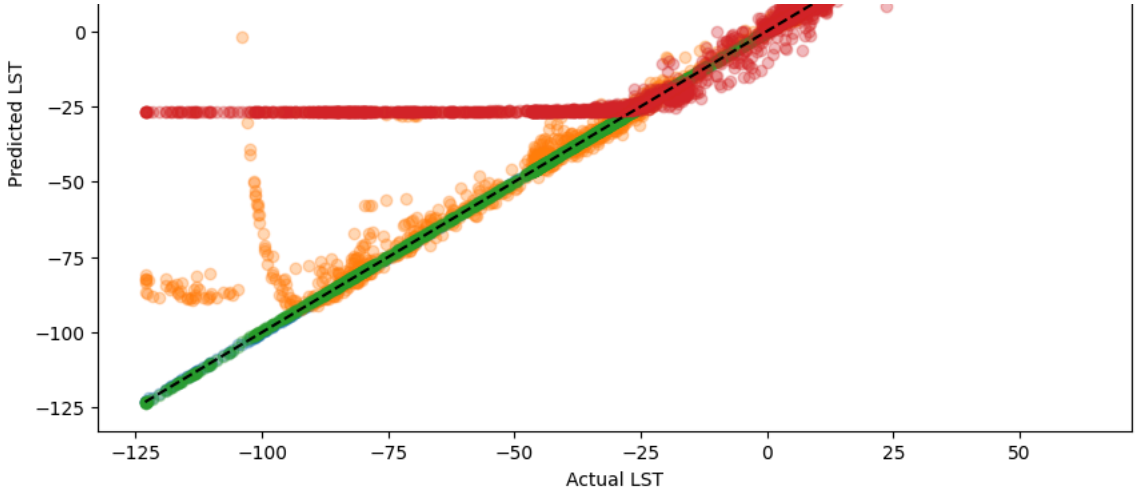
```
plt.figure(figsize=(10,6))
for name, pred in preds.items():
    plt.scatter(final_y_test.values, pred, label=name, alpha=0.3)
plt.plot([final_y_test.min(), final_y_test.max()], [final_y_test.min(), final_y_test.max()], 'k--')
plt.xlabel("Actual LST")
plt.ylabel("Predicted LST")
plt.legend()
plt.title("Scatter Plot: Actual vs Predicted")
plt.show()
```


RF
7499 1 0 0
1 7499 0 0
0 0 7500 0
0 0 0 7500
SVM
7438 62 0 0
58 7360 82 0
1 78 7319 102
3 0 99 7398
ANN
7479 21 0 0
21 7455 24 0
0 24 7447 29
0 0 29 7471
LSTM
7237 263 0 0
261 6812 427 0
2 424 6497 577
0 1 576 6923


Boxplot of Predictions
RF
SVM
ANN
LSTM


Line Plot - First 100 Predictions
RF
SVM
ANN
LSTM
Actual

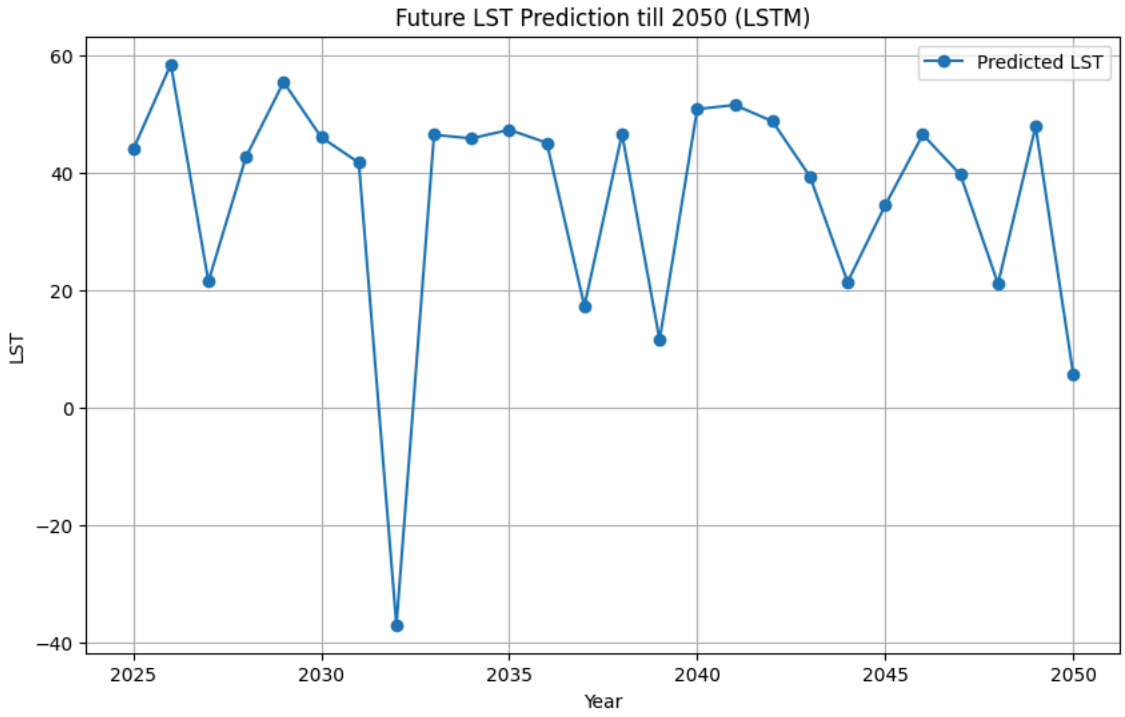

Scatter Plot: Actual vs Predicted
RF
SVM
ANN
LSTM



8.LSTM Future Prediction till 2050

```
# Use LSTM model to predict LST for years up to 2050
future_years = pd.DataFrame({'Year': list(range(2025, 2051))})
future_base = final_X_test.iloc[:len(future_years)].copy()
future_base['Year'] = future_years['Year'].values

# Add region/source manually if needed
future_base['Region'] = 0
future_base['Source'] = 0

future_scaled = scaler_lstm.transform(future_base)
future_scaled = future_scaled.reshape((future_scaled.shape[0], 1, input_dim))

future_pred = model_lstm.predict(future_scaled).flatten()

# Plot
plt.figure(figsize=(10,6))
plt.plot(future_years['Year'], future_pred, marker='o', label='Predicted LST')
plt.title("Future LST Prediction till 2050 (LSTM)")
plt.xlabel("Year")
plt.ylabel("LST")
plt.grid()
plt.legend()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
```

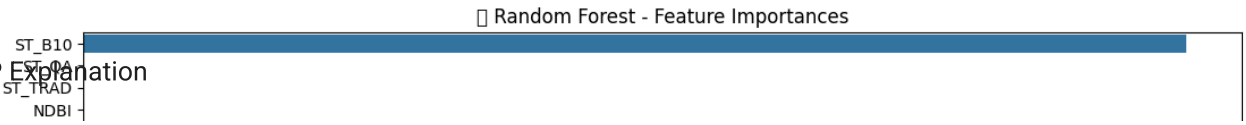


✅ Part 1: Feature Importance & SHAP Analysis (for Random Forest)

🔹 1.1 Feature Importance Plot for Random Forest

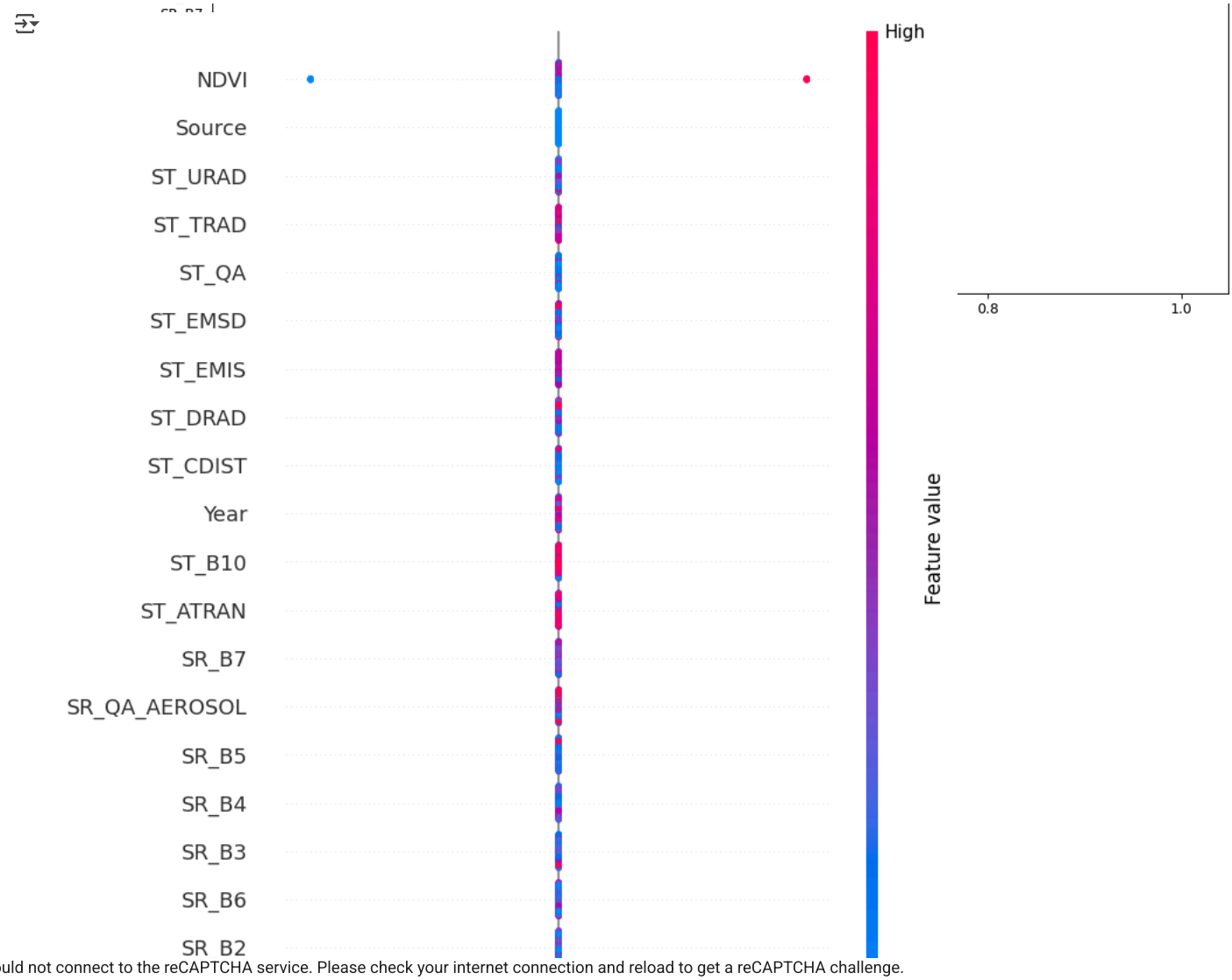
```
# Feature Importance from Random Forest
importances = model_rf.feature_importances_
features = final_X_test.columns
sorted_indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12,6))
sns.barplot(x=importances[sorted_indices], y=features[sorted_indices])
plt.title("🎯 Random Forest - Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-11-2605942263.py:11: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 127919 (\N{DIRECT HIT}) missing from font
  fig.canvas.print_figure(bytes_io, **kw)
```



1.2 SHAP Explanation

```
sampled_X = final_X_test.sample(100)
explainer = shap.Explainer(model_rf, sampled_X)
shap_values = explainer(sampled_X)
shap.summary_plot(shap_values, sampled_X)
```



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.