Tsunami Classification

Kendra Pinckney, Marina Fasano, Christina Corrado, Julie Kasday

Table of Contents

Purpose

- Tsunamis are a large threat to coastal communities, with no seasonality or preventative measures. Early predictions can assist in issuing timely warning for the communities impacted.

  How can we most accurately predict if a tsunami will occur? What model can train with the highest evaluation metrics?

Logistic Regression

- Great for classification Simple, fast, interpretable
- Assumes a linear relationship between features
- preferred for explaining what features contribute most
- Minimizes log-loss, maximizes likelihood that observed labels come from predicted probabilities

- For our data: want the highest prediction accuracy.
- nonlinear interactions (high magnitude and shallow depth).
- Logistic regression may underfit our data.

Random Forest

- Uses many decision trees and combines their predictions using ensemble learning
- predicts a point using the majority outcome across trees
- does not assume linearity, handles variation and high dimensional data well (many features or noisy data)
- Locally optimizes node purity, averaging predictions over trees

- Our data is nonlinear and interaction heavy (Magnitude, focal depth, and geographic location are modeled with complexity)
- Random forest finds a strong decision boundary and have high accuracy.

XGBoost

- Gradient boosted decision tree method that sequentially builds trees and corrects errors based on previous ones.
- Takes weaker models and builds them into a stronger model using gradient descent, minimizing the loss function (training loss, regularization) to improve prediction by reducing residual error
- Excels with complex, high-dimensional, non linear relationships.

- Our data: it may perform well at predicting a tsunami when patterns are hidden (rare combination of magnitude and geographic region)
- Our dataset is smaller, it may overfit (boosting can learn noise anyway, causing the model to be overly specific)

Data Design

Binary classification model that predicts whether an earthquake will generate a tsunami. Specifically testing non-linear vs linear models.

Inputs

Environmental and seismic features:

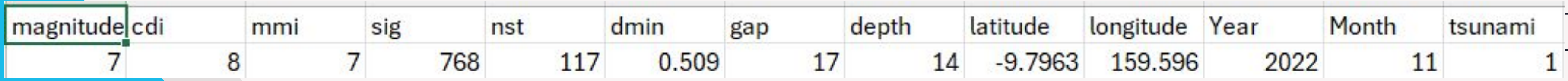
- Earthquake magnitude
- Oceanic location
- Depth
- etc.

Outputs

Binary classification:

- 1 = Tsunami occurs
- 0 = No tsunami occurs

Example: An earthquake with a magnitude of 7, shallow depth, and near pacific coast -> predicts high tsunami risk

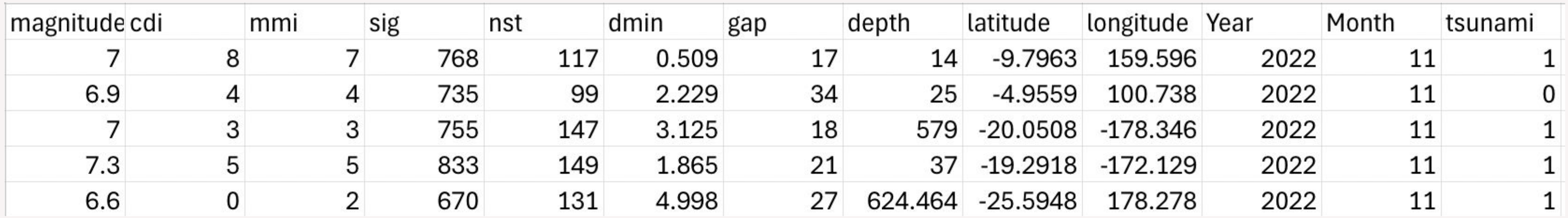


magnitude	cdi	mmi	sig	nst	dmin	gap	depth	latitude	longitude	Year	Month	tsunami
7	8	7	768	117	0.509	17	14	-9.7963	159.596	2022	11	1

Experimental Plan:

Global Earthquake-Tsunami Risk Assessment Dataset from Kaggle

- Records: 782 significant earthquakes
- Time Period: January 2001- December 2022



magnitude	cdi	mmi	sig	nst	dmin	gap	depth	latitude	longitude	Year	Month	tsunami
7	8	7	768	117	0.509	17	14	-9.7963	159.596	2022	11	1
6.9	4	4	735	99	2.229	34	25	-4.9559	100.738	2022	11	0
7	3	3	755	147	3.125	18	579	-20.0508	-178.346	2022	11	1
7.3	5	5	833	149	1.865	21	37	-19.2918	-172.129	2022	11	1
6.6	0	2	670	131	4.998	27	624.464	-25.5948	178.278	2022	11	1

Languages: Python
Libraries: Pandas, Numpy, Scikit-Learn (sklearn), XGBoost, Matplotlib.pyplot, Seaborn

Evaluation Metrics

Accuracy: What proportion of predictions were correct?

-doesn’t take into account biased data

Log Loss: Measures how good the model’s probability prediction is by calculating the negative log of predicted probabilities for the true class.

Precision: Of all the events that were predicted “X” (ex: tsunami=1), what percent were correct.

-High precision → few false alarms

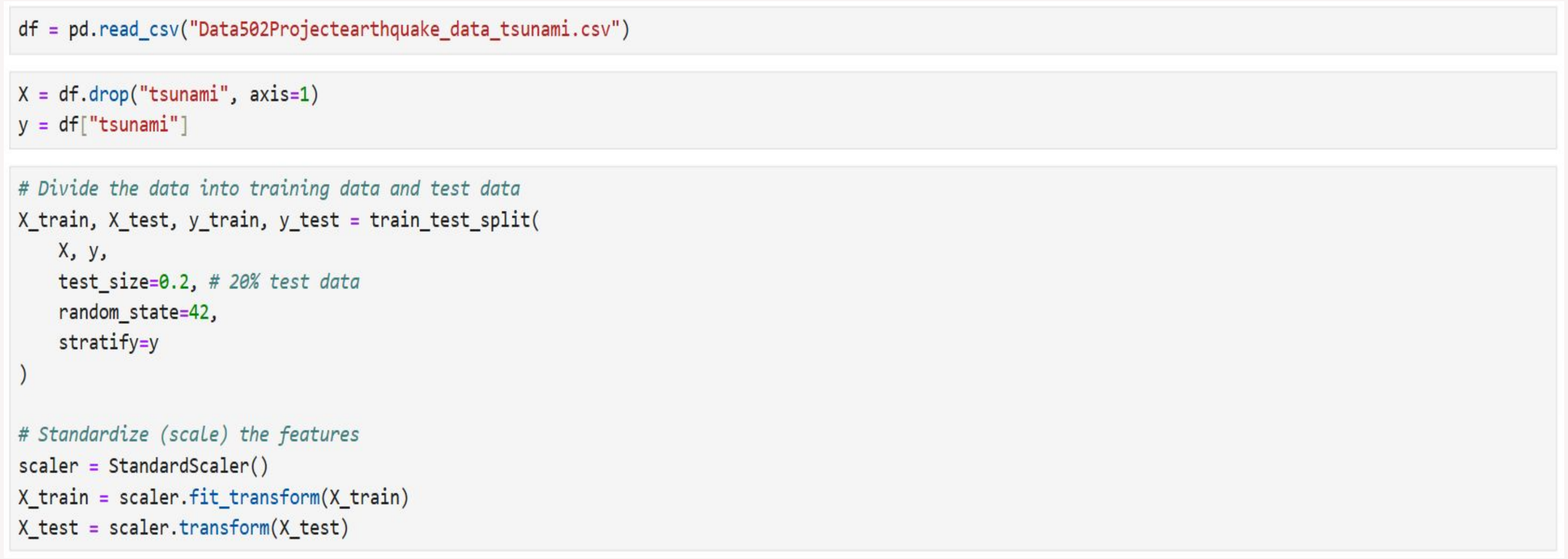
-Low precision → lots of false positives

*Recall: Of all the “X” events (ex: tsunami=1), how many were correctly predicted?

-High recall → model catches most tsunamis

-Low recall → model misses tsunamis

Demo: Loading and Training the Data

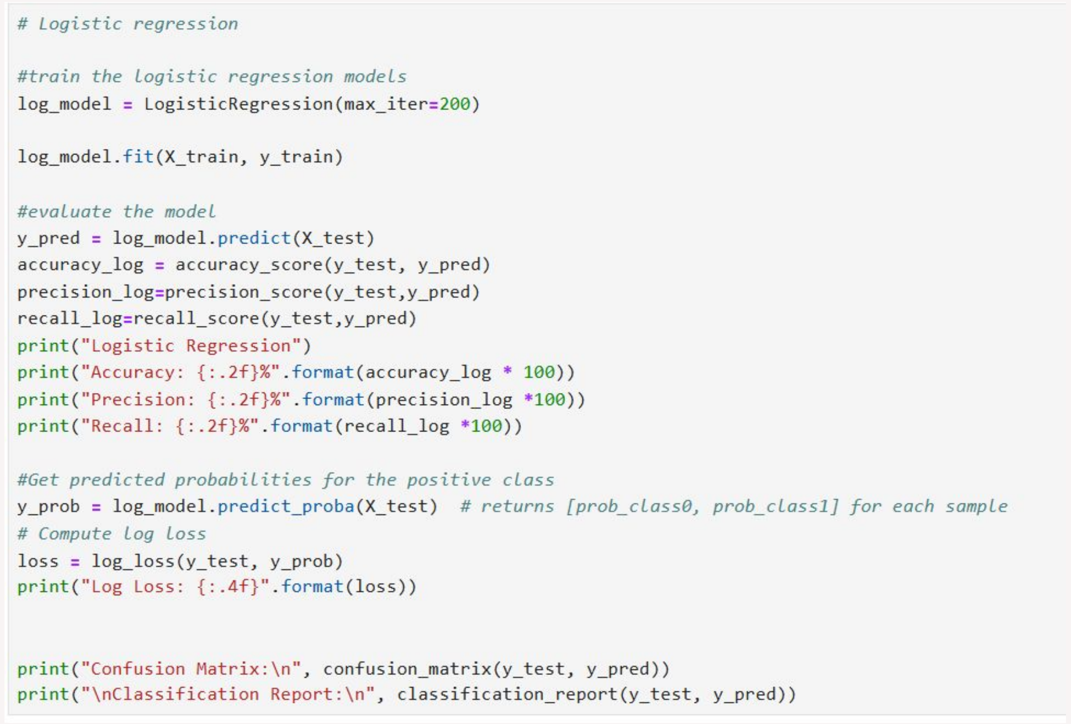


```
df = pd.read_csv("Data502Projectearthquake_data_tsunami.csv")
```

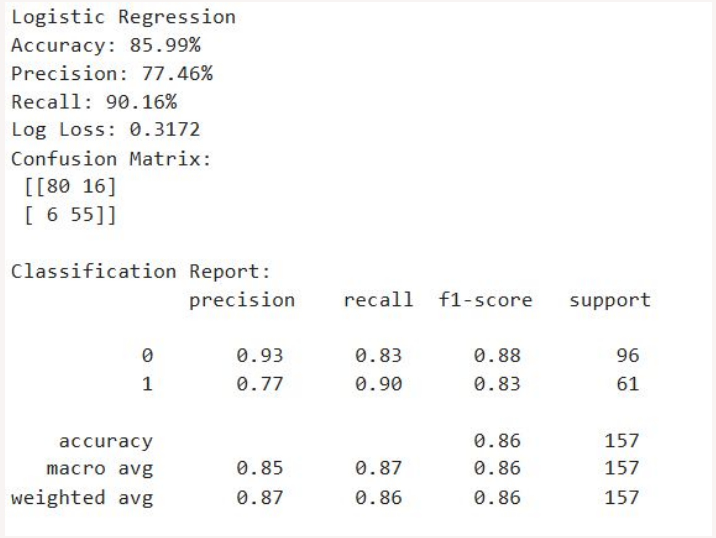
```
X = df.drop("tsunami", axis=1)
y = df["tsunami"]
```

```
# Divide the data into training data and test data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2, # 20% test data
    random_state=42,
    stratify=y
)

# Standardize (scale) the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Logistic Regression

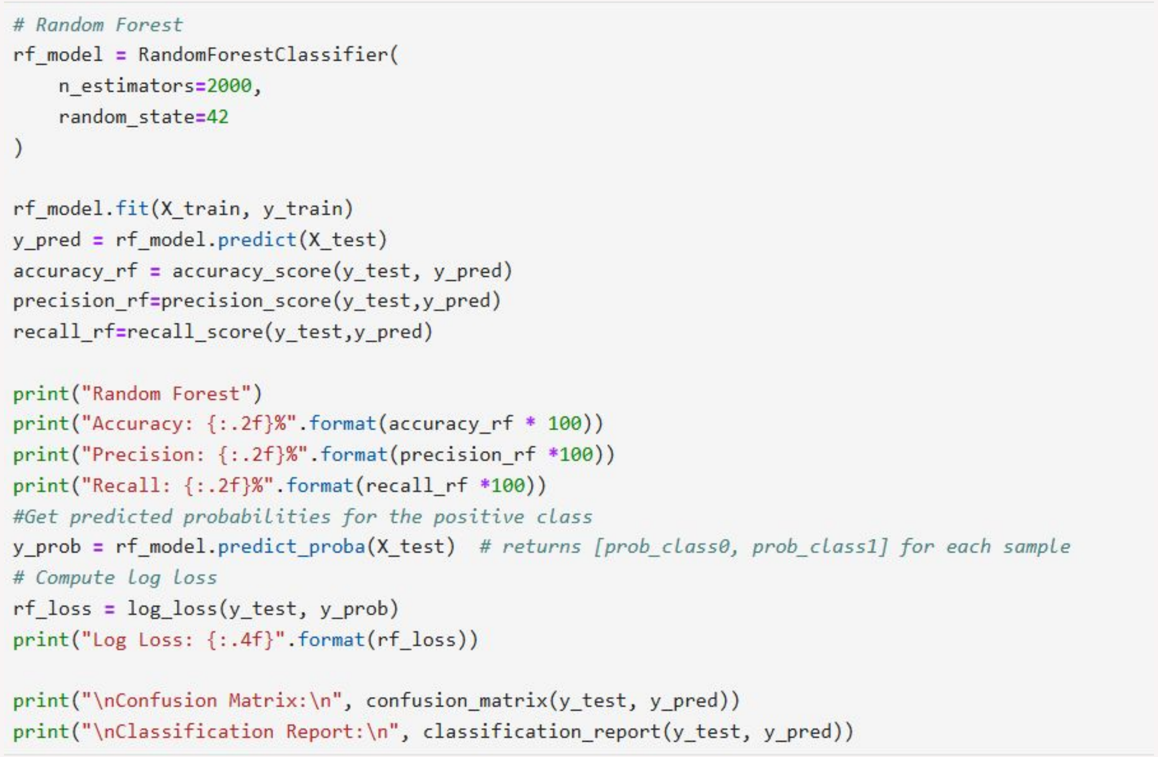


```python
# Logistic regression

#train the logistic regression models
log_model = LogisticRegression(max_iter=200)

log_model.fit(X_train, y_train)

#evaluate the model
y_pred = log_model.predict(X_test)
accuracy_log = accuracy_score(y_test, y_pred)
precision_log=precision_score(y_test,y_pred)
recall_log=recall_score(y_test,y_pred)
print("Logistic Regression")
print("Accuracy: {:.2f}%".format(accuracy_log * 100))
print("Precision: {:.2f}%".format(precision_log *100))
print("Recall: {:.2f}%".format(recall_log *100))

#Get predicted probabilities for the positive class
y_prob = log_model.predict_proba(X_test)  # returns [prob_class0, prob_class1] for each sample
# Compute log loss
loss = log_loss(y_test, y_prob)
print("Log Loss: {:.4f}".format(loss))


print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

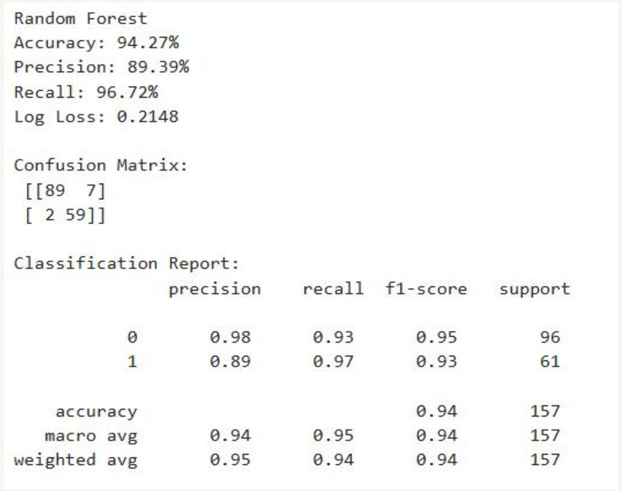


```
Logistic Regression
Accuracy: 85.99%
Precision: 77.46%
Recall: 90.16%
Log Loss: 0.3172
Confusion Matrix:
 [[80 16]
 [ 6 55]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.83      0.88        96
           1       0.77      0.90      0.83        61

    accuracy                           0.86       157
   macro avg       0.85      0.87      0.86       157
weighted avg       0.87      0.86      0.86       157
```

Random Forest

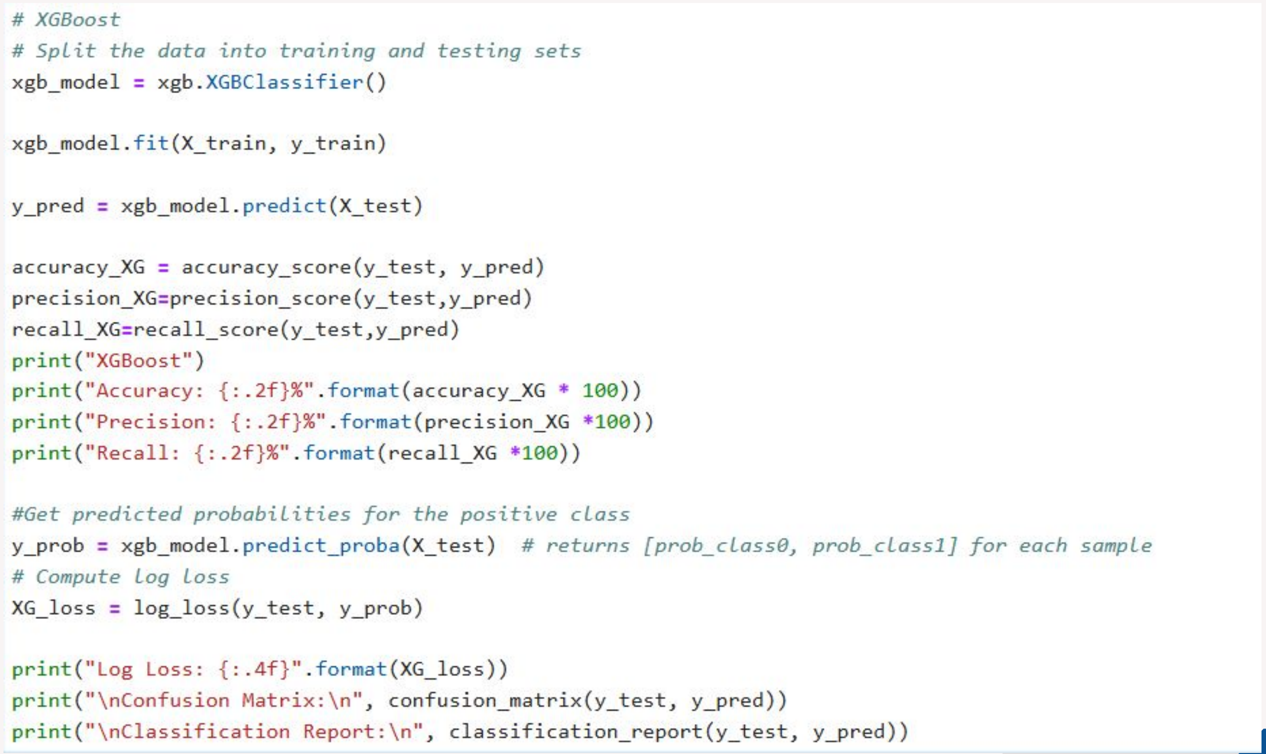


```python
# Random Forest
rf_model = RandomForestClassifier(
    n_estimators=2000,
    random_state=42
)

rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf=precision_score(y_test,y_pred)
recall_rf=recall_score(y_test,y_pred)

print("Random Forest")
print("Accuracy: {:.2f}%".format(accuracy_rf * 100))
print("Precision: {:.2f}%".format(precision_rf *100))
print("Recall: {:.2f}%".format(recall_rf *100))
#Get predicted probabilities for the positive class
y_prob = rf_model.predict_proba(X_test)  # returns [prob_class0, prob_class1] for each sample
# Compute log loss
rf_loss = log_loss(y_test, y_prob)
print("Log Loss: {:.4f}".format(rf_loss))

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

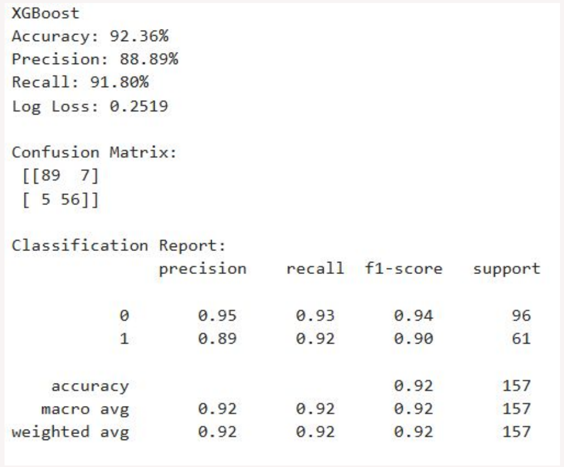


```
Random Forest
Accuracy: 94.27%
Precision: 89.39%
Recall: 96.72%
Log Loss: 0.2148

Confusion Matrix:
 [[89  7]
 [ 2 59]]

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.93      0.95        96
           1       0.89      0.97      0.93        61

    accuracy                           0.94       157
   macro avg       0.94      0.95      0.94       157
weighted avg       0.95      0.94      0.94       157
```

XGBoost



```
# XGBoost
# Split the data into training and testing sets
xgb_model = xgb.XGBClassifier()

xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)

accuracy_XG = accuracy_score(y_test, y_pred)
precision_XG=precision_score(y_test,y_pred)
recall_XG=recall_score(y_test,y_pred)
print("XGBoost")
print("Accuracy: {:.2f}%".format(accuracy_XG * 100))
print("Precision: {:.2f}%".format(precision_XG *100))
print("Recall: {:.2f}%".format(recall_XG *100))

#Get predicted probabilities for the positive class
y_prob = xgb_model.predict_proba(X_test)  # returns [prob_class0, prob_class1] for each sample
# Compute log loss
XG_loss = log_loss(y_test, y_prob)

print("Log Loss: {:.4f}".format(XG_loss))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

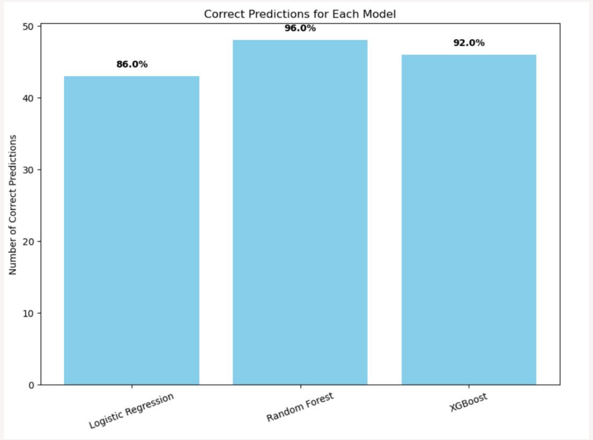


```
XGBoost
Accuracy: 92.36%
Precision: 88.89%
Recall: 91.80%
Log Loss: 0.2519

Confusion Matrix:
 [[89  7]
 [ 5 56]]

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.93      0.94        96
           1       0.89      0.92      0.90        61

    accuracy                           0.92       157
   macro avg       0.92      0.92      0.92       157
weighted avg       0.92      0.92      0.92       157
```

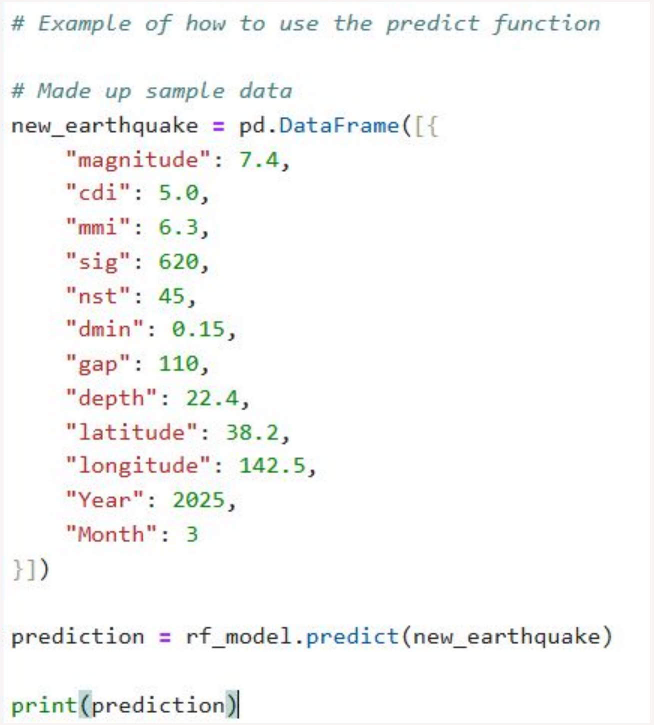
Accuracy for 50 Random Samples of our Data

```python
sample_size = 50
sample_idx = np.random.choice(len(X_test), sample_size, replace=False)

X_sample = X_test[sample_idx]
y_sample = y_test.iloc[sample_idx]

models = {
    "Logistic Regression": log_model,
    "Random Forest": rf_model,
    "XGBoost": xgb_model
}

accuracies = {}
correct_counts = {}

for name, model in models.items():
    preds = model.predict(X_sample)
    acc = accuracy_score(y_sample, preds)

    accuracies[name] = acc
    correct_counts[name] = sum(preds == y_sample)
```

```python
plt.figure(figsize=(10,7))
bars = plt.bar(correct_counts.keys(), correct_counts.values(), color='skyblue')

plt.ylabel("Number of Correct Predictions")
plt.title("Correct Predictions for Each Model")
plt.xticks(rotation=20)

for bar, model_name in zip(bars, accuracies.keys()):
    height = bar.get_height()
    acc_percent = f"{accuracies[model_name]*100:.1f}%"

    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height + 1,                 # slightly above the bar
        acc_percent,
        ha='center',
        va='bottom',
        fontsize=10,
        fontweight='bold'
    )

plt.show()
```

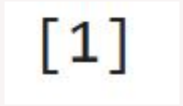


Predict Function Code



```python
# Example of how to use the predict function

# Made up sample data
new_earthquake = pd.DataFrame([{
    "magnitude": 7.4,
    "cdi": 5.0,
    "mmi": 6.3,
    "sig": 620,
    "nst": 45,
    "dmin": 0.15,
    "gap": 110,
    "depth": 22.4,
    "latitude": 38.2,
    "longitude": 142.5,
    "Year": 2025,
    "Month": 3
}])

prediction = rf_model.predict(new_earthquake)

print(prediction)
```

Sample Output:

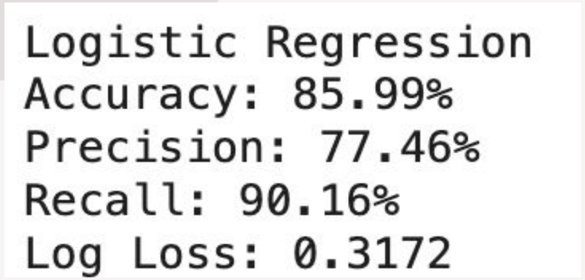


```
[1]
```

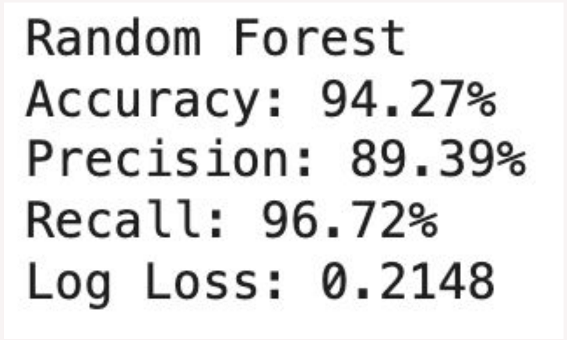
Outputs include:

[1] which indicates a tsunami will likely occur

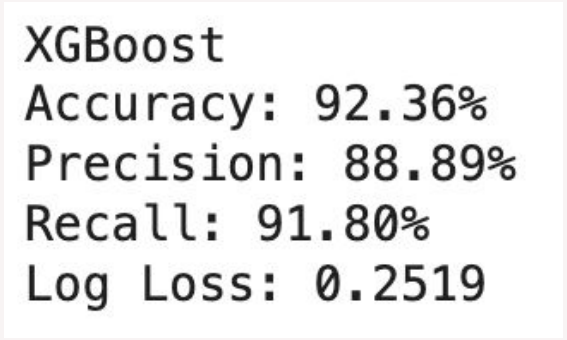
[0] indicates a tsunami will not likely occur



```
Logistic Regression
Accuracy: 85.99%
Precision: 77.46%
Recall: 90.16%
Log Loss: 0.3172
```



```
Random Forest
Accuracy: 94.27%
Precision: 89.39%
Recall: 96.72%
Log Loss: 0.2148
```



```
XGBoost
Accuracy: 92.36%
Precision: 88.89%
Recall: 91.80%
Log Loss: 0.2519
```

Results

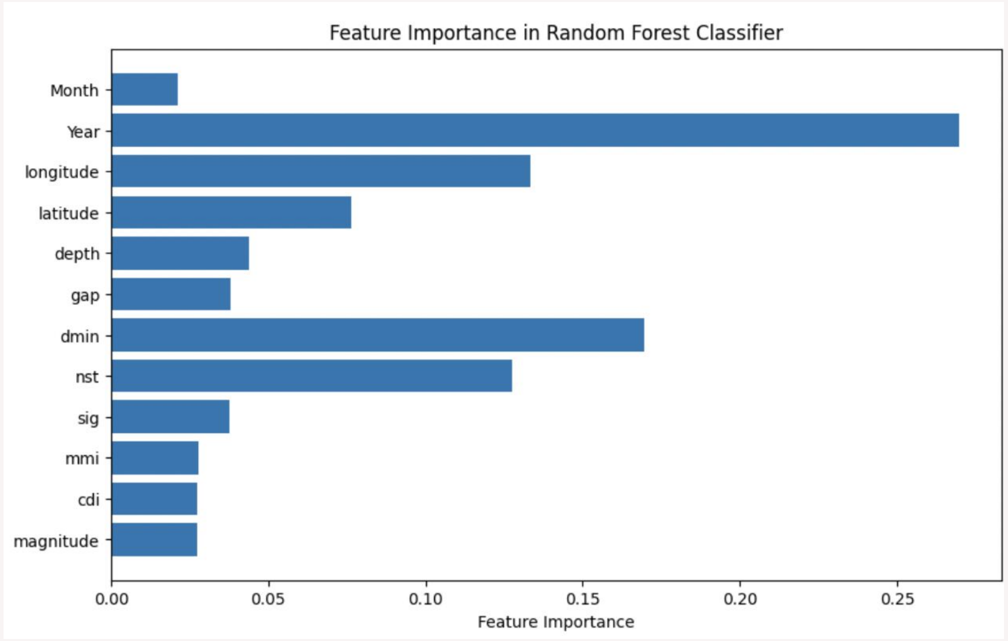
Random Forest and XGBoost performed the best overall, significantly outperforming Logistic Regression.
-Higher recall and precision
-Higher accuracy
-Lower log loss

Random Forest achieved the highest recall (96.72%), meaning it detected almost every tsunami event.

Random Forest achieved the lowest log loss (0.2148), showing the most reliable probability predictions.

Logistic Regression performed reasonably, but struggled compared to tree-based models on accuracy and probability calibration.

Improvements to our model



- Finding the most optimal number of trees- Grid Search
- Limiting tree depth max_depth
- Removing least important features
- Adjusting Bootstrap sample size

Limitations

1. Features
   a. Only seismic attributes, but tsunamis depend on oceanic and geographical factors as well
2. Simplicity of Binary Classification
   a. Ignoring severity of the specific tsunami occurrence
3. Generalization of Data
   a. Trained on past events, future earthquakes and shifts may have different characteristics not listed

Future Work

1. Expansion of Features
   a. Oceanographic and geospatial incorporation (population density, tide levels for example)
2. Severity Prediction
   a. Instead of just predicting the occurrence, predict the tsunami intensity
3. Real-time Data
   a. Connect with seismic monitors for early warning applications

Citations

Logistic Regression using Python - GeeksforGeeks

https://www.geeksforgeeks.org/dsa/random-forest-classifier-using-scikit-learn/

https://medium.com/@fraidoonomarzai99/xgboost-classification-in-depth-979f11ef4bf9

https://www.geeksforgeeks.org/dsa/random-forest-classifier-using-scikit-learn/

https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/

https://www.weather.gov/safety/tsunami

https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/

https://www.geeksforgeeks.org/machine-learning/xgboost/