

Model Building and Evaluation

Importing Packages

In this part, we will use different models to predict the recovery time for a given parameters using Random Forest, Ridge, and Lasso. The below code is all the libraries necessary to run all the models.

```
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
```

First, let's observe the data!

```
import pandas as pd

df = pd.read_csv("data/global_disaster_response_2018_2024.csv")
df.head()

      date      country disaster_type  severity_index  casualties \
0  2021-01-31      Brazil   Earthquake          5.99        111
1  2018-12-23      Brazil  Extreme Heat          6.53        100
2  2020-08-10      India    Hurricane          1.55         22
3  2022-09-15  Indonesia  Extreme Heat          4.55         94
4  2022-09-28  United States   Wildfire          3.80         64

      economic_loss_usd  response_time_hours  aid_amount_usd \
0            7934365.71             15.62       271603.79
1            8307648.99              5.03       265873.81
2            765136.99             32.54       49356.49
3           1308251.31              7.83       237512.88
4            2655864.36             21.90       188910.69

      response_efficiency_score  recovery_days  latitude  longitude
0                      83.21            67 -30.613 -122.557
1                      96.18            55  10.859 -159.194
2                      60.40            22   0.643 -160.978
3                      86.41            47 -33.547   30.350
4                      72.81            42 -19.170 -117.137
```

Let's first identify how many different disaster types and countries are in this data!

```
print("How many disaster types?", len(df['disaster_type'].value_counts()))
```

```
print("How many countries?", len(df['country'].value_counts()))
```

```
How many disaster types? 10
How many countries? 20
```

From looking at the data, the data is not organized, so let's first organize the data by time. Then, let's split the data for train data and test data. Since we the date starts from 2018 January to 2024 December, we can set train data from 2018 January to 2022 December and set test data from 2023 January to 2024 December.

```
df['date'] = pd.to_datetime(df['date'])
df.sort_values(by='date', inplace=True)

train = df[df['date'] <= "2022-12-31"].copy()
test = df[df['date'] >= "2023-01-01"].copy()
```

Right now, the date data has year, month, and day. So, it would be good to split it into year and month.

```
for d in [train, test]:
    d["year"] = d["date"].dt.year
    d["month"] = d["date"].dt.month
```

Predicting the Recovery Time

After splitting the date, we can drop the "date" column.

```
train = train.drop(columns = ['date'])
test = test.drop(columns = ['date'])
```

Since we are interested in the recovery day, we can set y as the recovery days column and X to be everything beside recovery days.

```
y_train = train['recovery_days']
y_test = test['recovery_days']
X_train = train.drop(columns=['recovery_days'])
X_test = test.drop(columns=['recovery_days'])
```

After that is done, we can create lists of num_features and cat_features, where we are going to use these lists of data and preprocess the data using Standard Scalar and One Hot Encoder. This allow us to automatically encode and scale the data.

```
num_features = [
    "severity_index",
    "casualties",
    "economic_loss_usd",
    "response_time_hours",
    "aid_amount_usd",
    "response_efficiency_score",
```

```

    "latitude",
    "longitude",
    "year",
    "month",
]
cat_features = ["country", "disaster_type"]
preprocessor = ColumnTransformer(transformers=[('num', StandardScaler(), num_features), ('ca

```

Model Fitting

Here, we use Random Forest, Ridge, and Lasso and fit the models, then we predict the outcomes for the given X_test.

```

random_forest_model = Pipeline([('prep', preprocessor), ('model', RandomForestRegressor(ran
random_forest_model.fit(X_train, y_train)
ridge_model = Pipeline([('prep', preprocessor), ('model', Ridge()))])
ridge_model.fit(X_train, y_train)
lasso_model = Pipeline([('prep', preprocessor), ('model', Lasso()))])
lasso_model.fit(X_train, y_train)

Pipeline(steps=[('prep',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                 ['severity_index',
                                                  'casualties',
                                                  'economic_loss_usd',
                                                  'response_time_hours',
                                                  'aid_amount_usd',
                                                  'response_efficiency_score',
                                                  'latitude', 'longitude',
                                                  'year', 'month']),
                                         ('cat',
                                          OneHotEncoder(handle_unknown='ignore'),
                                          ['country',
                                           'disaster_type'])])),
                 ('model', Lasso())])

```

After the models have been fit, we can then predict the outcome for the given X_test. After all the predictions are out, we can do MSE (Mean Squared Error) to see the best model and check whether our prediction was close to the actual outcome.

```

# (Optional) MSE for reference only; main metrics are RMSE/MAE/R2 saved to outputs/ml/metrics
mse_ridge = mean_squared_error(y_test, preds_ridge)
mse_ridge
24.786798609280467
import numpy as np

```

```

import pandas as pd
from pathlib import Path
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

preds_random_forest = random_forest_model.predict(X_test)
preds_ridge = ridge_model.predict(X_test)
preds_lasso = lasso_model.predict(X_test)

baseline_pred = np.full(len(y_test), y_train.mean())

def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def metrics_row(model_name, y_true, y_pred):
    return {
        "model": model_name,
        "RMSE": rmse(y_true, y_pred),
        "MAE": mean_absolute_error(y_true, y_pred),
        "R2": r2_score(y_true, y_pred),
    }

results = pd.DataFrame([
    metrics_row("baseline_mean", y_test, baseline_pred),
    metrics_row("random_forest", y_test, preds_random_forest),
    metrics_row("ridge", y_test, preds_ridge),
    metrics_row("lasso", y_test, preds_lasso),
]).sort_values("RMSE")

Path("outputs/ml").mkdir(parents=True, exist_ok=True)
results.to_csv("outputs/ml/metrics.csv", index=False)

results
      model      RMSE       MAE       R2
2     ridge  4.978634  3.970246  0.939425
1  random_forest  5.093356  4.059898  0.936602
3     lasso  5.099404  4.062804  0.936451
0  baseline_mean 20.228813 16.355676 -0.000025

from pathlib import Path
Path("outputs/ml/metrics.csv").exists()

True

Path("outputs/ml").mkdir(parents=True, exist_ok=True)
results.to_csv("outputs/ml/metrics.csv", index=False)
results

```

	model	RMSE	MAE	R2
2	ridge	4.978634	3.970246	0.939425
1	random_forest	5.093356	4.059898	0.936602
3	lasso	5.099404	4.062804	0.936451
0	baseline_mean	20.228813	16.355676	-0.000025