Global Disaster Response Analysis (2018–2024)

Monday 15th December, 2025

Curvenote

1 Exploratory Data Analysis (EDA)

This notebook is for analyzing the relationships between different variables to figure out what research question to ask. This is done using various graphs and visualizations. There are labels at the end of each graph describing anything interesting about the graph.

We will also use this initial analysis to determine what variables we want to use in the later parts and determine our research question.

All of the graph outputs are accessible through the folder graphs ->eda

We will be working with the Global Disaster Response Analysis dataset from Kaggle. Questions we intend exploring are related to how long it takes for a country to recover based on the aid amount and disaster type.

1.0.1 Load Data

```
# import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from scipy import stats

# load dataframe
import pandas as pd

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

	date	country	disaster_type	severity_index	casualties	economic_loss	response_time_hours	aid_amount	recovery_time_days	latitude	longitude
0	2021-01-31	Brazil	Earthquake	5.99	111	7934365.52	15.52	27160337.21	67	-30.613	-122.557
1	2018-12-23	Brazil	Extreme Heat	6.53	100	8307648.39	48.39	26587381.18	55	10.859	-159.194
2	2020-08-10	India	Hurricane	1.55	22	7651362.54	32.54	49356.40	9.40	22	0.643	-160.978
3	2022-09-15	Indonesia	Extreme Heat	4.55	94	1308257.31	83.31	2375128.68	41	47	-33.547	30.350
4	2022-09-28	United States	Wildfire	3.80	64	2655862.36	21.36	1889107.89	7.89	42	-19.170	-117.137

View unique values of some colums

```
df['disaster_type'].value_counts()
```

```
disaster_type
Landslide            5130
Earthquake           5068
Flood                5039
Hurricane            5002
Extreme Heat         5001
Storm Surge          4988
Volcanic Eruption    4983
Wildfire             4954
Tornado              4939
Drought              4896
Name: count, dtype: int64
```

```
df['country'].value_counts()
```

```
country
Brazil           2591
Australia        2563
Turkey           2554
Bangladesh       2553
Spain            2543
China            2539
Chile            2529
Nigeria          2528
Germany          2526
India            2509
Greece           2503
Italy            2503
South Africa     2497
Japan            2472
Indonesia        2467
Canada           2438
Philippines      2437
Mexico           2433
United States    2413
France           2402
Name: count, dtype: int64
```

1.1 Analyze relationships between variables

1.1.1 Aid and response:

Description of Methods: The following plots are joint plots that include a histogram of the distribution of each variable, along with a hexplot of the

relationship between the two variables. The darker the the color in the hexplot, the more data points are in that region of the plot.

You may notice that we take the square root of some variables. This is to fit a previously curved diagram using a linear model (see the Tuskey-Mosteller Bulging Rule).

To visualize the relationship between aid amount and disaster effects/response times, let's graph the following:

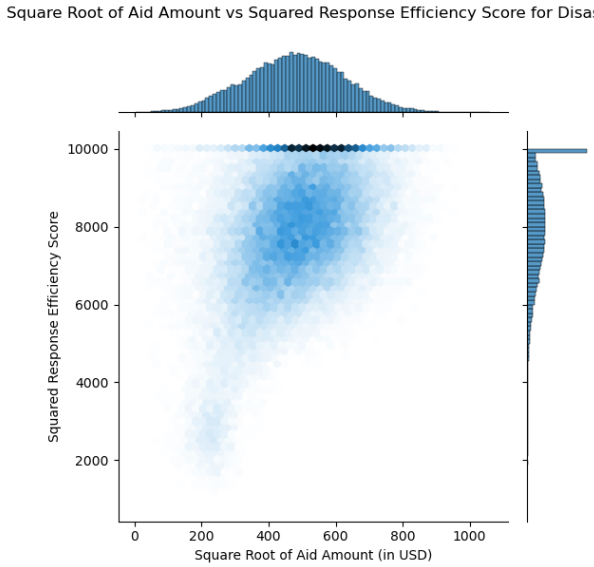
1. Joint plot: aid amount and response efficiency
2. Joint plot: aid amount and response time
3. Joint plot: aid amount and economic loss
4. Joint plot: aid amount and casualties

```
# joint plot of square root of aid amount and response efficiency

g = sns.jointplot(data=df, x=np.sqrt(df['aid_amount_usd']), y=df['response_efficiency_score'
g.fig.suptitle('Square Root of Aid Amount vs Squared Response Efficiency Score for Disasters
plt.xlabel('Square Root of Aid Amount (in USD)')
plt.ylabel('Squared Response Efficiency Score')
plt.tight_layout()
# Saving the plot as a JPEG file
plt.savefig("graphs/eda/1_aid_efficiency.jpg", bbox_inches='tight')
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FileNotFoundError                         Traceback (most recent call last)
Cell In[27], line 9
      7 plt.tight_layout()
      8 # Saving the plot as a JPEG file
 - - - -> 9 plt.savefig("graphs/eda/1_aid_efficiency.jpg", bbox_inches='tight')

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/pyplot.py:1251, in sa
   1248 fig = gcf()
   1249 # savefig default implementation has no return, so mypy is unhappy
   1250 # presumably this is here because subclasses can return?
 -> 1251 res = fig.savefig(*args, **kwargs)  # type: ignore[func -returns -value]
   1252 fig.canvas.draw_idle()  # Need this if 'transparent=True', to reset colors.
   1253 return res

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/figure.py:3490, in Fi
   3488     for ax in self.axes:
   3489         _recursively_make_axes_transparent(stack, ax)
 -> 3490 self.canvas.print_figure(fname, **kwargs)

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/backend_bases.py:2184
```

```
   2180 try:
   2181     # _get_renderer may change the figure dpi (as vector formats
   2182     # force the figure dpi to 72), so we need to set it again here.
   2183     with cbook._setattr_cm(self.figure, dpi=dpi):
 -> 2184         result = print_method(
   2185             filename,
   2186             facecolor=facecolor,
   2187             edgecolor=edgecolor,
   2188             orientation=orientation,
   2189             bbox_inches_restore=_bbox_inches_restore,
   2190             **kwargs)
   2191 finally:
   2192     if bbox_inches and restore_bbox:

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/backend_bases.py:2040
   2036     optional_kws = {  # Passed by print_figure for other renderers.
   2037         "dpi", "facecolor", "edgecolor", "orientation",
   2038         "bbox_inches_restore"}
   2039     skip = optional_kws - {*inspect.signature(meth).parameters}
 -> 2040     print_method = functools.wraps(meth)(lambda *args, **kwargs: meth(
   2041         *args, **{k: v for k, v in kwargs.items() if k not in skip}))
   2042 else:  # Let third -parties do as they see fit.
   2043     print_method = meth

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/backends/backend_agg.
    493 def print_jpg(self, filename_or_obj, *, metadata=None, pil_kwargs=None):
    494     # savefig() has already applied savefig.facecolor; we now set it to
    495     # white to make imsave() blend semi -transparent figures against an
    496     # assumed white background.
    497     with mpl.rc_context({"savefig.facecolor": "white"}):
 - -> 498         self._print_pil(filename_or_obj, "jpeg", pil_kwargs, metadata)

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/backends/backend_agg.
    425 """
    426 Draw the canvas, then save it using `.image.imsave` (to which
    427 *pil_kwargs* and *metadata* are forwarded).
    428 """
    429 FigureCanvasAgg.draw(self)
 - -> 430 mpl.image.imsave(
    431     filename_or_obj, self.buffer_rgba(), format=fmt, origin="upper",
    432     dpi=self.figure.dpi, metadata=metadata, pil_kwargs=pil_kwargs)

File /srv/conda/envs/notebook/lib/python3.12/site -packages/matplotlib/image.py:1657, in ims
   1655 pil_kwargs.setdefault("format", format)
   1656 pil_kwargs.setdefault("dpi", (dpi, dpi))
 -> 1657 image.save(fname, **pil_kwargs)
```

```
File /srv/conda/envs/notebook/lib/python3.12/site -packages/PIL/Image.py:2576, in Image.save
   2574         fp = builtins.open(filename, "r+b")
   2575     else:
 -> 2576         fp = builtins.open(filename, "w+b")
   2577 else:
   2578     fp = cast(IO[bytes], fp)

FileNotFoundError: [Errno 2] No such file or directory: 'graphs/eda/1_aid_efficiency.jpg'
```



The original distribution of aid amount is skewed to the right, so after taking its square root, it becomes roughly normal. The distribution of the response efficiency score is more dense around 70-100, with a very high concentration at 100. Below 100, the distribution also follows a bell-shaped curve. If we were to use the response efficiency score in a predicitive model, we would need to account for the high concentration of 100's. Disregarding the zeros, we can see a somewhat positive linear relationship between the two variables.

```
# joint plot of aid amount and response efficiency

g = sns.jointplot(data=df, x=np.sqrt(df['aid_amount_usd']), y=np.sqrt(df['response_time_hour
g.fig.suptitle('Square Root of Aid Amount vs Square Root of Response Time for Disasters', y=
plt.xlabel('Square Root of Aid Amount (in USD)')
plt.ylabel('Square Root of Response Time (in hours)')
plt.tight_layout()
# Saving the plot as a JPEG file
plt.savefig("graphs/eda/2_aid_response_time.jpg", bbox_inches='tight')
```

This is very similar to the graph above, reflected across the x-axis. The distribution of the response efficiency score is more dense around 0-30, with a very high concentration at 0. Above 0, the distribution also follows a bell-shaped curve. If we were to use the response efficiency score in a predicitive model, we would need to account for the high concentration of 0's. Disregarding the zeros, we can see a somewhat negative relationship between the two variables.

```
# Create the jointplot
g = sns.jointplot(data=df, x=np.sqrt(df['aid_amount_usd']), y=np.sqrt(df['economic_loss_usd'

# Add regression line
x = np.sqrt(df['aid_amount_usd'])
y = np.sqrt(df['economic_loss_usd'])
sns.regplot(x=x, y=y, scatter=False, ax=g.ax_joint, color='red', line_kws={'linewidth': 2})

# Calculate correlation coefficient
r, p_value = stats.pearsonr(x, y)

# Add correlation text to the plot
g.ax_joint.text(0.05, 0.95, f'r = {r:.3f}\np < {p_value:.3f}',
                transform=g.ax_joint.transAxes,
                verticalalignment='top',
                bbox=dict(boxstyle='round', facecolor='white', alpha=0.8),
                fontsize=10)

g.fig.suptitle('Square Root of Aid Amount vs Square Root of Economic Loss for Disasters', y=
g.set_axis_labels('Square Root of Aid Amount (USD)', 'Square Root of Economic Loss (USD)')
plt.tight_layout()

# Saving the plot as a JPEG file
plt.savefig("graphs/eda/3_aid_econ_loss.jpg", bbox_inches='tight')

# Create the jointplot
g = sns.jointplot(data=df, x=np.sqrt(df['aid_amount_usd']), y=np.sqrt(df['casualties']), kin

# Add regression line
x = np.sqrt(df['aid_amount_usd'])
y = np.sqrt(df['casualties'])
sns.regplot(x=x, y=y, scatter=False, ax=g.ax_joint, color='red', line_kws={'linewidth': 2})

# Calculate correlation coefficient
r, p_value = stats.pearsonr(x, y)

# Add correlation text to the plot
g.ax_joint.text(0.05, 0.95, f'r = {r:.3f}\np < {p_value:.3f}',
                transform=g.ax_joint.transAxes,
```

```
                verticalalignment='top',
                bbox=dict(boxstyle='round', facecolor='white', alpha=0.8),
                fontsize=10)

g.fig.suptitle('Square Root of Aid Amount vs Square Root of Casualties for Disasters', y=1.0
g.set_axis_labels('Square Root of Aid Amount (USD)', 'Square Root of Casualties')  # Fixed y
plt.tight_layout()

# Saving the plot as a JPEG file
plt.savefig("graphs/eda/4_aid_casualties.jpg", bbox_inches='tight')
```

The graphs above are very similar, both with a positive linear relationship and a correlation coefficient of 0.403. This would make aid amount an important variable in predicting economic losses and casualties. The relationship between the two variables could be positive because more damages would result in a higher aid amount. So aid amount is likely a response variable to the casualties and economic losses.

1.1.2 Economic Loss Per Country

```
# make a dataframe of the total aid amount received by countries from 2018 -2024, grouped by
aid_amount = df[['country', 'aid_amount_usd']].groupby(['country']).sum().sort_values('aid_a
aid_amount

# plot the aid received by countries in descending order
plt.bar(aid_amount.index, aid_amount['aid_amount_usd'] / 1e6)
plt.xlabel('Country')
plt.ylabel('Aid Amount (Millions USD)')
plt.title('Aid received by countries in descending order')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
```

The bars in the graph above do not look very different from each other. When graphing the economic loss per country, we can also use the percentage difference from the mean since all the values don't look very different when graphed from 0 to the maximum value.

```
mean_aid = aid_amount['aid_amount_usd'].mean() # calculate the mean aid received by countrie
pct_diff = ((aid_amount['aid_amount_usd'] - mean_aid) / mean_aid) * 100 # calculate the diff

plt.bar(aid_amount.index, pct_diff)
plt.axhline(y=0, color='red', linestyle=' - -', linewidth=1)
plt.xlabel('Country')
plt.ylabel('Percentage Difference from Mean (%)')
plt.title('Total Aid Amount: Deviation from Average')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
```

```
# Saving the plot as a JPEG file
plt.savefig("graphs/eda/5_aid_country.jpg", bbox_inches='tight')
```

The percent difference from the mean ranges from around -4% to 4%, with Brazil being the country receiving the most aid and the United States being the country receiving the least amount of aid.

```
# make a dataframe of the total economic loss from countries from 2018 -2024, grouped by cou

country_loss = df[['country', 'economic_loss_usd']].groupby(['country']).sum().sort_values('
country_loss

mean_loss = country_loss['economic_loss_usd'].mean() # calculate the mean economic loss over
pct_diff = ((country_loss['economic_loss_usd'] - mean_loss) / mean_loss) * 100 # calculate t

plt.bar(country_loss.index, pct_diff)
plt.axhline(y=0, color='red', linestyle=' - -', linewidth=1)
plt.xlabel('Country')
plt.ylabel('Percentage Difference from Mean (%)')
plt.title('Total Economic Loss by Country: Deviation from Average')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Saving the plot as a JPEG file
plt.savefig("graphs/eda/6_econ_loss_country.jpg", bbox_inches='tight')
```

In this graph, the total economic losses per country ranges from -3% from the mean to 4% from the mean. The country with the highest economic loss is Brazil, and the country with the lowest economic loss is France. While the economic loss may have similar trends to the amount of aid, this shows that it is not exactly the same because the country rankings in the economic loss graph and in the aid graph are not the same.

1.1.3 Economic Loss per Disaster Type

```
# Make a dataframe of total economic loss, grouped by disaster
disaster_loss = df[['disaster_type', 'economic_loss_usd']].groupby(['disaster_type']).sum().
disaster_loss

mean_loss = disaster_loss['economic_loss_usd'].mean() # calculate the mean economic loss ove
pct_diff = ((disaster_loss['economic_loss_usd'] - mean_loss) / mean_loss) * 100 # calculate

plt.bar(disaster_loss.index, pct_diff)
plt.axhline(y=0, color='red', linestyle=' - -', linewidth=1)
plt.xlabel('Country')
plt.ylabel('Percentage Difference from Mean (%)')
```

```
plt.title('Total Economic Loss by Disaster: Deviation from Average')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Saving the plot as a JPEG file
plt.savefig("graphs/eda/7_econ_loss_disaster.jpg", bbox_inches='tight')
```

This graph shows the how different the total economic loss by disaster is from the mean. Drought is much lower compared to the others at -4% from the mean, and volcanic eruptions are around 2% from the mean.

```
# Make a dataframe of average severity, grouped by disaster
disaster_severity = df[['disaster_type', 'severity_index']].groupby(['disaster_type']).mean(
disaster_severity

mean_severity = disaster_severity['severity_index'].mean() # calculate the mean economic los
pct_diff = ((disaster_severity['severity_index'] - mean_severity) / mean_severity) * 100 # c

plt.bar(disaster_severity.index, pct_diff)
plt.axhline(y=0, color='red', linestyle=' - -', linewidth=1)
plt.xlabel('Country')
plt.ylabel('Percentage Difference from Mean (%)')
plt.title('Total Severity per Disaster: Deviation from Average')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Saving the plot as a JPEG file
plt.savefig("graphs/eda/8_severity_disaster.jpg", bbox_inches='tight')
```

The mean severity per disaster shows less variation than the total economic loss, with tornados only being less than 2% above the mean and droughts being less than 2% below the mean. From the data, we can show that droughts have less impact and severity compared to the other disasters. For disasters like earthquakes, even though each event has an average of 1% below the mean, in total, they cause significant economic loss, both being higher than the average. Tornados, on the other hand, have a high severity per event, but the overall economic loss is lower than volcanic eruptions, landslides, and earthquakes.

1.2 Main Takeaways from EDA

- Aid amount is likely affected by the number of casualties and economic loss because their plots show a positive linear trend.
- Efficiency score has a disproportionate number of values at 100 and response time has a disproportionate number of values at 0.
- Tornadoes, volcanic eruptions, and wildfires have the highest total severity per disaster from 2018-2024.

- Volcanic eruptions and landslides have the highest total economic loss over all disasters from 2018-2024.

Based on the data, a good research question to ask is, "What would the amount of aid received be, given the country and amount of economic loss?" This is because the graphs show that the amount of aid is correlated with the amount of economic loss, and the aid amount varies per country. Response time and efficiency score seem like they may not be heavily correlated with the aid amount, so they might introduce some noise into the model.

2 Model Building and Evaluation

2.1 Importing Packages

In this part, we will use different models to predict the recovery time for a given paramters 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	economic_loss	response_time_hours	aid_amount	infrastructure_damage	weather_temperature	humidity	latitude	longitude
0	2021-01-31	Brazil	Earthquake	5.99	111	7934365.52	15.52	2716038.72	3.72	21	67	-30.613	-122.557
1	2018-12-23	Brazil	Extreme Heat	6.53	100	8307648.09	48.09	2658736.18	36.18	55		10.859	-159.194
2	2020-08-10	India	Hurricane	1.55	22	765136.95	32.54	49356.69	19.40	22		0.643	-160.978
3	2022-09-15	Indonesia	Extreme Heat	4.55	94	1308257.31	57.31	237512.68	28.68	41	47	-33.547	30.350
4	2022-09-28	United States	Wildfire	3.80	64	2655862.36	21.36	188910.72	7.28	31	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 orgainzied, so let's first orgainze 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
```

2.2 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
```

2.3 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(rand
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())])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[ ]Pipeline
?Documentation for PipelineiFitted
[ ]prep: ColumnTransformer
?Documentation for prep: ColumnTransformer
[ ]num

```
['severity_index', 'casualties', 'economic_loss_usd', 'response_time_hours', 'aid_amount_usd
```

[ ]StandardScaler
?Documentation for StandardScaler
[ ]cat

```
['country', 'disaster_type']
```

[ ]OneHotEncoder
?Documentation for OneHotEncoder
[ ]Lasso
?Documentation for 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/metric
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