Predicting Layoffs

AUTHOR
Laura Rosok

PUBLISHED
October 11, 2024

Introduction

This model aims to predict the number of layoffs that may occur within a company by analyzing various factors, including location, industry type, date of layoff, funding stage, and total funds raised.

The model was developed using a comprehensive dataset comprising 2,257 tech companies that experienced layoffs from COVID 2019 to 2024. This dataset, sourced from Kaggle, provides valuable insights into layoff trends and factors influencing workforce reductions.

You can access the dataset here.

Methods

```
# Imports

# Data manipulation
import pandas as pd
import numpy as np
import os

# Visualization
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display

# Machine learning
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import Lasso, LinearRegression, ElasticNet, Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
from sklearn.metrics import root_mean_squared_error
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.ensemble import VotingRegressor
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

# Kaggle utilities
import kagglehub
```

Data

```
# load data
path = kagglehub.dataset_download("swaptr/layoffs-2022")
csv_file = next(file for file in os.listdir(path) if file.endswith('.csv'))
data = pd.read_csv(os.path.join(path, csv_file), parse_dates=['date'])
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/swaptr/layoffs-2022?dataset_version_numbe
6...
```

```
100%|██████████| 68.0k/68.0k [00:00<00:00, 971kB/s]

Extracting files...
```

Data Dictionary

Target Variable

- **`total_laid_off`** (`float64`):
  - The total number of employees laid off by the company.

Feature Variables

1. **`location`** (`object`):
   - The geographical location of the company's headquarters.
2. **`industry`** (`object`):
   - The specific industry in which the company operates.
3. **`year`** (`int32`):
   - The year during which the layoff occurred.
4. **`month`** (`int32`):
   - The month during which the layoff occurred.
5. **`stage`** (`object`):
   - The current stage of funding for the company.
6. **`country`** (`object`):
   - The country where the company is based.
7. **`funds_raised`** (`float64`):
   - The amount of funds raised by the company, measured in millions of dollars.

Data Cleaning

```
# drop unnecessary or repetitive columns
data = data.drop(['percentage_laid_off'], axis=1)

# Extract year and month from date
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month

# Drop duplicated rows
data.drop_duplicates(inplace = True)

# Check for outliers using IQR method and replace them with the median
numeric_columns = ['total_laid_off', 'funds_raised', 'year', 'month']

for col in numeric_columns:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    median = data[col].median()
    data[col] = np.where((data[col] < lower_bound) | (data[col] > upper_bound), median, data[col])
```

```
# drop null values if >=20% of column is missing

for column in data.columns:
    missing_percentage = (data[column].isnull().sum()) / len(data) * 100
    if missing_percentage >= 20:
        print(f"{column} should drop null values (missing: {missing_percentage:.2f}%)")


data.dropna(subset=['total_laid_off'], inplace=True)

# impute null values if <20% of column is missing

numeric_features = data.select_dtypes(['int', 'float']).columns.tolist()
numeric_imputer = SimpleImputer(strategy='median')
data[numeric_features] = numeric_imputer.fit_transform(data[numeric_features])

categorical_features = data.select_dtypes(['category', 'object']).columns.tolist()
categorical_imputer = SimpleImputer(strategy='most_frequent')
data[categorical_features] = categorical_imputer.fit_transform(data[categorical_features])
```

```
total_laid_off should drop null values (missing: 34.81%)
```

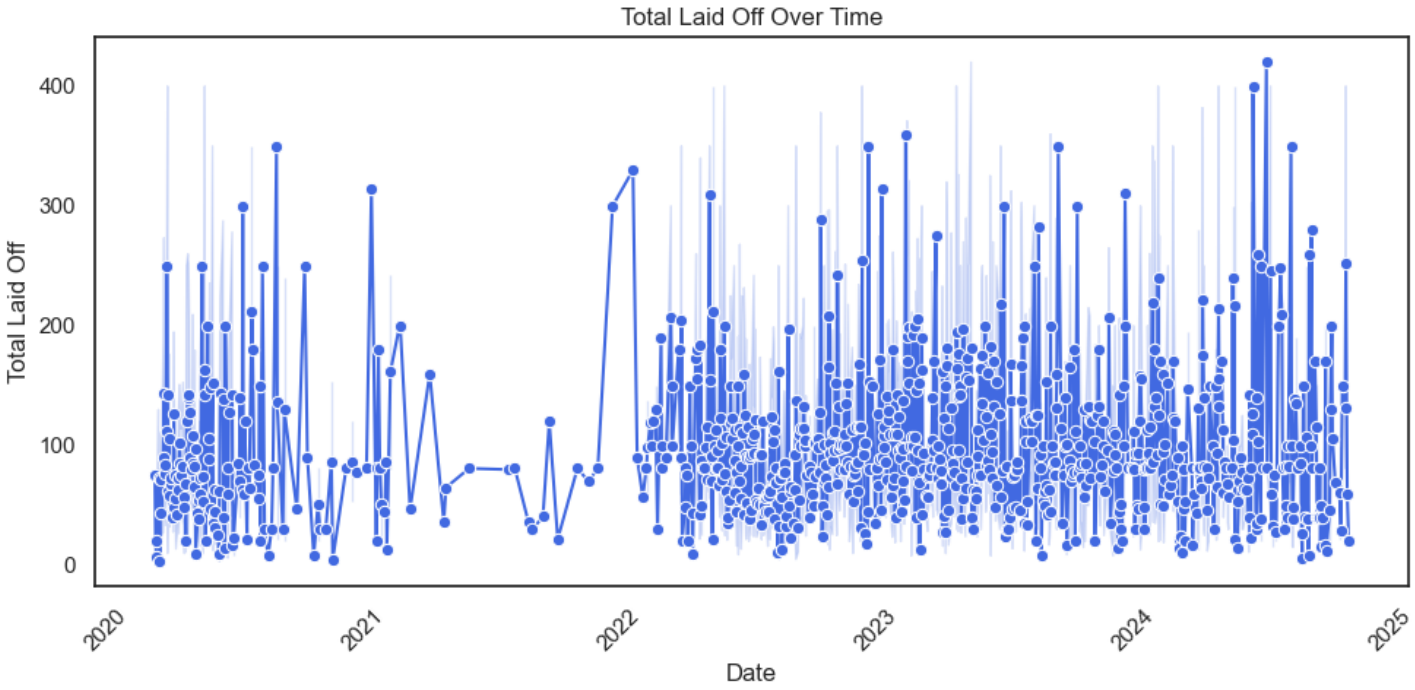
Table 1: Summary Statistics for Numeric Features

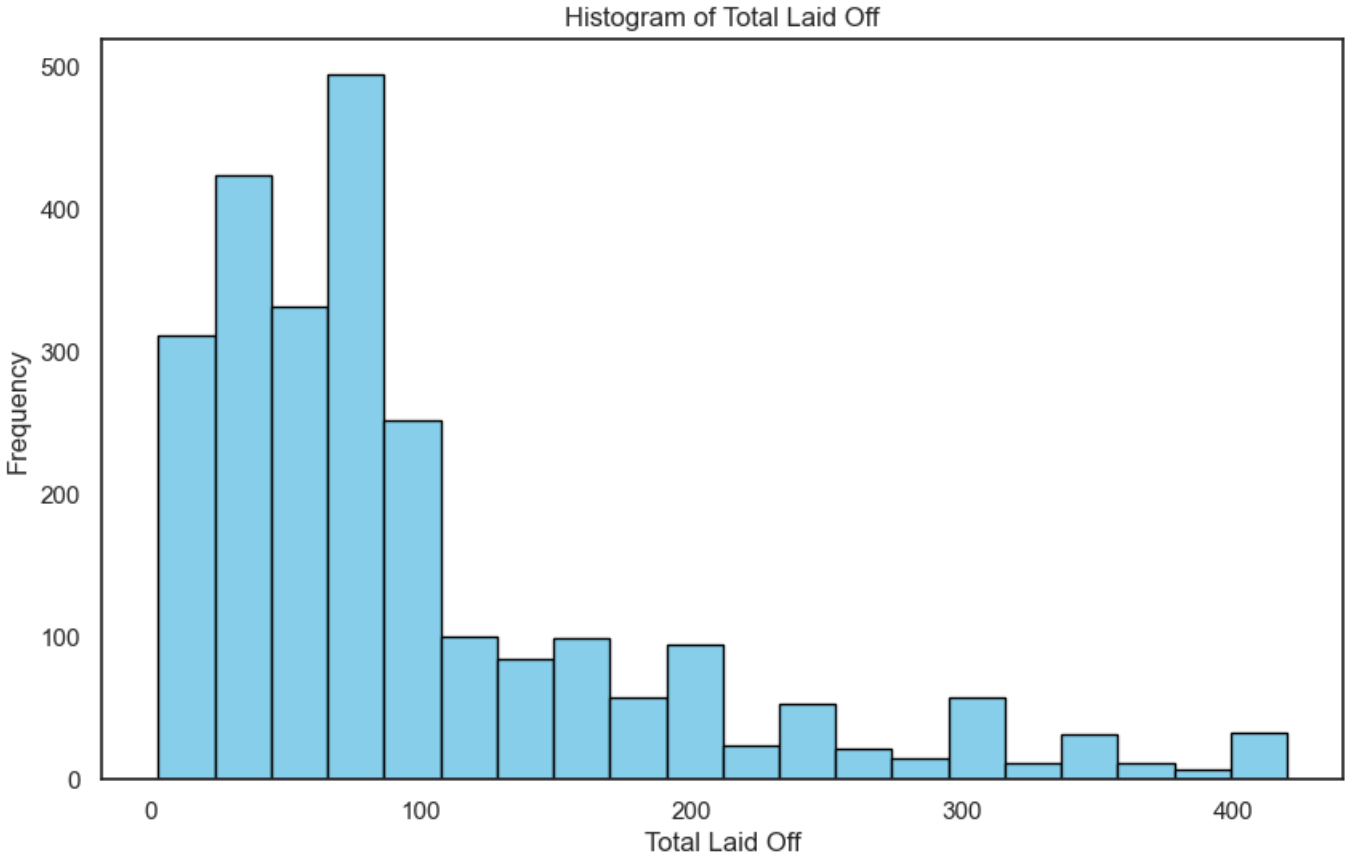
	Count	Mean	Standard Deviation	Min	25th Percentile	Median	75th Percentile	Max	missing_percentage
total_laid_off	2511	98.88	85.95	3.00	40.00	81.00	120.00	420.00	0.00%
funds_raised	2511	224.11	226.11	0.00	71.00	167.00	278.00	1100.00	0.00%
year	2511	2022.78	0.68	2021.00	2022.00	2023.00	2023.00	2024.00	0.00%
month	2511	5.82	3.23	1.00	3.00	6.00	8.00	12.00	0.00%

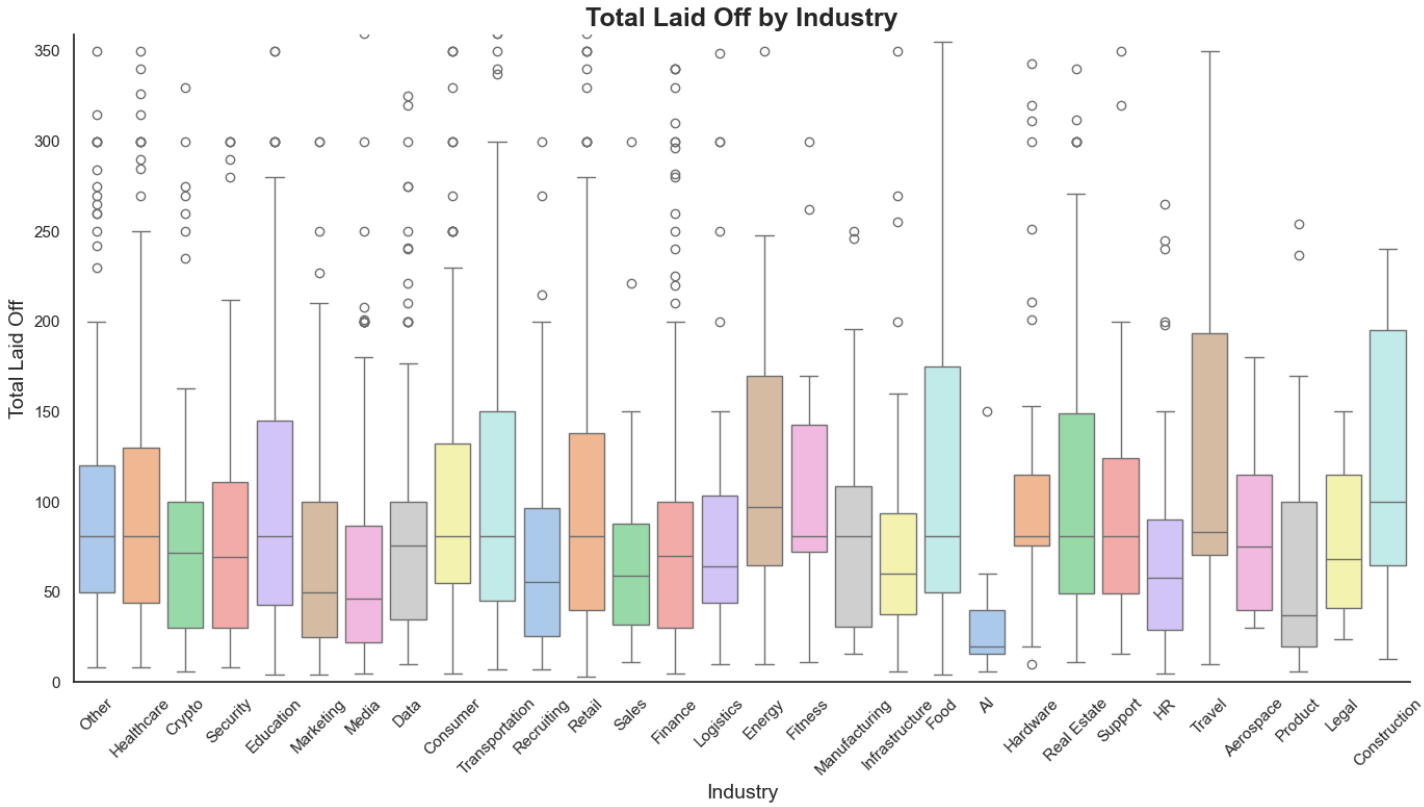
Table 2: Summary Statistics for Categorical Features

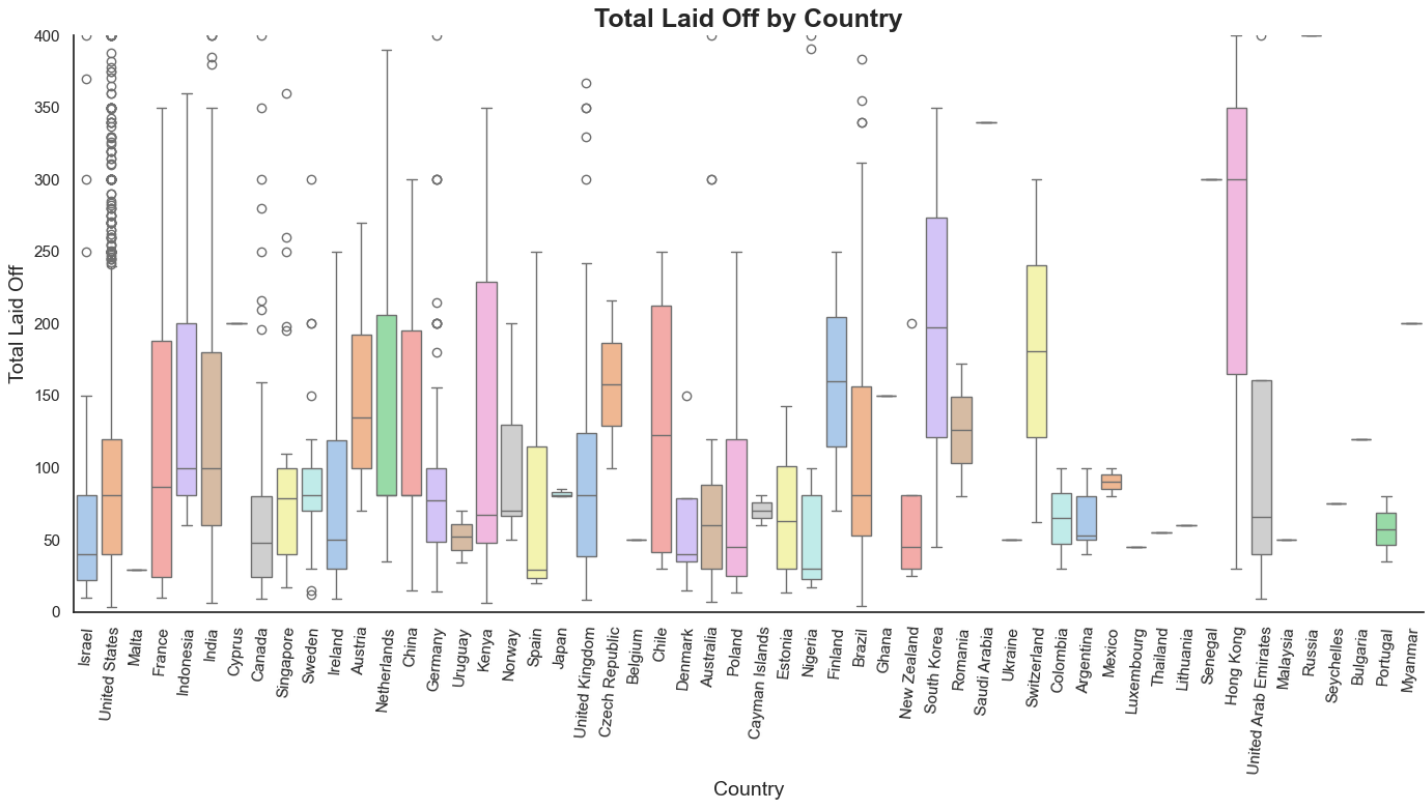
	Count	Unique Values	Most Frequent	Frequency	missing_percentage
company	2511	1846	Amazon	8	0.00%
location	2511	191	SF Bay Area	640	0.00%
industry	2511	30	Finance	314	0.00%
stage	2511	16	Post-IPO	583	0.00%
country	2511	54	United States	1584	0.00%

The tables above provide a detailed summary of numeric and categorical features in the dataset.




Histogram of Total Laid Off
Frequency
Total Laid Off
0
100
200
300
400
500


Total Laid Off by Industry
Total Laid Off
Industry
0
50
100
150
200
250
300
350
Other
Healthcare
Crypto
Security
Education
Marketing
Media
Data
Consumer
Transportation
Recruiting
Retail
Sales
Finance
Logistics
Energy
Fitness
Manufacturing
Infrastructure
Food
AI
Hardware
Real Estate
Support
HR
Travel
Aerospace
Product
Legal
Construction



The figures above illustrate the following:

1. **Fluctuations in Layoffs**: Total number of people laid off over time.
2. **Distribution of Layoffs**: A histogram of total layoffs indicates a left-skewed distribution.
3. **Industry Analysis**: Total number of layoffs categorized by industry type.
4. **Geographical Distribution**: Total layoffs by country.

Models

```
# process data for ML

# based on feature importance, drop unnecessary or repetitive columns
data['month_year'] = data['date'].dt.to_period('M').astype('object')
data = data.drop(['date', 'year', 'month'], axis=1)

# log transform target due to left-skew
data['total_laid_off'] = np.log1p(data['total_laid_off'])


X = data.drop('total_laid_off', axis=1)
y = data['total_laid_off']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=42)
```

```
# Identify numeric and categorical features
numeric_features = X_train.select_dtypes(['int', 'float']).columns.tolist()
cat_features = X_train.select_dtypes(['category', 'object']).columns.tolist()

# Pipeline for numeric features
Numeric_pipeline = Pipeline(steps=[
    ('Standardize', StandardScaler())
])

# Pipeline for categorical features
Categorical_pipeline = Pipeline(steps=[
    ('OneHotEncoder', OneHotEncoder(handle_unknown='infrequent_if_exist', max_categories=5))
])

# Column transformer
preprocessor = ColumnTransformer(transformers=[
    ('Numeric_processing', Numeric_pipeline, numeric_features),
    ('Categorical_processing', Categorical_pipeline, cat_features)
])

# Full pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', DummyRegressor())
])

# Baseline pipeline
baseline_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])
```

```
# Create dummy param grid for baseline testing
baseline_param_grid = {}

# Create param grid for all models
param_grid = [
    {
        "regressor": [DummyRegressor()],
    },
    {
        "regressor": [Lasso(random_state=42)],
        "regressor__alpha": [0.01, 0.1, 1, 10],
    },
    {
        "regressor": [KNeighborsRegressor()],
        "regressor__n_neighbors": [1, 3, 5, 7, 9, 11, 15],
    },
    {
        "regressor": [GradientBoostingRegressor(random_state=42)],
        "regressor__n_estimators": [100, 200, 500],
        "regressor__learning_rate": [0.01, 0.1, 0.05],
        "regressor__max_depth": [3, 5, 7],
    },
      {
        "regressor": [XGBRegressor(random_state=42)],
        "regressor__n_estimators": [100, 200, 500],
        "regressor__learning_rate": [0.01, 0.1, 0.2],
        "regressor__max_depth": [3, 5, 7],
        "regressor__min_child_weight": [1, 2, 3],
    },
    {
        "regressor": [MLPRegressor(max_iter=1000, tol=1e-4, random_state=42)],
        "regressor__hidden_layer_sizes": [(50,), (100,), (50, 50)],
        "regressor__activation": ['relu', 'tanh'],
        "regressor__alpha": [0.0001, 0.001, 0.01],
    },
    {
        "regressor": [VotingRegressor(estimators=[
            ('rf', RandomForestRegressor(random_state=42)),
            ('svr', SVR())
        ])],
    },
    {
        "regressor": [ElasticNet(random_state=42)],
        "regressor__alpha": [0.01, 0.1, 1, 10],
        "regressor__l1_ratio": [0.1, 0.5, 0.9],
    },
    {
        "regressor": [Ridge(random_state=42)],
        "regressor__alpha": [0.1, 1, 10, 100],
    }
]
```

```
# baseline model

baseline_mod = GridSearchCV(
    baseline_pipeline,
    param_grid=baseline_param_grid,
    n_jobs=-1,
    cv=5,
    verbose=0,
    scoring="neg_mean_squared_error",

)

# Fit model on the training data
baseline_mod.fit(X_train, y_train)

# Predict on the training set
baseline_train_y_pred = baseline_mod.predict(X_train)
baseline_train_y_pred_orig_scale = np.expm1(baseline_train_y_pred) # return to original scale

# Predict on the test set
baseline_test_y_pred = baseline_mod.predict(X_test)
baseline_test_y_pred_orig_scale = np.expm1(baseline_test_y_pred) # return to original scale

# Calculate RMSE on the training set
y_train_orig_scale = np.expm1(y_train) # return to original scale
baseline_train_rmse = root_mean_squared_error(y_train_orig_scale, baseline_train_y_pred_orig_scale)

# Calculate RMSE on the test set
y_test_orig_scale = np.expm1(y_test) # return to original scale
baseline_test_rmse = root_mean_squared_error(y_test_orig_scale, baseline_test_y_pred_orig_scale)

if baseline_train_rmse < baseline_test_rmse:
    print("Based on RMSE, the model is overfitting.\n")
elif baseline_train_rmse > baseline_test_rmse:
    print("Based on RMSE, the model is underfitting.\n")
else:
    print("Based on RMSE, the model is neither overfitting nor underfitting.\n")

print(f"Training Root Mean Squared Error: {baseline_train_rmse}")
print(f"Test Root Mean Squared Error: {baseline_test_rmse}")
```

```
Based on RMSE, the model is overfitting.

Training Root Mean Squared Error: 83.69386626027271
Test Root Mean Squared Error: 89.04572941497392
```

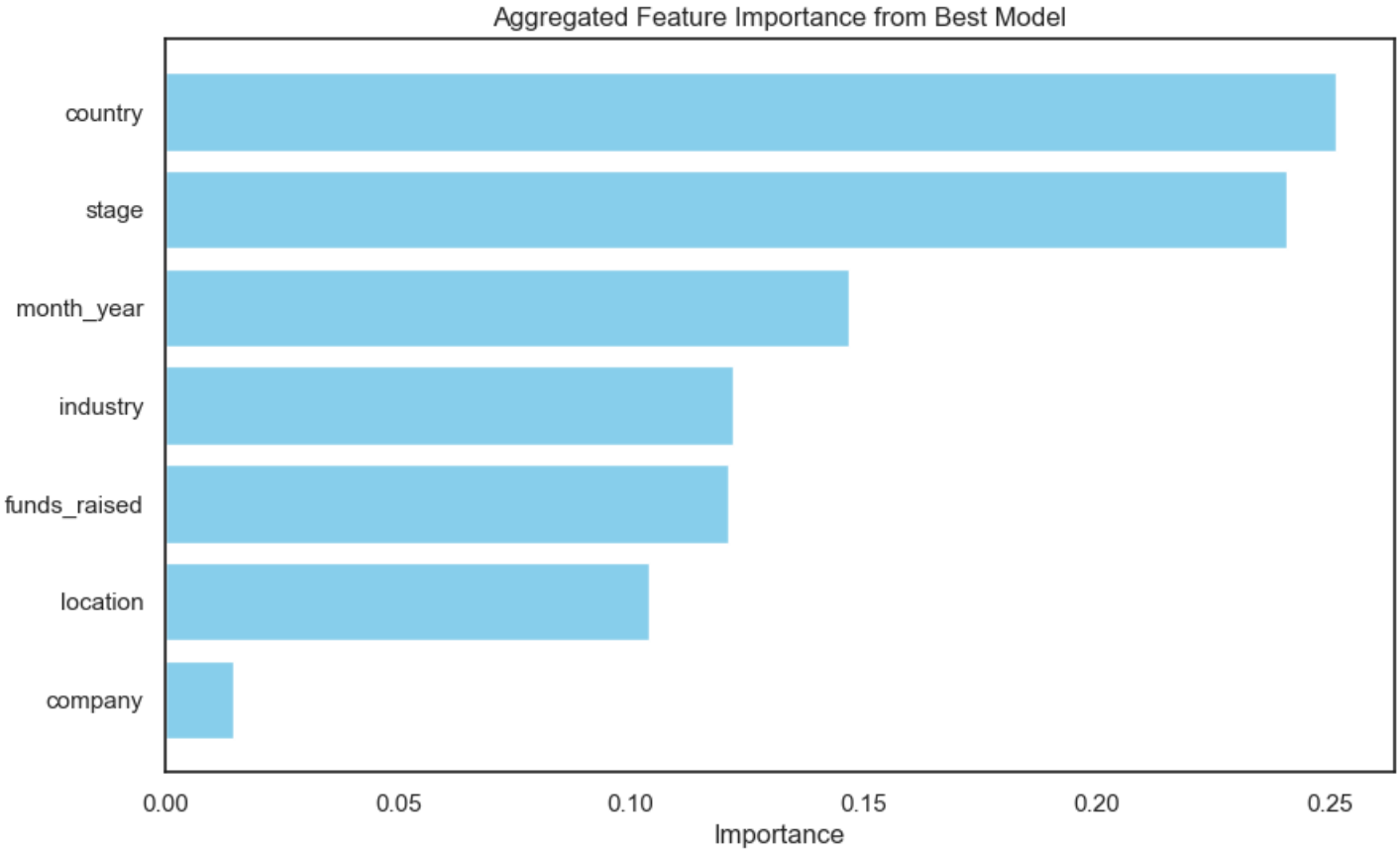
```
# full pipeline

mod = GridSearchCV(
    pipeline,
    param_grid=param_grid,
    n_jobs=-1,
    cv=5,
    verbose=0,
    scoring="neg_mean_squared_error"
)

mod.fit(X_train, y_train)
y_pred = mod.predict(X_test)
train_y_pred = mod.predict(X_train)
```

XGBRegressor was found to be the best model for this dataset after using cross-validation with GridSearchCV.

Results

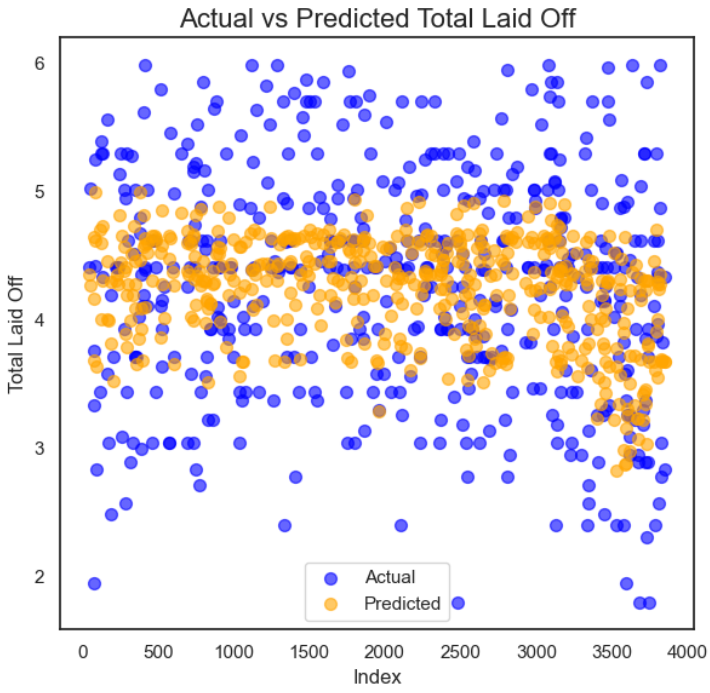
```
# Calculate RMSE on the training set
train_y_pred_orig_scale = np.expm1(train_y_pred)
train_rmse = root_mean_squared_error(y_train_orig_scale, train_y_pred_orig_scale)

# Calculate RMSE on the test set
y_pred_orig_scale = np.expm1(y_pred)
test_rmse = root_mean_squared_error(y_test_orig_scale, y_pred_orig_scale)


# Determine if over- or under-fitting
if train_rmse < test_rmse:
    print("Based on RMSE, the model is overfitting.\n")
elif train_rmse > test_rmse:
    print("Based on RMSE, the model is underfitting.\n")
else:
    print("Based on RMSE, the model is neither overfitting nor underfitting.\n")

print(f"Train RMSE: {train_rmse}")
print(f"Test RMSE: {test_rmse}")
```

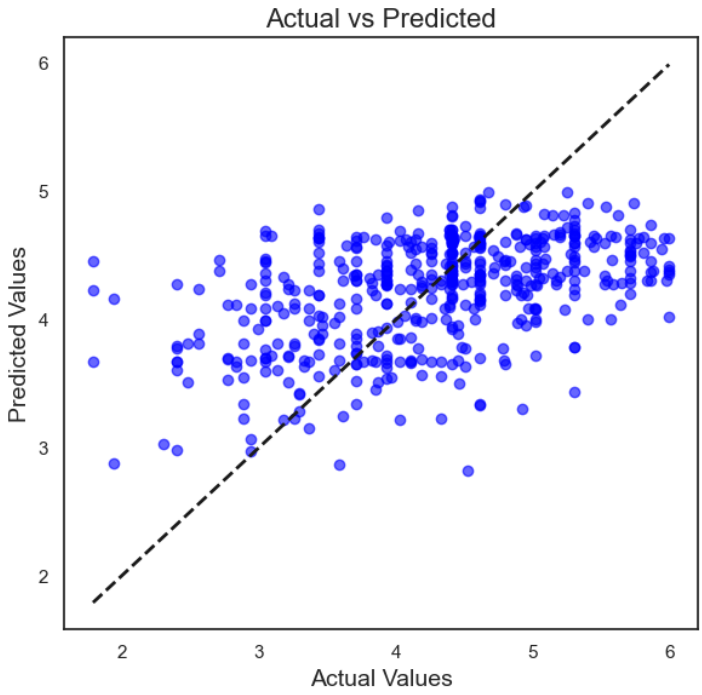
```
Based on RMSE, the model is overfitting.

Train RMSE: 81.5933210428989
Test RMSE: 86.36199885554595
```

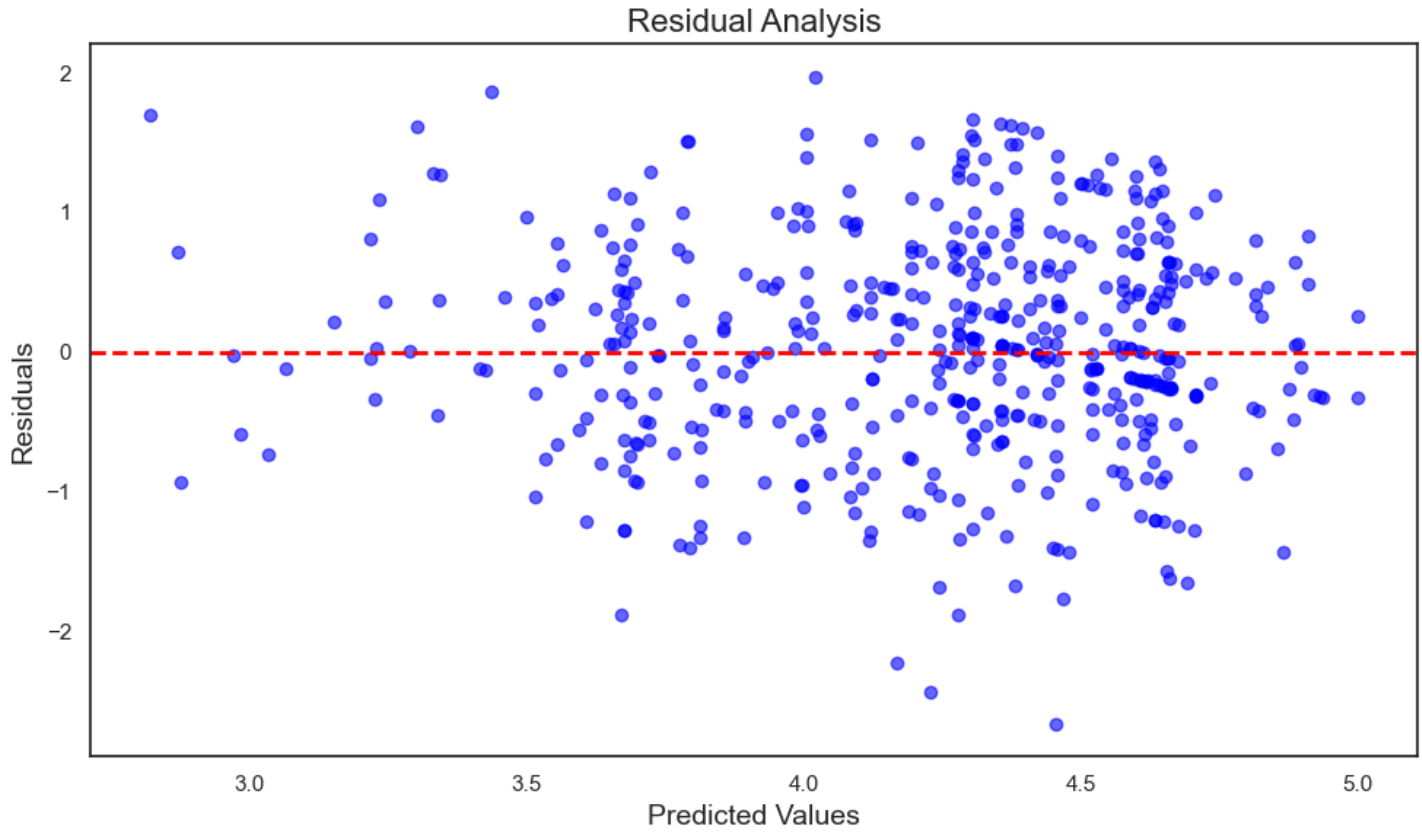


	Feature	Importance
4	country	0.251228
3	stage	0.240737
5	month_year	0.146568
2	industry	0.121939
6	funds_raised	0.120928
1	location	0.103971
0	company	0.014628

Feature importance analysis indicated that funding stage is the most critical predictor of layoffs, followed by the country where the company is located, the industry of the company, the month-year of the layoffs, and the amount of funds raised. The city of the company also contributes marginally, while company title is the least influential predictor.







These visualizations illustrate the model's capacity to predict the number of layoffs accurately. While the model is not flawless, as indicated by the Actual vs. Predicted figure, its performance is relatively adequate. The residual analysis shows that the residuals are mostly randomly distributed around zero, suggesting that the model effectively captures the underlying patterns in the data.

Discussion

The current model is underfitting, as indicated by the training RMSE exceeding the test RMSE. This suggests that it may not be capturing the underlying complexities of the data. To address this, we should explore more sophisticated techniques, particularly neural networks, which excel in modeling nonlinear relationships and feature interactions. Additionally, enhancing feature engineering and applying regularization techniques could help the model learn more effectively while maintaining generalizability. Implementing these strategies may lead to improved predictive accuracy regarding layoffs.