PowerOutage

April 21, 2025

1 Causes of Major Power Outages

Name: Tanish Kalwad

Website Link: https://kalwad.github.io/power-outage-analysis/

[1]:
```
import pandas as pd
import numpy as np

import plotly.express as px
pd.options.plotting.backend = 'plotly'
from lec_utils import *
```

1.1 Step 1: Introduction

Introduction:
This project takes a look at the dataset "Major Power Outage Events in the Continental U.S. (2000–2016)." The dataset has detailed records of large power outages which includes information on their time of occurence, affected states, outage duration, customers affected, and the cause of the outage. The causes section include severe weather, intentional attack, equipment failure, etc.

Research Question:
What factors can predict the cause of a major power outage?

Predicting the causes of power outages is important because it can help utility companies and emergency planners anticipate and respond to outages better and more efficiently. The analysis and predictive modeling done in this project focus on determining which features, (such as the duration of the outage, number of customers affected, etc.), are most related to the underlying cause of the outage.

Dataset Overview:
- **Relevant Columns for our analysis include:**
- `OUTAGE_DURATION_HOURS`: The duration between the start of outage and restoration times. - `CUSTOMERS.AFFECTED`: The amount of customers that were impacted by the outage. - `DEMAND.LOSS.MW`: The recorded loss of demand in megawatts. - `HURRICANE.NAMES`: This indicates whether a hurricane is involved or not. - `CAUSE.CATEGORY`: The target variable (such as, 'severe weather', 'intentional attack', etc.)

1.2 Step 2: Data Cleaning and Exploratory Data Analysis

Data Cleaning and EDA:

After loading in the excel file, dropping unnamed extra columns, and reassiging them to easy-to-read names, I can begin the actual data cleaning.

1. First, I merged date and time into single datetime columns.
2. Then, the outage duration in hours was computed.
3. Missing values were then explored.
4. Finally, univariate and bivariate plots were performed.

[2]:
```python
# loading in excel file
file_path = './outage.xlsx'
df = pd.read_excel(file_path, header=None, skiprows=9)

# dropping unnamed columns
if df.columns[0] == 0 or 'Unnamed: 0' in df.columns:
    df.drop(columns=[df.columns[0]], inplace=True)

# reassigning names to cols for clarity
col_names = [
    "OBS", "YEAR", "MONTH", "U.S._STATE", "POSTAL.CODE", "NERC.REGION",␣
 ↪"CLIMATE.REGION",
    "ANOMALY.LEVEL", "CLIMATE.CATEGORY", "OUTAGE.START.DATE", "OUTAGE.START.
 ↪TIME",
    "OUTAGE.RESTORATION.DATE", "OUTAGE.RESTORATION.TIME", "CAUSE.CATEGORY",
    "CAUSE.CATEGORY.DETAIL", "HURRICANE.NAMES", "OUTAGE.DURATION", "DEMAND.LOSS.
 ↪MW",
    "CUSTOMERS.AFFECTED", "RES.PRICE", "COM.PRICE", "IND.PRICE", "TOTAL.PRICE",
    "RES.SALES", "COM.SALES", "IND.SALES", "TOTAL.SALES", "RES.PERCEN", "COM.
 ↪PERCEN",
    "IND.PERCEN", "RES.CUSTOMERS", "COM.CUSTOMERS", "IND.CUSTOMERS", "TOTAL.
 ↪CUSTOMERS",
    "RES.CUST.PCT", "COM.CUST.PCT", "IND.CUST.PCT", "PC.REALGSP.STATE", "PC.
 ↪REALGSP.USA",
    "PC.REALGSP.REL", "PC.REALGSP.CHANGE", "UTIL.REALGSP", "TOTAL.REALGSP",␣
 ↪"UTIL.CONTRI",
    "PI.UTIL.OFUSA", "POPULATION", "POPPCT_URBAN", "POPPCT_UC", "POPDEN_URBAN",
    "POPDEN_UC", "POPDEN_RURAL", "AREAPCT_URBAN", "AREAPCT_UC", "PCT_LAND",
    "PCT_WATER_TOT", "PCT_WATER_INLAND"
]
assert len(col_names) == df.shape[1], "Column count mismatch"
df.columns = col_names

print(df.shape)
df.head()

# combine start and restoration into datetime
df['OUTAGE.START'] = pd.to_datetime(
```

```
    df['OUTAGE.START.DATE'].astype(str) + ' ' + df['OUTAGE.START.TIME'].
 ↪astype(str), errors='coerce')
df['OUTAGE.RESTORATION'] = pd.to_datetime(
    df['OUTAGE.RESTORATION.DATE'].astype(str) + ' ' + df['OUTAGE.RESTORATION.
 ↪TIME'].astype(str), errors='coerce')

# compute duration in hours
df['OUTAGE_DURATION_HOURS'] = (
    df['OUTAGE.RESTORATION'] - df['OUTAGE.START']
).dt.total_seconds() / 3600

# drop og date/time cols
drop_cols = ["OUTAGE.START.DATE","OUTAGE.START.TIME","OUTAGE.RESTORATION.
 ↪DATE","OUTAGE.RESTORATION.TIME"]
df.drop(columns=drop_cols, inplace=True)

# print missing values
print(df.isnull().sum())

# drop all rows that don't have essential columns
df.dropna(subset=['OUTAGE.START','OUTAGE.RESTORATION','CAUSE.CATEGORY'],␣
 ↪inplace=True)
```

```
(1532, 56)
OBS                       0
YEAR                      0
MONTH                     9
                         ..
OUTAGE.START              9
OUTAGE.RESTORATION       58
OUTAGE_DURATION_HOURS    58
Length: 55, dtype: int64
```

1.3 Step 3: Univariate Analysis

The univariate plot shows the distribution of outage durations. The first histogram shows most outages are short with a long tail of multi-day events succeeding them. The second histogram shows the frequency of power outage cuases. Severe weather and intentional attacks dominate the dataset.

Exploratory Data Analysis:
- I plot the distribution of `OUTAGE_DURATION_HOURS` (univariate analysis).
- I visualize the distribution of `CAUSE.CATEGORY`.
- I produce a box plot of outage duration by year and a scatter plot of outage duration vs. customers affected. - I also compute the mean outage duration per cause category as an aggregate table.

[3]:
```
# first histogram
fig1 = px.histogram(df, x='OUTAGE_DURATION_HOURS', nbins=50,
```

```
                      title='Distribution of Outage Duration (Hours)')
fig1.show()

# second histogram
fig2 = px.histogram(df, x='CAUSE.CATEGORY', title='Cause Category Distribution')
fig2.show()
```

1.4 Step 4: Bivariate Analysis

The first bivariate plot shows the outage duration by year. It is a box plot that reveals year-to-year variability in outage lengths.

The second bivariate plot is a scatterplot that shows duration vs. customers affected. There was a modest positive trend where longer outages tend to affect more customers than shorter outages. This is to be expected.

The aggregate table shows the mean duration by cause category. It essentially shows average outage lengths per cause.

[4]:
```
# Bivariate Boxplot
df['OUTAGE_YEAR'] = df['OUTAGE.START'].dt.year
fig3 = px.box(df, x='OUTAGE_YEAR', y='OUTAGE_DURATION_HOURS',
              title='Outage Duration by Year')
fig3.show()

# Bivariate Scatterplot
fig4 = px.scatter(df, x='OUTAGE_DURATION_HOURS', y='CUSTOMERS.AFFECTED',
                  title='Duration vs. Customers Affected', trendline='ols')
fig4.show()

# Aggregate Table
agg = df.groupby('CAUSE.CATEGORY')['OUTAGE_DURATION_HOURS'].mean().reset_index()
agg.rename(columns={'OUTAGE_DURATION_HOURS':'Mean_Duration'}, inplace=True)
print(agg)
```

```
                 CAUSE.CATEGORY  Mean_Duration
0             equipment failure          30.30
1         fuel supply emergency         224.89
2            intentional attack           7.18
3                     islanding           3.34
4                 public appeal          24.47
5                severe weather          64.74
6  system operability disruption          12.15
```

1.5 Step 5: Baseline Model

Next, I will create a baseline model using logisitic regression. The pipeline consists of StandardScaler + multinomial LogisticRegression.

As you can see below, an accuracy of ~82% was acheived, however there was poor recall on minority classes.

Prediction Problem:
I am trying to predict the **CAUSE.CATEGORY** of a major power outage.

Features Used:
- `OUTAGE_DURATION_HOURS`: Duration of the outage.
- `CUSTOMERS.AFFECTED`: Number of customers impacted.

Type of Problem:
This is a **multiclass classification** problem because the target, `CAUSE.CATEGORY`, has multiple different categorical classes.

Evaluation Metrics:
I can use accuracy as well as precision, recall, and f1-score, macro and weighted averages, to assess the model's performance.

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# prepare the data
features = ['OUTAGE_DURATION_HOURS','CUSTOMERS.AFFECTED']
target   = 'CAUSE.CATEGORY'
df_base = df.dropna(subset=features+[target])
X = df_base[features]; y = df_base[target]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=42)

# building the baseline pipeline
base_pipe = Pipeline([
    ('scaler',StandardScaler()),
   ␣
 ↪('clf',LogisticRegression(multi_class='multinomial',max_iter=1000,random_state=42))
])
base_pipe.fit(X_train,y_train)
y_pred=base_pipe.predict(X_test)
print("Baseline Logistic Regression")
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Baseline Logistic Regression
                               precision    recall  f1-score   support

            equipment failure       0.00      0.00      0.00         4
           intentional attack       0.59      0.97      0.73        31
                    islanding       0.00      0.00      0.00         9
```

```
                 public appeal       0.00      0.00      0.00         5
                severe weather       0.89      0.99      0.93       144
system operability disruption       0.00      0.00      0.00        18

                      accuracy                           0.82       211
                     macro avg       0.25      0.33      0.28       211
                  weighted avg       0.69      0.82      0.75       211

[[  0   2   0   0   2   0]
 [  0  30   0   0   1   0]
 [  0   8   0   0   1   0]
 [  0   1   0   0   4   0]
 [  0   2   0   0 142   0]
 [  0   8   0   0  10   0]]
```

1.6 Step 6: Final Model

Final Model – I can improve the prediction using a Random Forest classifier

To improve upon the baseline, I engineered two additional features:

1. **DEMAND_LOSS_PER_CUSTOMER:**
   This is calculated as ratio of `DEMAND.LOSS.MW` to `CUSTOMERS.AFFECTED`. It gives a measure of the outage impact per affected customer.
2. **IS_HURRICANE:**
   This is a binary indicator that is set to 1 if the `HURRICANE.NAMES` field shows that a hurricane is involved. If not it gives a 0.

The final model uses these engineered features as well as the original two features in a Random Forest classifier. I can perform hyperparameter tuning using GridSearchCV, which searches over the number of estimators and maximum depth, and uses balanced class weights.

Finally, I evaluated the final model on the test set and compared its performance to baseline.

[6]:
```python
# prepare final data modeling
df_final = df.dropna(subset=['OUTAGE_DURATION_HOURS','CUSTOMERS.
 ↪AFFECTED','CAUSE.CATEGORY']).copy()
df_final['DEMAND.LOSS.MW'].fillna(0, inplace=True)
df_final['HURRICANE.NAMES'].fillna('NA', inplace=True)
# feature engineering
df_final['DEMAND_LOSS_PER_CUSTOMER'] = df_final['DEMAND.LOSS.MW']/
 ↪(df_final['CUSTOMERS.AFFECTED']+1e-5)
df_final['IS_HURRICANE'] = df_final['HURRICANE.NAMES'].apply(lambda x: 0 if␣
 ↪str(x).strip() in ['NA',''] else 1)

final_feats = ['OUTAGE_DURATION_HOURS','CUSTOMERS.
 ↪AFFECTED','DEMAND_LOSS_PER_CUSTOMER','IS_HURRICANE']
Xf = df_final[final_feats]; yf = df_final['CAUSE.CATEGORY']
Xtr,Xte,ytr,yte = train_test_split(Xf,yf,test_size=0.2,random_state=42)
```

```python
# random Forest pipeline + grid search
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf_pipe = Pipeline([
    ('scaler',StandardScaler()),
    ('clf',RandomForestClassifier(class_weight='balanced',random_state=42))
])
param_grid = {'clf__n_estimators': [50,100,200], 'clf__max_depth':␣
 ↪[None,10,20,30]}
gs = GridSearchCV(rf_pipe,param_grid,cv=5,scoring='accuracy')
gs.fit(Xtr,ytr)
print("Best Params:", gs.best_params_)
print("CV Acc:", gs.best_score_)

# evaluate final model
y_rf = gs.predict(Xte)
print("\nFinal Random Forest Performance")
print(classification_report(yte,y_rf))
print(confusion_matrix(yte,y_rf))
```

```
Best Params: {'clf__max_depth': None, 'clf__n_estimators': 200}
CV Acc: 0.8305649478726401

Final Random Forest Performance
                              precision    recall  f1-score   support

            equipment failure       0.00      0.00      0.00         4
        fuel supply emergency       0.00      0.00      0.00         0
           intentional attack       0.91      0.97      0.94        31
                    islanding       0.57      0.44      0.50         9
                public appeal       0.67      0.40      0.50         5
               severe weather       0.91      0.96      0.93       144
system operability disruption       0.45      0.28      0.34        18

                     accuracy                           0.85       211
                    macro avg       0.50      0.44      0.46       211
                 weighted avg       0.83      0.85      0.84       211

[[  0   0   0 …   0   2   1]
 [  0   0   0 …   0   0   0]
 [  0   0  30 …   1   0   0]
 …
 [  0   2   0 …   2   1   0]
 [  1   0   1 …   0 138   4]
 [  1   0   0 …   0  10   5]]
```

[7]:

```
fig1.write_html("docs/assets/outage-duration.html", include_plotlyjs='cdn')
fig2.write_html("docs/assets/cause-distribution.html", include_plotlyjs='cdn')
fig3.write_html("docs/assets/duration-by-year.html", include_plotlyjs='cdn')
fig4.write_html("docs/assets/duration-vs-customers.html", include_plotlyjs='cdn')
agg.to_html("docs/assets/agg_table.html", index=False)
```