Predicting Power Outage Severity

Name: Issac Roy

Website Link: https://theboyroy05.github.io/Power-Outages/

In [ ]:
```
import pandas as pd
import numpy as np
from pathlib import Path
from datetime import datetime
import seaborn as sns
import requests
import folium

import plotly.express as px

pd.options.plotting.backend = "plotly"
```

Step 1: Introduction

First, we'll read in the data to take a look at what we're working with

In [284…
```
raw = pd.read_excel(Path("data") / "outage.xlsx")
raw.head()
```

Out[284...

	Major power outage events in the continental U.S.	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6
0	Time period: January 2000 - July 2016	NaN	NaN	NaN	NaN	NaN	NaN
1	Regions affected: Outages reported in this dat...	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	variables	OBS	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION

5 rows × 57 columns

From what we can see here, it's clear that the data needs to be cleaned before we can use it for anything. Taking a look at row 4, we can see the columns that we're working with here including month, year, state, and many of other columns describing each outage.

Brainstorming

Here's a few of the things that I want to explore in this dataset:

- distribution of outages per state
- distribution of outages over time
- correlations between the columns
- patterns which I can predict with an ML model

Here's a few questions that I have:

- Do power outages have any temporal patterns, i.e. could I predict when an outage would occur?
- Do power outages have any spacial patterns, i.e. could I predict where an outage would occur?
- Could I predict the severity of a power outage given it's location and other initial data?

Step 2: Data Cleaning and Exploratory Data Analysis

Data Cleaning

Let's go ahead and clean the data by dropping unnecessary rows and columns. This gets us the following:

In [285…
```
df = (
    raw[6:]
    .reset_index(drop=True)
    .rename(columns=raw.iloc[4])
    .drop(["variables", "OBS"], axis=1)
)
df.head()
```

Out[285…

	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.
0	2011	7	Minnesota	MN	MRO	East North Central	
1	2014	5	Minnesota	MN	MRO	East North Central	
2	2010	10	Minnesota	MN	MRO	East North Central	
3	2012	6	Minnesota	MN	MRO	East North Central	
4	2015	7	Minnesota	MN	MRO	East North Central	

5 rows × 55 columns

Let's also clean up the start and restoration times so that they're combined into one column and are of the the datetime type.

In [ ]:
```
cleaned = df.copy()


def clean_datetime(row, col):
    if pd.notna(row[f"OUTAGE.{col}.DATE"]) and pd.notna(row[f"OUTAGE.{col}.TIME"]):
        return datetime.combine(
            row[f"OUTAGE.{col}.DATE"].date(), row[f"OUTAGE.{col}.TIME"]
        )


for col in ["START", "RESTORATION"]:
    cleaned[f"OUTAGE.{col}"] = df.apply(lambda row: clean_datetime(row, col), axis=
    cleaned.drop([f"OUTAGE.{col}.DATE", f"OUTAGE.{col}.TIME"], axis=1, inplace=True

cleaned[["OUTAGE.START", "OUTAGE.RESTORATION"]].head()
```

Out[ ]:

	OUTAGE.START	OUTAGE.RESTORATION
0	2011-07-01 17:00:00	2011-07-03 20:00:00
1	2014-05-11 18:38:00	2014-05-11 18:39:00
2	2010-10-26 20:00:00	2010-10-28 22:00:00
3	2012-06-19 04:30:00	2012-06-20 23:00:00
4	2015-07-18 02:00:00	2015-07-19 07:00:00

Univariate Analysis

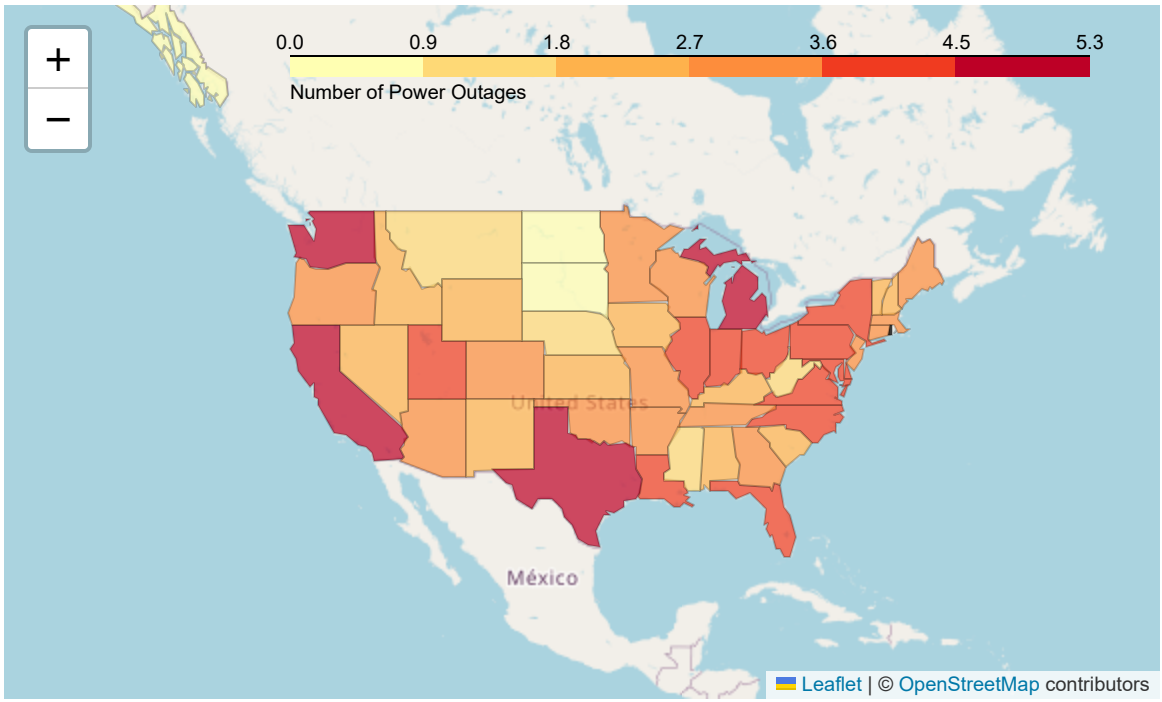
First of all, I would like to explore the distribution of power outages per state. We can do so by using the folium module to plot geographical data. In my opinion this is more interesting to look at than a bar chart. Let's also apply a log scale to the colors since state's with more people and infrastructure will tend to have more power outages. You can hover over each state to see the actual number of outages recorded in this dataset.

In [363…

```
state_geo = requests.get(
    "https://raw.githubusercontent.com/python-visualization/folium-example-data/mai
).json()


def plot_geo(data, legend_name, scale_fn=lambda x: x, tooltip=True):
    m = folium.Map(location=[48, -102], zoom_start=4)

    folium.Choropleth(
        geo_data=state_geo,
        name="choropleth",
        data=data.apply(scale_fn),
        columns=[data.index, data],
        key_on="feature.id",
        fill_color="YlOrRd",
        fill_opacity=0.7,
        line_opacity=0.2,
        legend_name=legend_name,
    ).add_to(m)

    if not tooltip:
        return m

    # Tooltip
    for state_code, value in data.items():
        feature = [f for f in state_geo["features"] if f["id"] == state_code]
        if feature:
            feature[0]["properties"]["power_outages"] = value

    folium.GeoJson(
        state_geo,
        style_function=lambda _: {"color": "transparent"},
        tooltip=folium.GeoJsonTooltip(
```

```
            fields=["name", "power_outages"],
            aliases=["State:", f"{legend_name}:"],
        ),
    ).add_to(m)

    return m


plot_geo(
    cleaned.groupby("POSTAL.CODE").size(),
    legend_name="Number of Power Outages",
    scale_fn=np.log,
)
```

Out[363…



It seems that California has the most power outages in the dataset with Texas, Michigan, and Washington having about half as much. It also seems that the rural states in the middle of the country experience less power outages which makes sense because there's less people and infrastructure out there.

Next, I want to look at the distribution of outages over time. To do so, we can use a histogram to plot the number of outages every year as well as a line graph to plot the number of outages per month. This will give more resolution to the distribution of outages over time.

In [288…

```
import plotly.graph_objects as go

bar_data = cleaned.groupby("YEAR").size()
bar_data.index = pd.to_datetime(bar_data.index, format="%Y")

line_data = (
    cleaned.assign(TIME=cleaned["OUTAGE.START"].dt.strftime("%Y-%m"))
    .groupby("TIME")
```

```
    .size()
)

fig = go.Figure()

fig.add_trace(
    go.Bar(
        x=bar_data.index,
        y=bar_data.values,
        name="Outages per Year",
        marker=dict(color="royalblue"),
        yaxis="y1",
    )
)

fig.add_trace(
    go.Scatter(
        x=line_data.index,
        y=line_data.values,
        name="Outages per Month",
        mode="lines+markers",
        line=dict(color="orange", width=2),
        yaxis="y2",
    )
)

fig.update_layout(
    title="Combined View: Power Outages over Time",
    xaxis=dict(title="Time"),
    yaxis=dict(title="Outages per Year"),
    yaxis2=dict(
        title="Outages per Month",
        overlaying="y",
        side="right",
    ),
    legend_title="Metrics",
    template="plotly_white",
)

fig.show()
```

Doing this reveals that our data is only from 2000-2016. Interstingly, it seems that there were a lot of power outages recorded in 2011. This could be because there were simply a lot of power outages in that year or it could be other factors. For example, data could have been lost or not recorded for other years. Or perhaps more minor power outages were included in 2011 than other years. To see if this is a possibility, we can explore the number of major vs minor power outages over time in the bivariate analysis.

Bivariate Analysis

Let's explore the number of major vs minor outages over time. To do so, I will define a major outage to be in the top 30th percentile of outages in terms of outage duration.

In [289…
```
severity_data = (
    cleaned.assign(
        SEVERITY=cleaned["OUTAGE.DURATION"] > cleaned["OUTAGE.DURATION"].quantile(0
    )
    .groupby(["YEAR", "SEVERITY"])
    .size()
    .reset_index()
    .rename(columns={0: "COUNT"})
    .replace({True: "High", False: "Low"})
)

px.bar(
    severity_data,
    x="YEAR",
    y="COUNT",
    color="SEVERITY",
    title="Severity of Power Outages over Time",
    barmode="group",
)
```

It seems that my hypothesis seems to be on the right track. Although the number of major outages did increase slightly from 2010 to 2011, it wasn't nearly as much as the increase in minor outages in the same time interval.

Another relationship that I would like to explore is the relationship between Demand Loss and Customers Affected during major power outages since it could tell us how strongly different measures of severities are correlated. Since I want to explore this relationship in the context of major outages, I'll only look at outages with more than 100 Customers Affected. Moreover, I'll use a log scale for both Demand Loss and Customers Affected since they scale up exponentially. Finally I'll draw the trendline on the graph as well.

In [290…
```
no_zeros = cleaned[
    (cleaned["DEMAND.LOSS.MW"] > 0) & (cleaned["CUSTOMERS.AFFECTED"] > 100)
]

no_zeros[["DEMAND.LOSS.MW", "CUSTOMERS.AFFECTED"]].dropna().plot(
    kind="scatter",
    x="CUSTOMERS.AFFECTED",
    y="DEMAND.LOSS.MW",
    log_x=True,
    log_y=True,
    trendline="ols",
    trendline_color_override="red",
    trendline_options=dict(log_x=True, log_y=True),
)
```

Interesting Aggregates

One aggregate that I would like to explore is distibution of causes over time. To do so we can aggregate by the Cause Category and Year columms.

In [291…
```
num_outages_per_cause_over_time = pd.pivot_table(
    cleaned,
    index="CAUSE.CATEGORY",
    columns="YEAR",
    values="POSTAL.CODE",
    aggfunc="count",
    fill_value=0,
)

num_outages_per_cause_over_time
```

Out[291…

YEAR	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
CAUSE.CATEGORY											
equipment failure	5	1	0	6	5	3	1	6	9	10	5
fuel supply emergency	0	0	0	0	0	1	1	3	4	3	3
intentional attack	2	0	1	2	0	0	0	0	0	0	0
islanding	0	0	0	0	0	0	0	1	6	4	9
public appeal	0	3	0	1	7	0	2	2	2	10	14
severe weather	13	1	13	30	56	47	54	40	76	45	62
system operability disruption	6	10	3	7	3	4	9	4	14	6	13

It seems that the prevailing cause every year is severe weather except for 2011 where the majority of outages were caused by intentional attacks. Interestingly, it seems that the data only starts counting intentional attacks in 2011 and it's only been going down ever since.

Another interesting aggregate that we could explore is the mean number of customers affected over Climate Region and Month to see if perhaps some climates are affected more seasonally than others.

In [292…
```
mean_customers_affected_per_month_by_climate = pd.pivot_table(
    cleaned,
    index="CLIMATE.REGION",
    columns="MONTH",
    values="CUSTOMERS.AFFECTED",
    aggfunc=lambda x: np.mean(x) / 1000 if not x.empty else 0,
    fill_value=0,
).astype(int)
```

```
mean_customers_affected_per_month_by_climate
```

Out[292…

MONTH	1	2	3	4	5	6	7	8	9	10	11	12
CLIMATE.REGION												
Central	113	87	65	94	76	148	129	228	186	193	49	57
East North Central	72	182	93	116	97	147	125	243	104	129	148	130
Northeast	53	112	131	58	29	132	87	328	100	147	42	108
Northwest	170	43	17	56	0	5	0	164	0	53	104	129
South	240	296	96	133	216	134	79	114	484	138	73	155
Southeast	180	157	133	107	80	148	72	202	318	717	86	51
Southwest	0	24	166	0	30	36	55	0	0	44	0	100
West	538	131	118	64	418	19	123	148	308	148	194	298
West North Central	0	0	0	0	0	123	0	0	0	35	0	24

From this data, we can see that there are generally more customers affected in the summer months for regions such as Cetnral and Northeast while other regions such as the West has more customers affected during winter months.

Step 3: Assessment of Missingness

First, let's take a look at the number of missing values in each column.

In [324…

```
missing = cleaned.apply(lambda x: x.isna().sum()).sort_values(ascending=False)
missing[missing > 0]
```

Out[324…

```
HURRICANE.NAMES          1462
DEMAND.LOSS.MW            705
CAUSE.CATEGORY.DETAIL     471
CUSTOMERS.AFFECTED        443
OUTAGE.DURATION            58
OUTAGE.RESTORATION         58
TOTAL.PRICE                22
COM.SALES                  22
RES.SALES                  22
RES.PRICE                  22
IND.PRICE                  22
COM.PRICE                  22
IND.SALES                  22
RES.PERCEN                 22
COM.PERCEN                 22
TOTAL.SALES                22
IND.PERCEN                 22
POPDEN_UC                  10
POPDEN_RURAL               10
ANOMALY.LEVEL               9
MONTH                       9
CLIMATE.CATEGORY            9
OUTAGE.START                9
CLIMATE.REGION              6
dtype: int64
```

In [346…

```
fig = px.imshow(
    cleaned.isna().astype(int), title="Missingness in the Dataset", height=1000
)
fig.update_traces(showscale=False, coloraxis=None)
```

NMAR Analysis

It seems that Hurricane Names have the most missing values but this is Missing by Design since that column is only used when the cause is a Hurricane. I would also argue that none of the missing columns are NMAR (Not Missing at Random). The missingness of two of the top missing columns, Demand Loss and Customers Affected are likely related to Outage Duration (another measure of severity), which would most likely make it MAR. The missingness of Cause Category Detail likely depends on Cause Category, making it MAR. And the rest of columns seem to have similar amounts of missing values. This likely means that the missingness of these columns depends on when the data was collected, making it MAR as well on.

Missingness Dependency

Two interesting columns to test missingness on are Demand Loss and Cause Details. For both of these, I'll test if they're MAR on the Cause Category. That is, does Cause Category affect whether Demand Loss or Cause Details are missing? I'll use TVD as a test statistic since

I'm comparing distributions for a categorical variable and run a permutation test where I permute the Cause Categories.

In [294…
```
def assess_missingness_on_cause(col, plot=True):
    category_dist = cleaned["CAUSE.CATEGORY"].value_counts(normalize=True)
    missing_dist = cleaned[cleaned[col].isna()]["CAUSE.CATEGORY"].value_counts(
        normalize=True
    )

    if plot:
        fig = px.bar(pd.DataFrame({"Missing": missing_dist, "Total": category_dist}
        fig.update_layout(barmode="group", xaxis={"categoryorder": "total descendin
        fig.show()

    tvd = lambda x, y: np.abs(x - y).sum() / 2
    category_dist = cleaned["CAUSE.CATEGORY"].value_counts(normalize=True).values
    missing_dist = (
        cleaned[cleaned[col].isna()]["CAUSE.CATEGORY"]
        .value_counts(normalize=True)
        .values
    )
    observed = tvd(missing_dist, category_dist)

    stats = [
        tvd(
            cleaned.sample(n=cleaned[col].isna().sum())["CAUSE.CATEGORY"]
            .value_counts(normalize=True)
            .values,
            category_dist,
        )
        for _ in range(1000)
    ]

    pval = (np.array(stats) >= observed).mean()
    result = "Reject" if pval < 0.01 else "Fail to Reject"
    print(f"p-value: {pval}\nHypothesis: {result}")
```

I'll set up our first Hypothesis Test as such:

- H_0: The missingness of Demand Loss is independent of Cause Category
- H_a: The missingness of Demand Loss is depenedent on Cause Category

In [295…
```
assess_missingness_on_cause("DEMAND.LOSS.MW")
```

```
p-value: 0.0
Hypothesis: Reject
```

As you can see above, the resulting p-value was 0 and so we reject the Null at an p-value. As we can see from the bar chart above. The Demand Loss values are more likely to be missing if the Cause Category is sever weather or an intentional attack and less likely to be missing if the Cause was system operability disrution, equipment failure, or islanding.

I'll set up the next Hypothesis Test as such:

- H_0: The missingness of Cause Category Detail is independent of Cause Category
- H_a: The missingness of Cause Category Detail is depenedent on Cause Category

In [296…
```
assess_missingness_on_cause("CAUSE.CATEGORY.DETAIL")
```

```
p-value: 0.0
Hypothesis: Reject
```

As you can see above, we again reject the Null for any p-value. Interestingly, the difference is much more dramatic here with it being much more likely for the detail to be missing if the cause was system operability disruption, public appeal, or islanding while it's much less likely to be missing if the cause was severe weather or intentional attack.

Step 4: Hypothesis Testing

For my regular Hypothesis Test, I'll be testing if the number of outages are dependant on the month to see if there are any seasonal trends. Said more formally, my null and alternative Hypotheses are:

- H_0: The number of power outages that occur is independant of the month (i.e. there is a uniform distribution of outages over months)
- H_a: The number of power outages that occur is dependant on the month (i.e. there isn't a uniform distribution of outages over months)

First, let's plot number of outages per calendar month.

In [297…
```
fig = px.bar(
    cleaned.groupby("MONTH").size(),
    labels={"value": "Number of Outages"},
    title="Outages per Month",
)
fig.update_traces(showlegend=False)
```

As you can see above, it seems that there are more outages during the summer and winter months, but we can't know for sure unless we run a Hypothesis Test. To run this test, I will be sampling from a uniform distribution and I'll using TVD again as my test statistic because we can treat months as categorical variables.

In [298…
```
tvd = lambda x, y: np.abs(x - y).sum() / 2
uniform = np.ones(12) / 12
month_props = cleaned.value_counts("MONTH", normalize=True).values
observed = tvd(month_props, uniform)

stats = np.array([])
for _ in range(1000):
    rand = np.random.choice(12, size=cleaned.shape[0], p=uniform)
    stat = pd.Series(rand).value_counts(normalize=True)
    stats = np.append(stats, tvd(stat, uniform))
```

```
pval = (stats >= observed).mean()
result = "Reject" if pval < 0.01 else "Fail to Reject"
print(f"p-value: {pval}\nHypothesis: {result}")
```

```
p-value: 0.0
Hypothesis: Reject
```

As you can see above, we again reject the null for any p-value. It seems that there may be some seasonal trend in the number of power outages.

Step 5: Framing a Prediction Problem

The question that I'm going to choose to frame a prediction problem around is the severity of power outages. The first thing I would like to look at is patterns in temporal data to see if I could predict how severe an outage is given when it occurs. I looked at many different columns but none of them seem to have any solid temporal pattern from what I could tell. Here's one such example where I look at Demand Loss, Outage Duration, and Customers Affected (all of them on log scales) over time.

In [ ]:
```
def plot_monthly_average(df, cols):
    return px.line(
        df.dropna(subset=cols)
        .assign(TIME=df["OUTAGE.START"].dt.strftime("%Y-%m"))
        .groupby("TIME")[cols]
        .mean(),
        title="Monthly Averages of Outage Metrics",
    )


log = lambda x: np.log(x) if x else np.nan
extra_features = cleaned.assign(
    LOG_DEMAND_LOSS_MW=cleaned["DEMAND.LOSS.MW"].apply(log),
    LOG_OUTAGE_DURATION=cleaned["OUTAGE.DURATION"].apply(log),
    LOG_CUSTOMERS_AFFECTED=cleaned["CUSTOMERS.AFFECTED"].apply(log),
    START_HR=cleaned["OUTAGE.START"].dt.hour,
    START_DAY=cleaned["OUTAGE.START"].dt.day,
    START_DAYOFWEEK=cleaned["OUTAGE.START"].dt.dayofweek,
)

plot_monthly_average(
    extra_features,
    ["LOG_DEMAND_LOSS_MW", "LOG_OUTAGE_DURATION", "LOG_CUSTOMERS_AFFECTED"],
)
```

As you can see, there is a correlation between these different measures of severity. However, there's not much of a solid pattern here to build a prediction model from. Next, let's take a look at a correlation matrix to see if we could create a regression model to predict severity.

In [300…
```
cat_cols = [
    "U.S._STATE",
```

```
    "POSTAL.CODE",
    "NERC.REGION",
    "CLIMATE.REGION",
    "CLIMATE.CATEGORY",
    "CAUSE.CATEGORY",
    "CAUSE.CATEGORY.DETAIL",
    "HURRICANE.NAMES",
]

severity_cols = [
    "DEMAND.LOSS.MW",
    "OUTAGE.DURATION",
    "CUSTOMERS.AFFECTED",
    "LOG_DEMAND_LOSS_MW",
    "LOG_OUTAGE_DURATION",
    "LOG_CUSTOMERS_AFFECTED",
]

corr_matrix = extra_features.drop(cat_cols, axis=1).corr()
corr_matrix = corr_matrix.loc[severity_cols].drop(severity_cols, axis=1)
fig = px.imshow(corr_matrix, title="Correlation Matrix")
fig.update_xaxes(tickangle=45)
```

Although there do seem to be some correlation to these measures of severity, basically all of them are weak or don't help us much. Taking a look at the scale, the correlation coefficiants only range from -0.3 to 0.2, indicating weak correlations. Customers Affected does have a few correlations including Sales and Customers which makes sense, but those are also weak correlations with a correlation coefficiant of less than 0.2.

The columns with the strongest correlations are Year, Outage Start, and Outage Restoration to Log Outage Duration. However, these columns essentially have the same data, which we just explored with the time plot where we concluded that there wasn't much of a pattern to predict from, especially if I want to predict severities of future outages.

Problem Identification

This means that we won't be able to do regression with much degree of accuracy since none of the quantative columns give us much to work with. Instead, I'll turn to classifying outages as High or Low Severity based on their duration. To do so, let's take a look at the distribution of Log Outage Duration and classify each data point whether they are above or below the mean. More specifically, an outage will be classified as High Severity if it's about 15 hours or longer.

In [ ]:
```
MID = extra_features["LOG_OUTAGE_DURATION"].quantile(0.5)


def get_severity(row):
    if np.isnan(row["LOG_OUTAGE_DURATION"]):
        return np.nan
    if row["LOG_OUTAGE_DURATION"] > MID:
```

```
        return "HIGH"
    return "LOW"


severity = extra_features.assign(SEVERITY=extra_features.apply(get_severity, axis=1
fig = severity.plot(x="LOG_OUTAGE_DURATION", kind="hist", nbins=100, color="SEVERIT
fig.add_vline(MID, line_color="red")
```

From the above graph, we can see that the distribution is tri-modal with a spike at 0. This means that we have close to 100 outages that only lasted a minute. After doing some research, it seems that the majority of power outages are only seconds or minutes long. The other two modes of this distribution are centered around 5 hours and 36 hours. Perhaps this could be because if an outage can't be restored after a certain period of time and it's already late, it could be put off until the next morning since it won't affect many people in the middle of the night.

Next, I would like to look at quantative columns that I may be able to predict Outage Duration from. To do this, let's examine the columns with the highest correlations to Log Outage Duration. I'll discard Year, Outage Start, and Outage Restoration since they all essentially have the same data and also won't be very useful for predicting future outage severities.

In [302…
```
corr_matrix.loc["LOG_OUTAGE_DURATION"].apply(abs).sort_values(ascending=False).drop
    ["YEAR", "OUTAGE.START", "OUTAGE.RESTORATION"]
).head()
```

Out[302…
```
PC.REALGSP.USA    0.215041
UTIL.CONTRI       0.200287
PCT_LAND          0.159314
PCT_WATER_TOT     0.159293
RES.SALES         0.141430
Name: LOG_OUTAGE_DURATION, dtype: float64
```

In [304…
```
corr_matrix.loc["OUTAGE.DURATION"].apply(abs).sort_values(ascending=False).drop(
    ["YEAR", "OUTAGE.START", "OUTAGE.RESTORATION"]
).head()
```

Out[304…
```
PCT_LAND          0.121614
PCT_WATER_TOT     0.121605
UTIL.CONTRI       0.102768
PC.REALGSP.USA    0.070885
START_HR          0.070884
Name: OUTAGE.DURATION, dtype: float64
```

We could try using these columns, but they could end up hurting the model rather than helping since weak correlations could lead to overfit data. Even if there's not linear correlation to the other columns, we can still visually examine data for trends. To do so, I've created a function to plot a column and it's relationship to Outage Duration.

In [280…
```
def plot_duration(col, logify=True):
    return (
        severity.groupby(col)["OUTAGE.DURATION"]
        .median()
        .apply(log if logify else lambda x: x)
        .sort_values(ascending=False)
        .reset_index()
        .plot(
            x=col,
            y="OUTAGE.DURATION",
            kind="bar",
            title=f"Median Outage Duration by {col}",
        )
    )
```

In [281…
```
plot_duration("START_HR", logify=False)
```

In [279…
```
plot_duration("START_DAYOFWEEK", logify=False)
```

Doing so reveals some interesting patterns. Firstly, outages that start in the middle of the day tend to last much shorter than outages than begin in the evening and during the night. This is most likely because more effort and resources are put towards restoring an outage that starts in the middle of the day since that would hurt the economy the most because of lost productivity. Similarly, plotting the start day of week reveals that outages that start on the weekend tend to last longer. This is probably also due to the same reason. Outages need to restored faster on weekdays so as to not lose productive working hours.

Step 6: Baseline Model

To create the baseline model, I'll be using the pattern that we noticed above. That is, Start Hour and Start Day of Week both influence how long the outage lasts. I'll also try the other numerical columns that were linearly correlated to outage duration as well. To create the pipeline, I'll be using a RandomForestClassifier on Start Hour and Day of Week while running GridSearchCV to fine-tune my hyperparemeters.

In [ ]:
```
import plotly.graph_objects as go


def plot_accuracy_surface(grid_search):
    df = pd.DataFrame(grid_search.cv_results_)
    grouped = (
        df.groupby(["param_classifier__n_estimators", "param_classifier__max_depth"
            "mean_test_score"
        ]
        .max()
        .reset_index()
    )
    pivot_table = grouped.pivot(
        index="param_classifier__n_estimators",
```

```
        columns="param_classifier__max_depth",
        values="mean_test_score",
    )

    surface = go.Surface(
        z=pivot_table.values, x=pivot_table.columns, y=pivot_table.index
    )
    fig = go.Figure(data=[surface])
    fig.update_layout(
        title="Grid Search Results",
        scene=dict(
            xaxis_title="Max Depth",
            yaxis_title="Number of Estimators",
            zaxis_title="Mean Test Score",
        ),
        autosize=False,
        width=1000,
        height=800,
    )
    fig.show()
```

In [348…

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score

preprocessor = ColumnTransformer(
    [("numeric", "passthrough", ["START_HR", "START_DAYOFWEEK", "PC.REALGSP.USA"])]
    remainder="drop",
)

pipeline = Pipeline(
    [
        ("preprocessor", preprocessor),
        ("classifier", RandomForestClassifier(random_state=42)),
    ]
)

param_grid = {
    "classifier__max_depth": np.arange(1, 11),
    "classifier__n_estimators": np.arange(1, 51),
}

grid_search = GridSearchCV(
    pipeline,
    param_grid,
    scoring=make_scorer(accuracy_score),
    cv=5,
    n_jobs=-1,
    verbose=1,
)

dropped_na = severity.dropna(
    subset=["CLIMATE.REGION", "CAUSE.CATEGORY.DETAIL", "SEVERITY"]
)
```

```
X = dropped_na.drop("SEVERITY", axis=1)
y = dropped_na["SEVERITY"]

grid_search.fit(X, y)
plot_accuracy_surface(grid_search)
print(f"Best score: {grid_search.best_score_}\nBest params: {grid_search.best_param
```

```
Fitting 5 folds for each of 500 candidates, totalling 2500 fits
Best score: 0.6735464315238358
Best params: {'classifier__max_depth': np.int64(4), 'classifier__n_estimators': np.i
nt64(9)}
```

When trying out different models, I discovered that including PC.REALGSP.USA along with the Start Hour and Day of Week increased test accuracy while the other numerical columns generally decreased test accuracy. Plotting the hyperparameter tuning also reveals some interesting trends. Increasing Max Depth too much leads to overfitting and accuracy drops while increasing Number of Estimators doesn't decrease accuracy much, but rather levels off.

Step 7: Final Model

The first thing that I would like to do is impute the Cause Category Details to use them in the model. If I were to use it otherwise, I would have to drop about a third of my data due to the Cause Details being missing. As we saw before, the missingness of Cause Category Details is MAR on Cause Category so I'll probabilistically impute the details on the cause.

In [ ]:
```
import numpy as np


def impute_details(df, plot=True):
    missing_detail = df["CAUSE.CATEGORY.DETAIL"].isna()
    conditional_probs = (
        df.loc[~missing_detail]
        .groupby("CAUSE.CATEGORY")["CAUSE.CATEGORY.DETAIL"]
        .value_counts(normalize=True)
        .unstack(fill_value=0)
    )

    if plot:
        fig = conditional_probs.plot(kind="bar")
        fig.show()

    def sample_from_distribution(row):
        if row["CAUSE.CATEGORY"] not in conditional_probs.index:
            return row["CAUSE.CATEGORY"]

        probs = conditional_probs.loc[row["CAUSE.CATEGORY"]]
        return np.random.choice(probs.index, p=probs.values)

    result = df.dropna(subset=["START_HR", "CLIMATE.REGION", "SEVERITY"])
    result.loc[missing_detail, "CAUSE.CATEGORY.DETAIL"] = df.loc[missing_detail].ap
        sample_from_distribution, axis=1
    )
```

```
    return result


final = impute_details(severity)
```

The Visualization above shows what each Cause Category is composed of in terms of details. For example, 90% of intentional attacks are categorized as vandalism, while other causes such as severe weather has a more diverse set of details. Anyways, I have imputed the details based on the distribution of of the details in each category. Now, I'll include Cause Category and Details in my model by one hot encoding them.

In [365…

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score

cat_transformer = (
    "categorical",
    OneHotEncoder(handle_unknown="ignore"),
    ["CAUSE.CATEGORY.DETAIL", "CAUSE.CATEGORY"],
)

preprocessor = ColumnTransformer(
    [
        cat_transformer,
        ("numeric", "passthrough", ["START_HR", "START_DAYOFWEEK"]),
    ],
    remainder="drop",
)

pipeline = Pipeline(
    [
        ("preprocessor", preprocessor),
        ("classifier", RandomForestClassifier(min_samples_leaf=7, random_state=42))
    ]
)

param_grid = {
    "classifier__max_depth": np.arange(1, 31, 3),
    "classifier__n_estimators": np.arange(1, 51),
}

grid_search = GridSearchCV(
    pipeline,
    param_grid,
    scoring=make_scorer(accuracy_score),
    cv=5,
    n_jobs=-1,
    verbose=3,
)
```

```
X = final.drop("SEVERITY", axis=1)
y = final["SEVERITY"]
grid_search.fit(X, y)
model = grid_search.best_estimator_

plot_accuracy_surface(grid_search)
print(f"Best score: {grid_search.best_score_}\nBest params: {grid_search.best_param
```

```
Fitting 5 folds for each of 500 candidates, totalling 2500 fits
Best score: 0.7838632319950491
Best params: {'classifier__max_depth': np.int64(16), 'classifier__n_estimators': np.
int64(49)}
```

For this model, I noticed that the model performs about the same with and without the PC.REALGSP.USA column, so I decided to remove it for the sake of efficiency. I also tried different combinations of other columns including POSTAL.CODE and UTIL.CONTRI but having more columns tended to decrease accuracy, most likely due to overfitting.

Looking at the hyperparameter tuning graph, we can see that increasing Max Depth and Number of Estimators didn't have much of an effect after a certain point as the graph leveled off at around 78%, an 11% improvement from the baseline model.

Step 8: Fairness Analysis

First, let's take a look at the model's confusion matrix as well as recall, precision, and accuracy.

In [358…

```
from sklearn.metrics import confusion_matrix
import plotly.figure_factory as ff
import numpy as np

labels = ["LOW", "HIGH"]
cm = confusion_matrix(final["SEVERITY"], grid_search.predict(X), labels=labels)
cm_normalized = cm.astype("float") / cm.sum()

fig = ff.create_annotated_heatmap(
    z=cm_normalized,
    x=labels,
    y=labels,
    colorscale="deep_r",
    annotation_text=np.round(cm_normalized, 2),
)

fig.update_layout(
    xaxis_title="Predicted Label",
    yaxis_title="True Label",
    font=dict(size=14),
    yaxis=dict(autorange="reversed"),
    autosize=False,
    width=500,
    height=500,
)
```

```
fig.show()

recall = float(cm[1, 1] / (cm[1, 1] + cm[1, 0]))
precision = float(cm[1, 1] / (cm[1, 1] + cm[0, 1]))
accuracy = float((cm[0, 0] + cm[1, 1]) / cm.sum())
print(f'Recall: {recall:.2f}\nPrecision: {precision:.2f}\nAccuracy: {accuracy:.2f}'
```

```
Recall: 0.84
Precision: 0.76
Accuracy: 0.78
```

It seems that our Recall score was pretty high, indicating that we did well against False Negatives. That is, we rarely predicted that an outage is Low Severity when it was actually High Severity. This is desirable for this type of model since thinking that an outage is Low Severity when in reality, it's High Severity means that you'll plan for the best case scenerio rather than the worst case. This could mean making plans with the expectation that the power will return sooner than when it actually returns.

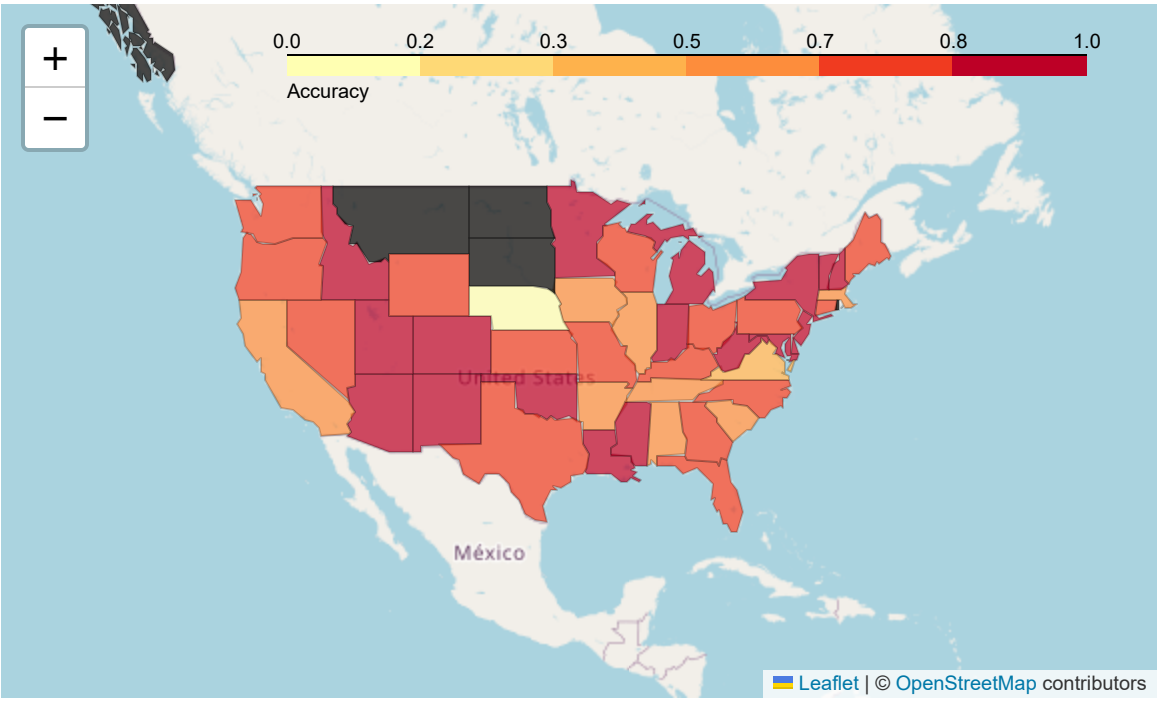
Fairness Analysis

Now, to perform the fairness analysis, I'll check how well the model performs in different States to see if it's more or less accurate in some states. Then, I'll plot it using folium again to visualize accuracy as geographical data.

In [364…

```
def accuracies_by_state(df):
    accuracy = lambda df: (
        model.predict(df.drop("SEVERITY", axis=1)) == df["SEVERITY"]
    ).mean()
    return {
        state: accuracy(df[df["POSTAL.CODE"] == state])
        for state in df["POSTAL.CODE"].unique()
        if len(df[df["POSTAL.CODE"] == state])
    }

dropped_na = severity.dropna(subset=["CAUSE.CATEGORY.DETAIL", "SEVERITY"])
observed_accuracies = pd.Series(accuracies_by_state(dropped_na))
plot_geo(observed_accuracies, legend_name="Accuracy")
```

Out[364...



It seems like we do pretty well in terms of accuracy for most states except for Nebraska for some reason. Regardless, let's run a permuation test to see if this variance is due to random chance or not. I'll be using standard deviation of accuracy as my test statistic while shuffling the postal codes to perform this permutation test. I'll set up the hypotheses as such:

- H_0: The standard deviation in accuracies over states is due to random chance.
- H_a: The standard deviation in accuracies over states is not due to random chance.

In [ ]:
```
from tqdm import tqdm

observed = observed_accuracies.std()
stats = np.array([])

for _ in tqdm(range(1000)):
    shuffled = dropped_na.assign(
        **{"POSTAL.CODE": np.random.permutation(dropped_na["POSTAL.CODE"])}
    )
    stat = pd.Series(accuracies_by_state(shuffled)).std()
    stats = np.append(stats, stat)

pval = (stats >= observed).mean()
result = "Reject" if pval < 0.01 else "Fail to Reject"
print(f"p-value: {pval}\nHypothesis: {result}")
```

```
100%|██████████| 1000/1000 [09:23<00:00,  1.78it/s]
p-value: 0.023
Hypothesis: Fail to Reject
```

It seems that we get a p-value of 0.023 when running this permutation test, allowing us to to reject the null at a significance of 0.05 but not at 0.01. I would also like to see the p-value if

we remove Nebraska from the mix. I'll then plot both observed test statsitics as on the histogram of permutated test statistics.

In [ ]:
```
observed_wo_nebraska = observed_accuracies.drop("NE").std()
pval_wo_nebraska = (stats >= observed_wo_nebraska).mean()

fig = px.histogram(pd.Series(stats), title="Permutation Test Results")
fig.add_vline(x=observed, line_color="red", annotation_text=pval)
fig.add_vline(
    x=observed_wo_nebraska, line_color="blue", annotation_text=pval_wo_nebraska
)
fig.show()
```

This graph reveals that removing Nebraska gives us a p-value of 0.285, meaning we fail to reject the null. That is, it seems that the variation of accuracies of states is mostly likely due to random chance except for Nebraska. I wonder what's going on in Nebraska.