Analyzing the Impact of Climate and Regional Information on Power Outages

Name: Sweekrit Bhatnagar

Website Link: https://sweekrit-b.github.io/powerOutageML/

In [403…

```
from dsc80_utils import * # Feel free to uncomment and use this.

# Import basic libraries
import pandas as pd
import numpy as np
from pathlib import Path
import os
import seaborn as sns
import math
import tabulate

# Import plotting libraries
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import matplotlib.pyplot as plt
pd.options.plotting.backend = 'plotly' # Set pandas plotting backend to plotly

# Import ML libraries
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Step 1: Introduction

Choice of dataset

In this project, I analyze major outage events witnessed in the continental US from January 2000 to July 2016, where a "major outage" refers to an outage that impacted at least 50,000 customers or caused a firm load loss of 300 MW. In addition to the outages' characteristics, the dataset also contains information regarding regional climate information, land-use, electricity consumption patterns, and economic characteristics.

I chose this dataset because it yields itself to interesting analysis on variables that predict outage, especially considering the diversity and amount of geographic, climate, and economic information available. Furthermore, the dataset itself is important due to the **real world consequences** of power outages - these events have major societal impact and affect key infrastructure, including hospitals, transport systems, communication networks, etc. Working with this dataset allows us to do important data analysis on the causes of these events, and inform policymaking and mitigation strategies in order to product cities and homes from major power outages.

Questions brainstorm and selection

While looking at the dataset, I thought of the following questions that I could analyze:

1. What is the background regional and climate information (i.e. the environment) of major power outages of varying severity and cause?
2. How do socioeconomic conditions of the region affect the severity of outages as measured by the duration and number of people affected?
3. Is there a distinct geographic association with coastal regions and power outage cause and/or severity? What background characteristics play into the distinction?

However, I ultimately chose to go with the first question: **What is the background regional and climate information (i.e. the environment) of major power outages of varying severity and cause?**

Specifically, I plan to look into how various features that can affect regional climate (such as the geographic region, the month the outage occurred, the presence of a hurricane, etc.) play into the impact characteristics of outages, focusing on outage duration and the cause category.

Reasons why this question is important:

1. Different climate patterns can strongly influence storm severity, equipment stress, grid load, etc. and can be key preidctors about whether an area is affected by a major power outage.
2. Climate change sees an increased variance of climate events - being able to predict power outages quickly before they happened can be important to save infrastucture in times of climate disaster (ex. hurricanes or major storms).

Description of key columns and overall shape of dataset

There are **1534 rows** in the dataset, indicating the presence of 1534 major outage events that occurred in the time period of the study, of which we find the following features relevant:

Category	Variable	Description

Category	Variable	Description
General Information	YEAR	Year when the outage event occurred
	MONTH	Month when the outage event occurred
	U.S._STATE	U.S. state where the outage occurred
	POSTAL.CODE	Postal code of the U.S. state
	NERC.REGION	NERC region involved in the outage
Regional Climate Information	CLIMATE.REGION	U.S. climate region as defined by the National Centers for Environmental Information (9 regions total)
	ANOMALY.LEVEL	Oceanic Niño/La Niña index (ONI), 3-month running mean of SST anomalies
	CLIMATE.CATEGORY	Climate category ("Warm," "Cold," or "Normal") based on ONI index ±0.5°C
Outage Event Information	OUTAGE.START.DATE	Calendar day when the outage started
	OUTAGE.START.TIME	Time of day when the outage started
	OUTAGE.RESTORATION.DATE	Calendar day when power was fully restored
	OUTAGE.RESTORATION.TIME	Time of day when power was fully restored
	CAUSE.CATEGORY	High-level category describing the cause of the outage
	CAUSE.CATEGORY.DETAIL	Detailed description of the event cause
	HURRICANE.NAMES	Hurricane name if the outage was caused by a hurricane
	OUTAGE.DURATION	Duration of the outage in minutes
	DEMAND.LOSS.MW	Peak demand lost during the outage (megawatts)
	CUSTOMERS.AFFECTED	Number of customers impacted by the outage
Regional Land-Use Characteristics	POPULATION	Population of the U.S. state in the given year
	POPPCT_URBAN	Percentage of the population living in urban areas
	POPPCT_UC	Percentage of the population living in urban clusters
	POPDEN_URBAN	Population density of urban areas (persons/sq. mile)

Category	Variable	Description
	POPDEN_UC	Population density of urban clusters (persons/sq. mile)
	POPDEN_RURAL	Population density of rural areas (persons/ sq. mile)
	AREAPCT_URBAN	Percentage of the state's land area classified as urban
	AREAPCT_UC	Percentage of the state's land area classified as urban clusters
	PCT_LAND	Percentage of total U.S. land area represented by the state
	PCT_WATER_TOT	Percentage of total U.S. water area represented by the state
	PCT_WATER_INLAND	Percentage of total U.S. inland water area represented by the state

In [404…
```
# Load the data for analysis - outage is data is marked as cleaned because I removed irrelevant rows at the top of the Excel sheet that contained study information.
# Note, I did not do any other cleaning here; it is all part of the project work.
data = pd.read_csv('outage_cleaned.csv')
# Determine the number of rows in the dataset
data.shape[0]
```

Out[404… 1534

Step 2: Data Cleaning and Exploratory Data Analysis

I took the following steps to clean my data. Note that in this step, I DID NOT impute `NaN` values, as those will be important for future missingness analysis:

1. Set the index of the dataset using a unique identifier `OBS` instead of sticking with the default sequence. This allowed for a cleaner dataset that did not have a duplicate index variable, preventing its inclusion in exploratory data analysis (as it would just be a static range) or predictions.
2. Perform datetime conversions. There seem to be separate columns for the calendar date and the time - combining these would make future analysis easier, and converting to datetime would allow for them to become quantitative discrete variables that I can use in future analysis if I choose to. Specifically, I combine `OUTAGE.START.DATE` and `OUTAGE.START.TIME` and `OUTAGE.RESTORATION.DATE` and `OUTAGE.RESTORATION.TIME` and convert them using `pd.to_datetime()`
3. Drop any unnecessary and/or redundant columns. Although in Steps 3-8 I am already filtering for the columns that I want, this will make my data cleaner and more easily

perform the univariate and bi-variate analysis of this step. The columns I drop are `OUTAGE.START.TIME`, `OUTAGE.RESTORATION.TIME`, and `U.S._STATE`.

- I drop `OUTAGE.START.TIME` and `OUTAGE.RESTORATION.TIME` because their data has already been included into `OUTAGE.START.DATE` and `OUTAGE.RESTORATION.DATE`
- I drop `U.S._STATE` because it contains the same information as `POSTAL.CODE`
- Note that although I am looking at specifically climate and regional information, I don't drop any more columns. This is because in the case that my research question changes further along in the project to account for economic factors (for example, in Step 8), I want the freedom to access these values.

4. I replace 0 values in the columns that I am using to look at my impact with `NaN`. These include `CUSTOMERS.AFFECTED` and `DEMAND.LOSS.MW`. This is because in the data specifications, we know that the amount of customers affected is at least 50,000 and the demand lost is at least 300 MW.
    - I do not replace 0 values in `OUTAGE.DURATION`. Although it is unlikely that a major outage event lasts 0 minutes there is no indication that this is a requirement for the dataset, and it is possible that power was restored immediately.

In [405…
```
# Step 1 - set the observation number to be the index of the dataset
data = data.set_index('OBS')

# Step 2 - datetime conversions.
# Step 2.1 - Convert the OUTAGE.START.DATE and OUTAGE.START.TIME to datetime
data['OUTAGE.START.DATE'] = pd.to_datetime(data['OUTAGE.START.DATE'] + ' ' +
data['OUTAGE.START.TIME'], format='mixed')
# Step 2.2 - Convert the OUTAGE.RESTORATION.DATE and OUTAGE.RESTORATION.TIME to
datetime
data['OUTAGE.RESTORATION.DATE'] = pd.to_datetime(data['OUTAGE.RESTORATION.DATE'] +
' ' + data['OUTAGE.RESTORATION.TIME'], format='mixed')

# Step 3- drop redundant/unnecessary columns
# Drop the OUTAGE.START.TIME and OUTAGE.RESTORATION.TIME columns
data = data.drop(columns=['OUTAGE.START.TIME', 'OUTAGE.RESTORATION.TIME'])
# Drop the US state column as it is redundant
data = data.drop(columns=['U.S._STATE'])

# Step 4 - replace 0 values in impact columns with NaN
impact_columns = ['CUSTOMERS.AFFECTED', 'DEMAND.LOSS.MW']
for col in impact_columns:
    data[col] = data[col].replace(0, np.nan)
```

In [ ]:
```
# Print the head of the data for the report
print(data.head().to_markdown(index=False))
```

From here, I need to perform univariate and bivariate analysis. I wanted to create general functions to plot and save graphs and grab relevant ones in the notebook when I needed to. I defined the following functions:

1. `plot_against_categorical_var` plots a boxplot of a categorical variables (such as the U.S. climate region) against a numerical metric (such as the outage duration).
2. `plot_against_numerical_var` plots a scatter plot of a numerical variable (such as the anomaly level of El Nino or La Nina) against another numerical metric (such as the amount of customers affected).
3. `plot_bivariate` uses the basis that the above two functions formed to create a figure with three subplots (and therefore perform an all-encompassing bivariate analysis) for a specified column in the dataset. Specifically, as defined in the problem statement above, I want to see how various climate-based factors affect outage impact factors, so for each column, I made a subplot comparing it to outage duration, demand loss (in Megawatts), and the customers affected.
4. `plot_univariate` plots a univariate analysis of a column - if the data was categorical (such as the U.S. climate region), a histogram was plotted to demonstrate the category density. If the data was numerical, then the Seaborn library was used in conjunction with Matplotlib to create a KDE plot.
5. `save_figure` saves an image to a folder for easy access and usage.
6. `save_combined_univariate` plots a univariate analysis of all of the column in the dataset and creates a figure with all of them at once.

Note that for a lot of these graphs, I noticed that outlier values, especially those in the impact metrics, were making the values difficult to interpret. Therefore, even though I did not remove them from the data, for the purpose of understanding the patterns present in the data better, I included an `exclude_top_pct` variable. Please note that this DOES NOT impact any future analysis, all it does is reveal patterns in graphs that would normally just look like uninterpretable lines with a few plotted outliers.

In [407…

```
# Plot a boxplot of a numerical column against a categorical variable, excluding
the top X% of values
def plot_against_categorical_var(categorical_var_col, column_name,
exclude_top_pct=0.05):
    threshold = data[column_name].quantile(1 - exclude_top_pct)  # grab the
threshold value for top X%
    filtered = data[data[column_name] <= threshold] # filter data to exclude top
X%
    title = f'{column_name} vs {categorical_var_col} (excluding top
{exclude_top_pct*100:.0f}% values)' # title for the plot
    fig = px.box(filtered, x=categorical_var_col, y=column_name, title=title) #
use plotly express to create boxplot
    fig.show() # show the figure

# Plot a scatterplot of a numerical column against another numerical variable,
excluding the top X% of values
def plot_against_numerical_var(numerical_var_col, column_name,
exclude_top_pct=0.05):
    threshold = data[column_name].quantile(1 - exclude_top_pct) # grab the
threshold value for top X%
```

```
    filtered = data[data[column_name] <= threshold] # filter data to exclude top
X%
    title = f'{column_name} vs {numerical_var_col} (excluding top
{exclude_top_pct*100:.0f}% values)' # title for the plot
    fig = px.scatter(filtered, x=numerical_var_col, y=column_name,
trendline='ols', trendline_color_override='red', title=title) # use plotly express
to create scatterplot
    fig.show() # show the figure

# Plot bivariate analysis of a column against three metrics using subplots for a
combined analysis
def plot_bivariate(column, categorical = True, exclude_top_pct=0.05,
save_folder="bivariate_analysis", df=data):
    metrics = ['OUTAGE.DURATION', 'DEMAND.LOSS.MW', 'CUSTOMERS.AFFECTED'] # outage
impact metrics to plot against
    subplot_titles = metrics # title variable for subplots

    os.makedirs(save_folder, exist_ok=True) # ensure folder that plots are being
saved to exists
    print(f"Plotting {column}...") # indicate start of plotting for debugging

    fig = make_subplots(rows=1, cols=3, subplot_titles=subplot_titles,
vertical_spacing=1) # create subplots with 1 row and 3 columns

    for i, metric in enumerate(metrics): # iterate over metrics
        # Filter out top fraction using the same logic as above plotting functions
        threshold = df[metric].quantile(1 - exclude_top_pct) # calculate threshold
for top X%
        filtered = df[df[metric] <= threshold] # filter data to exclude top X%

        if categorical: # if the column is defined as categorical in the function
call
            # Boxplot for categorical x-axis
            for category in filtered[column].unique(): # iterate over unique
categories
                fig.add_trace( # add boxplot trace for each category
                    go.Box(
                        y=filtered[filtered[column] == category][metric], # y
values for the specific category within the column
                        name=str(category), # name of the category
                        boxmean='sd', # show mean and standard deviation
                        showlegend=(i==0)  # show legend only for first subplot
                    ),
                    row=1, col=i+1 # specify row and column for subplot
                )
        else: # if the column is defined as numerical in the function call
            # Scatterplot with line of best fit
            scatter_fig = px.scatter(filtered, x=column, y=metric) # create
scatterplot using plotly express
            for trace in scatter_fig.data: # iterate over traces in scatterplot
                fig.add_trace(trace, row=1, col=i+1) # add scatterplot trace to
subplot

    fig.update_layout(
        height=600,
        width=1800,
```

```
        title_text=f"Metrics vs {column} (excluding top {exclude_top_pct*100:.0f}%
values)",
        title_x=0.2,              # centers the title horizontally
        title_y=1,             # moves the title up (default is ~0.95)
        title_font_size=24,       # optional, larger font
        title_pad=dict(t=10),      # adds padding (pixels) between title and
subplots
        margin=dict(t=50, l=50, r=50, b=50)  # top, left, right, bottom
    )

    for ann in fig['layout']['annotations']:
        ann['y'] -= 0.05  # adjust this number as needed

    print(f"Finished plotting {column}.") # indicate end of plotting
    return fig # return the figure object

# Plot univariate analysis of a column - histogram for categorical, KDE for
numerical
def plot_univariate(column_name, categorical=False, df=data):
    if categorical: # if the column is defined as categorical in the function call
        # Histogram/bar chart for categorical data
        fig = go.Figure() # create empty figure
        fig.add_trace( # add histogram trace
            go.Histogram(
                x=df[column_name],
                name=f'{column_name} counts',
            )
        )
        fig.update_layout( # update layout of figure
            title_text=f'Histogram of {column_name}',
            xaxis_title=column_name,
            yaxis_title='Count',
            width=600, height=400
        )
    else:
        values = df[column_name].values # get values of the column

        kde_fig = sns.kdeplot(values, bw_method='scott') # create KDE plot using
seaborn
        kde_data = kde_fig.get_lines()[0].get_data() # get KDE data points
        x_grid = kde_data[0] # x values of the KDE plot
        kde_values = kde_data[1] # y values of the KDE plot
        kde_fig.figure.clear()  # clear the seaborn figure for future plots

        fig = go.Figure() # create empty figure
        fig.add_trace( # add scatterplot trace from the values derived from the
KDE
            go.Scatter(
                x=x_grid,
                y=kde_values,
                fill='tozeroy',
                name='KDE'
            )
        )
        fig.update_layout( # update layout of figure
            width=600, height=400,
```

```
            title_text=f'KDE of {column_name}',
            xaxis_title=column_name,
            yaxis_title='Density',
            showlegend=False
        )

    return fig

# Save figure to specified folder
def save_figure(fig, column_name, save_folder):
    os.makedirs(save_folder, exist_ok=True) # ensure folder exists
    save_path = os.path.join(save_folder, f"{column_name}.png") # construct save
path
    fig.write_image(save_path) # save figure as PNG
    print(f"Saved figure to {save_path}") # indicate where figure was saved

# Create a combined univariate analysis figure for multiple columns
def save_combined_univariate(columns_to_plot, save_path="univariate_analysis/
combined.png", cols_per_row=5):
    n_cols = cols_per_row # number of columns per row
    n_rows = math.ceil(len(columns_to_plot) / n_cols) # calculate number of rows
needed

    # Create subplot titles
    subplot_titles = [col for col, _ in columns_to_plot]

    fig = make_subplots(rows=n_rows, cols=n_cols, subplot_titles=subplot_titles) #
create a figure with subplots

    for i, (column_name, categorical) in enumerate(columns_to_plot): # iterate
over columns to plot
        row = i // n_cols + 1 # calculate row index
        col = i % n_cols + 1 # calculate column index

        # Create figure for this column
        col_fig = plot_univariate(column_name, categorical=categorical)

        # Add traces to the combined figure
        for trace in col_fig.data:
            fig.add_trace(trace, row=row, col=col)

        # Adjust x-axis and y-axis titles for each subplot
        fig.update_xaxes(title_text=column_name, row=row, col=col)
        fig.update_yaxes(title_text='Count' if categorical else 'Density',
row=row, col=col)

    fig.update_layout(
        height=n_rows * 400,
        width=n_cols * 600,
        showlegend=False
    ) # update layout of the combined figure to be more readable

    os.makedirs(os.path.dirname(save_path), exist_ok=True) # ensure save directory
exists
    fig.write_image(save_path) # save the combined figure as PNG
    print(f"Saved combined figure to {save_path}") # indicate where figure was
```

```
saved
    return fig # return the combined figure object
```

In [ ]:
```
# Specify columns to plot in a list of tuples (column_name, is_categorical)
columns_to_plot = [
    # Time of year
    ('MONTH', True),

    # Geographic region information
    ('NERC.REGION', True),
    ('POSTAL.CODE', True),

    # Regional Climate Information
    ('CLIMATE.REGION', True),
    ('CLIMATE.CATEGORY', True),
    ('ANOMALY.LEVEL', False),

    # Event causes
    ('CAUSE.CATEGORY', True),
    ('HURRICANE.NAMES', True),

    # Electricity price information
    ('RES.PRICE', False),
    ('COM.PRICE', False),
    ('IND.PRICE', False),

    # Electricity consumption information
    ('RES.SALES', False),
    ('COM.SALES', False),
    ('IND.SALES', False),
    ('RES.PERCEN', False),
    ('COM.PERCEN', False),
    ('IND.PERCEN', False),

    # Customers served information
    ('RES.CUSTOMERS', False),
    ('COM.CUSTOMERS', False),
    ('IND.CUSTOMERS', False),

    # Regional economic output information
    ('PC.REALGSP.STATE', False),
    ('PC.REALGSP.USA', False),
    ('PC.REALGSP.REL', False),
    ('PC.REALGSP.CHANGE', False),
    ('UTIL.REALGSP', False),
    ('TOTAL.REALGSP', False),
    ('UTIL.CONTRI', False),
    ('PI.UTIL.OFUSA', False),

    # Population information
    ('POPULATION', False),
    ('POPPCT_URBAN', False),
    ('POPPCT_UC', False),
    ('POPDEN_URBAN', False),
    ('POPDEN_UC', False),
    ('POPDEN_RURAL', False),
```

```
    # Land information
    ('AREAPCT_URBAN', False),
    ('AREAPCT_UC', False),
    ('PCT_LAND', False),
    ('PCT_WATER_TOT', False),
    ('PCT_WATER_INLAND', False)
]

# Specify output columns to plot
output_columns = [('OUTAGE.DURATION', False), ('DEMAND.LOSS.MW', False),
('CUSTOMERS.AFFECTED', False)]

# Plot and save bivariate and univariate analyses for specified columns
for col, is_categorical in columns_to_plot:
    fig1 = plot_bivariate(col, categorical=is_categorical) # create the figure for
bivariate analysis
    fig2 = plot_univariate(col, categorical=is_categorical) # create the figure
for univariate analysis
    save_figure(fig1, col, save_folder="bivariate_analysis") # save the bivariate
figure
    save_figure(fig2, col, save_folder="univariate_analysis")  # save the
univariate figure

for col, is_categorical in output_columns:
    fig = plot_univariate(col, categorical=is_categorical) # create the figure for
univariate analysis
    save_figure(fig, col, save_folder="univariate_analysis") # save the univariate
figure

save_combined_univariate(columns_to_plot + output_columns,
save_path="univariate_analysis/all_univariate.png", cols_per_row=5) # save
combined univariate figure
print("Finished!")
```

```
Finished!
```

In [409…

```
# Convert a plotly figure to HTML and save it to the assets folder for the report
def convert_to_html(fig, title):
    # Ensure the assets directory exists
    os.makedirs('assets', exist_ok=True)

    # Use raw string or forward slashes to avoid backslash issues on Windows
    fig.write_html(f'assets/{title}.html', include_plotlyjs='cdn', full_html=True,
config={'responsive': True})

# Bivariate plots to convert to HTML
fig_climate_region = plot_bivariate('CLIMATE.REGION', categorical=True)
convert_to_html(fig_climate_region, 'bivariate_climate_region')

fig_climate_category = plot_bivariate('CLIMATE.CATEGORY', categorical=True)
convert_to_html(fig_climate_category, 'bivariate_climate_category')

fig_month = plot_bivariate('MONTH', categorical=True)
convert_to_html(fig_month, 'bivariate_month')
```

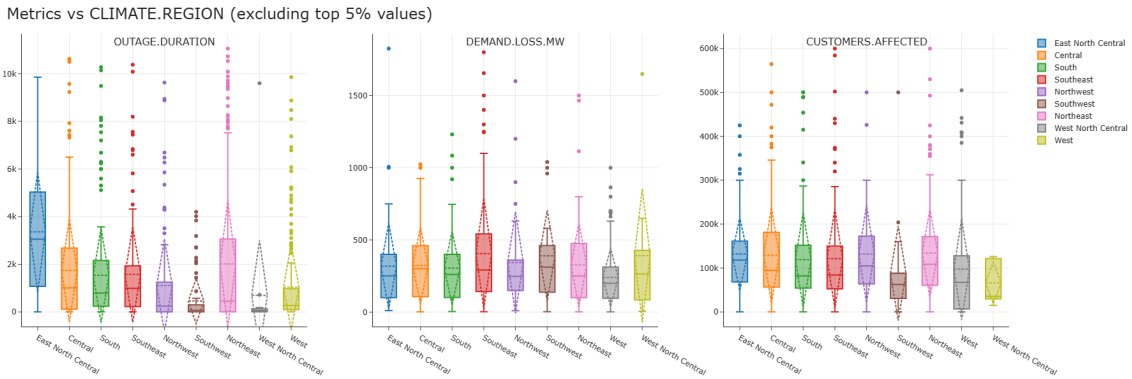
```
# Univariate plots to convert to HTML
fig1 = save_combined_univariate(columns_to_plot=[('OUTAGE.DURATION', False),
('DEMAND.LOSS.MW', False), ('CUSTOMERS.AFFECTED', False)],
save_path="univariate_analysis/output_combined.png", cols_per_row=3)
convert_to_html(fig1, 'univariate_output_combined')

fig2 = save_combined_univariate(columns_to_plot=[('CLIMATE.REGION', True),
('CLIMATE.CATEGORY', True), ('MONTH', True)], save_path="univariate_analysis/
climate_time_combined.png", cols_per_row=3)
convert_to_html(fig2, 'univariate_climate_time_combined')
```

```
Plotting CLIMATE.REGION...
Finished plotting CLIMATE.REGION.
Plotting CLIMATE.CATEGORY...
Finished plotting CLIMATE.CATEGORY.
Plotting MONTH...
Finished plotting MONTH.
Saved combined figure to univariate_analysis/output_combined.png
Saved combined figure to univariate_analysis/climate_time_combined.png
<Figure size 1000x500 with 0 Axes>
```

In order to showcase the graphs as part of the analysis, I have uploaded some relevant plots here and describe/interpret the trends that are present. Note that these graphs were generated using the above functions and saved in a separate folder, which I then pulled from to place these images in the Jupyter Notebook.

Bivariate analysis

Climate region

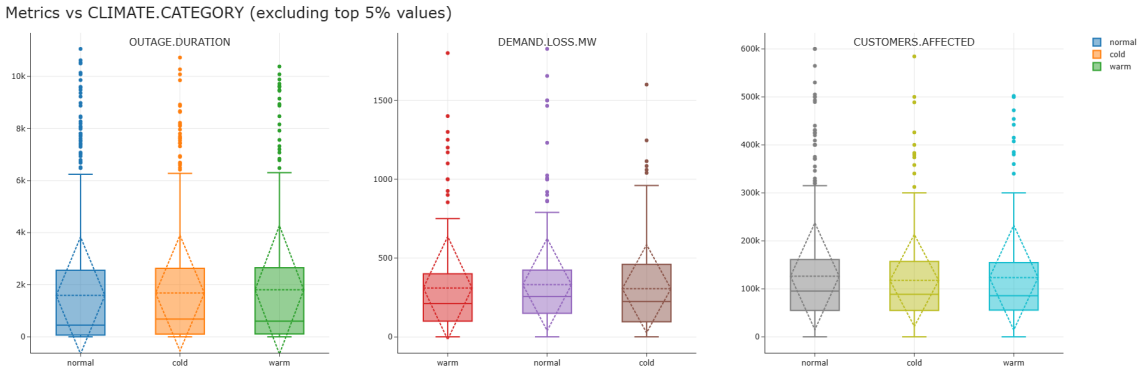


We can make the following notes about this graph:

1. The East North Central, although having the largest mean and median outage duration, does not have a significantly greater demand lost or customers affected.
2. The Northeast region seems to have significant variation in both the outage duration and the amount of customers affected, but relatively standard variance (as compared to other columns) for the demand lost.
3. Considering we removed the top 5% of values, we find that for outage duration, demand lost, and customers affected, a majority of the values tend to be on the lower end of the range of values in the column. This implies that outliers may become an issue if I choose to do regression-based predictions of these values in later parts of the

project.

Climate category

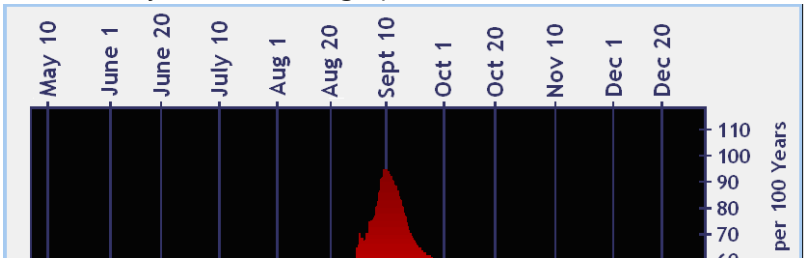


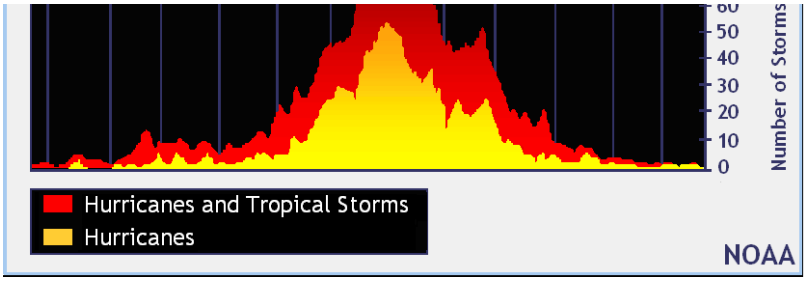
We can see that there seems to be no significant difference between any of the impact metrics that we have defined and the category of climate (i.e. whether the year is warm, cold, or normal). This implies that there is no relationship with the severity of an outage and the temperature of that year. This is unexpected - I expected years with higher or lower than average temperatures to have greater outages, possibly due to increased severe weather events.

Month

We can make the following notes about this graph:

1. At first glance, other than September (which we will talk about in the next point), there do not seem to be any obvious differences month by month for each of the impact metrics - outage duration shows the most variation by month, but common groupings (ex. seasonal) do not reveal any clear patterns.
2. However, here seems to be a slightly larger mean outage duration and demand lost in September. This makes sense, considering that September is the peak of hurricane season, as showcased by the reference graph from NOAA.



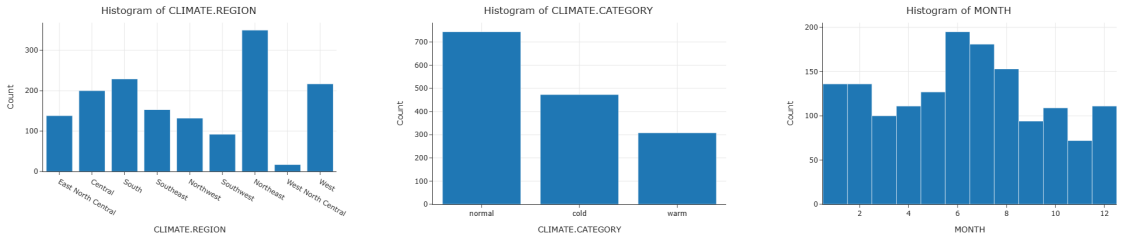


Univariate Analysis

Outage metrics

The graphs above are the univariate analyses of the outage metrics that we had defined before, including the outage duration, customers affected, and demand loss by Megawatts. These graphs confirm what we theorized earlier in our bivariate analysis - that all of these columns are significantly right-skewed. This can cause problems in regression analysis if we were to try to predict these metrics.

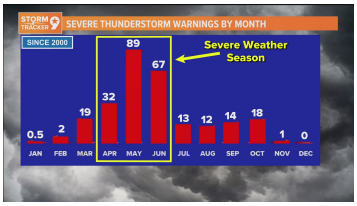
Climate features (from bivariate analysis)

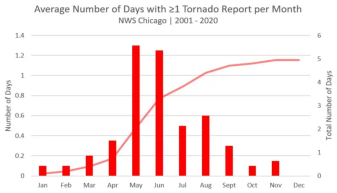


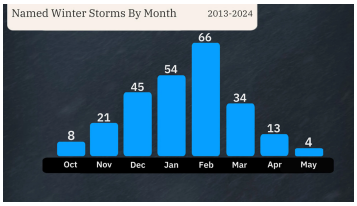
These graphs show the univariate analyses of the features we discussed in the bivariate analysis above. We can make the following observations:

1. The Northeast region by far faces the largest amount of power outages, with over 300 of them, followed by the South and the West. Furthermore, the West North Central and Southwest regions face the least amount of power outages. This could imply that there are climate/regional characteristics for these regions that cause them to have more/less outages.
2. The most power outages occur in normal years, then cold years, and finally in warm years. However, unlike the climate region, we cannot be quick to assume that regional/climate characteristics cause this. It is more likely that the "normal" climate category encompasses a much larger portion of the years studied because the average year **is** normal.
3. The summer months seem to have the largest proportion of power outages, followed closely by the late winter months. This might be due to thunderstorms and tornadoes in

the summer months and winter storms in the late winter. This can be verified by looking at the following reference graphs (News West 9, Fox 32, The Weather Channel).







Interesting Aggregates

Following the univariate and bivariate analysis, I was interested in aggregating the cause category, climate region, and the mean/count of outages that occurred. This would give me insight into which combination of the two features would have the highest mean outage duration and highest number of outages. Not only could this allow me to define more specific and interesting hypothesis tests later on, but also gives insight into what causes high-impact outages in each region.

To do this, I defined the function `pivot_and_heatmap`, which upon taking in a dataframe, values for the index, columns, and values, and the aggregation function, created a pivot table and then a heatmap of the aggregation. Once again, as with the univariate and bivariate analysis, I saved these images in a folder. However, I display them after the code.

In [410…

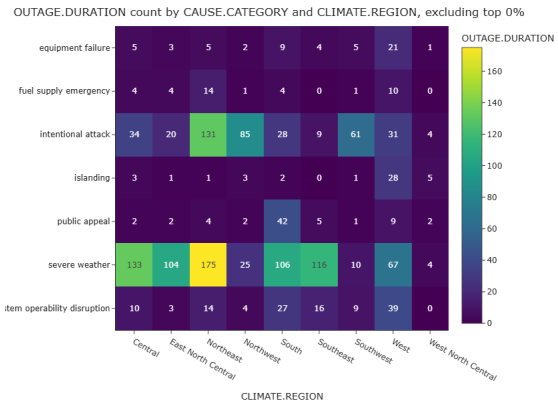
```
# Create pivot table and heatmap from the data given the indices, columns, values, and aggregation function
def pivot_and_heatmap(df, index_col, columns_col, values_col, aggfunc='mean', exclude_top_pct=0.10): # function to create pivot table and heatmap
    threshold = df[values_col].quantile(1 - exclude_top_pct) # calculate threshold for top X%
    filtered = df[df[values_col] <= threshold] # filter data to exclude top X%
    pivot_table = filtered.pivot_table(index=index_col, columns=columns_col, values=values_col, aggfunc=aggfunc) # create pivot table
    pivot_table.fillna(0, inplace=True) # fill NaN values with 0
    fig = px.imshow( # create heatmap using plotly express
        pivot_table,
        labels=dict(x=columns_col, y=index_col, color=values_col),
        x=pivot_table.columns,
        y=pivot_table.index,
        text_auto=True,
        aspect='auto',
        color_continuous_scale='Viridis'
    )
    fig.update_layout( # update layout of figure
        title=f'{values_col} {aggfunc} by {index_col} and {columns_col}, excluding top {exclude_top_pct*100:.0f}%',
        xaxis_title=columns_col,
        yaxis_title=index_col,
        height=600,
        width=900
    )
```

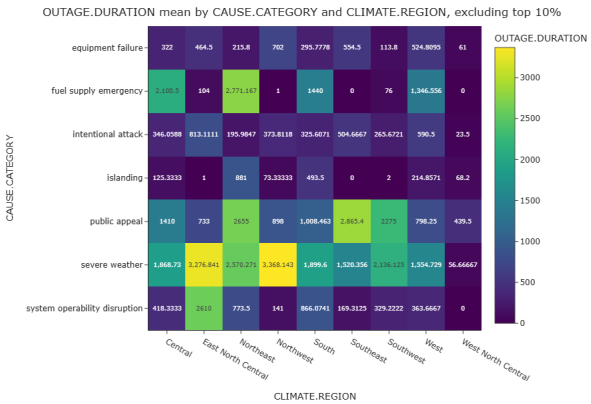
```
    return pivot_table, fig # return the heatmap figure

pivot_table_mean, fig_mean = pivot_and_heatmap(data, index_col='CAUSE.CATEGORY',
columns_col='CLIMATE.REGION', values_col='OUTAGE.DURATION', aggfunc='mean',
exclude_top_pct=0.10)
pivot_table_count, fig_count = pivot_and_heatmap(data, index_col='CAUSE.CATEGORY',
columns_col='CLIMATE.REGION', values_col='OUTAGE.DURATION', aggfunc='count',
exclude_top_pct=0)

save_figure( # plot and save heatmap of mean outage duration by cause category and
climate region
    fig_mean,
    'heatmap_mean_outage_duration',
    save_folder='heatmap_analysis'
)
convert_to_html( # convert the heatmap figure to HTML for the report
    fig_mean,
    'heatmap_mean_outage_duration'
)

save_figure( # plot and save heatmap of count of outages by cause category and
climate region
    fig_count,
    'heatmap_count_outage_duration',
    save_folder='heatmap_analysis'
)
convert_to_html( # convert the heatmap figure to HTML for the report
    fig_count,
    'heatmap_count_outage_duration'
)
```

```
Saved figure to heatmap_analysis\heatmap_mean_outage_duration.png
Saved figure to heatmap_analysis\heatmap_count_outage_duration.png
```

In [ ]:
```
# Print pivot tables as markdown for the report
print(pivot_table_mean.to_html(index=False))
print(pivot_table_count.to_html(index=False))
```

I was able to get the following graphs from my aggregate analysis:





We can gain some interesting insights from this.

1. The most outages seem to happen in the Northeast and in severe weather, with Northeastern severe weather outages being the most common power outage aggregate

by far. This can imply that the Northeastern region faces significantly harsher climate conditions, as our bivariate analysis before also revealed its uniqueness in the severity of its climate impact metrics. However, it also has the highest rate of intentional attacks by a significant margin, implying that there might be infrastructure issues in the Northeast causing power outages.
2. The highest mean outage duration seems to be for severe weather events in the Northwest and East North Central region, implying that those regions have a more difficult time restoring power after severe weather events. However, we also note that severe weather as a whole has some of the highest outage durations on a per-region basis, implying that each region might be uniquely ill-equipped to deal with severe weather outages.

Overall, we are able to get a variety of interesting insights through the graphs and many possible hypotheses that we can explore. It is important to note that although I extrapolate a lot of possibilities from the graphs drawn, **none of them are confirmed yet**.

Step 3: Assessment of Missingness

NMAR Analysis

I think that the `DEMAND.LOSS.MW` is likely an NMAR variable, which means the missingness of it depends on itself. There are a few reasons this might be:

1. The outage occurs over a very large (and therefore hard to calculate) area or where peak demand usage is highly variable, then MW lost might not be recorded. In this case, you find that high (or hard to calculate) values of `DEMAND.LOSS.MW` might be missing.
2. The outage occurs in a certain utility company's jurisdiction. If that company has no historical measurement infrastructure or poor monitoring, then the `DEMAND.LOSS.MW` value might not be viable to calculate.

In order to verify that `DEMAND.LOSS.MW` NMAR or explaining the missingness as MAR, we can obtain a couple of pieces of additional data:

1. The amount of area that an outage covers.
2. Whether the outage occurred in a rural/urban area - urban areas are more likely to have variable peak demands.
3. The utility company in charge of the electrical grid where the major power outage occurred.

Missingness Dependency

For this missingness analysis, I chose to focus on the `OUTAGE.DURATION` column - I was interested in analyzing whether this column's missing values were MAR (missing at random)

and therefore dependent on any other "feature columns". Specifically, in my analysis I tried to determine if `OUTAGE.DURATION` was MAR or not MAR when compared against `CLIMATE.REGION` and `POSTAL.CODE`. To perform my analysis most efficiently, I chose to define a variety of helper functions:

1. `get_necessary_columns` retrieves relevant columns from the data to make calling on the data within functions and debugging easier.
2. `shuffle_missing` is the shuffling function used to perform the permutation test between null and not-null data.
3. `calculate_categorical_proportion` determined the proportion of each category in a column to determine the TVD between null and not-null data.
4. `calculate_tvd` is the function to calculate the TVD between two distributions of categorical data.
5. `calculate_tvd_between_nulls` is a function that filters, calculates categorical proportions, and then calculates the TVD between null and not null values.
6. `perform_missingness_permutation_test` is a performs the permutation test by combining all the previous functions and `n_repetitions`
7. `calculate_p_value` calculates the p-value of a distribution and an observed statistic.
8. `plot_permutation_test_distribution` uses a histogram to plot and visualize the permutation test.

In [412…
```
data['DEMAND.LOSS.MW'].isnull().sum()
```

Out[412…
```
np.int64(901)
```

In [413…
```
missing_counts = data.isnull().sum() # count missing values per column in a series
missing_counts = missing_counts[missing_counts > 0].sort_values(ascending=False) #
filter to only columns with missing values and sort descending

def get_necessary_columns(data, col_list): # function to get necessary columns
from data
    return data[col_list]

def shuffle_missing(data, shuffle_column): # function to shuffle a column in the
data
    data_shuffled = data.assign(shuffled =
np.random.permutation(data[shuffle_column])) # create new column with shuffled
values
    return data_shuffled

def calculate_categorical_proportions(data, categorical_column, target_column): #
function to calculate proportions of categorical variable
    groupby_filtered = data.groupby(categorical_column)
[target_column].size().fillna(0) # group by categorical column and count target
column
    groupby_filtered = groupby_filtered / groupby_filtered.sum() # normalize to
```

```
get proportions
    return groupby_filtered

def calculate_tvd(proportions1, proportions2): # function to calculate total
variation distance between two distributions
    return (proportions1 - proportions2).abs().sum() / 2

def calculate_tvd_between_nulls(data, categorical_column, target_column): #
function to calculate TVD between null and non-null distributions
    data_not_null = data[data[target_column].notnull()] # filter data to only non-
null target column values
    groupby_not_null = calculate_categorical_proportions(data_not_null,
categorical_column, target_column)
    data_null = data[data[target_column].isnull()] # filter data to only null
target column values
    groupby_null = calculate_categorical_proportions(data_null,
categorical_column, target_column)
    tvd = calculate_tvd(groupby_not_null, groupby_null)
    return tvd

def perform_missingness_permutation_test(data, categorical_column, target_column,
n_repetitions=1000): # function to perform permutation test for missingness
    filtered_data = get_necessary_columns(data, [categorical_column,
target_column]) # filter data to necessary columns
    observed_value = calculate_tvd_between_nulls(filtered_data,
categorical_column, target_column) # calculate observed TVD
    tvds = [] # list to store TVDs from permutations

    for _ in range(n_repetitions):  # perform permutations
        shuffled_data = shuffle_missing(filtered_data, target_column) # shuffle
target column to break association
        tvd = calculate_tvd_between_nulls(shuffled_data, categorical_column,
'shuffled') # calculate TVD for shuffled data
        tvds.append(tvd) # store TVD from this permutation

    return observed_value, tvds # return observed TVD and list of TVDs from
permutations

def calculate_p_value(observed_statistic, permuted_vals): # function to calculate
p-value from permutation test
    p_value = np.mean(np.array(permuted_vals) >= observed_statistic) # calculate
p-value as proportion of permuted values >= observed statistic
    return p_value

def plot_permutation_test_distribution(permuted_vals, observed_statistic,
categorical_column_name, target_column_name, p_value=None, title=None): # function
to plot permutation test distribution
    if title is None:
        title = (
            f'Empirical Distribution of the TVD of {categorical_column_name}
Proportion Differences<br>'
            f'Between Null and Not Null Values of {target_column_name}'
        ) # title for the plot
    fig = px.histogram(pd.DataFrame(permuted_vals), x=0, nbins=50,
histnorm='probability',
                       title=title, width=800, height=500,) # create histogram of
```

```
permuted TVDs
    fig.add_vline(x=observed_statistic, line_color='red', line_width=1, opacity=1)
# add vertical line for observed statistic
    fig.add_annotation(text=f'<span style="color:red">Observed TVD =
{round(observed_statistic, 2)}</span><br><span style="color:red">P-value =
{round(p_value, 3)}</span>',
                       x=1.2 * observed_statistic, showarrow=False, y=0.05) # add
annotation for observed statistic and p-value
    return fig
```

MAR Dependence on Climate Region

The following code performs the test checking if the missing values in outage duration depend on the climate region. Specifically, the pair of hypotheses are:

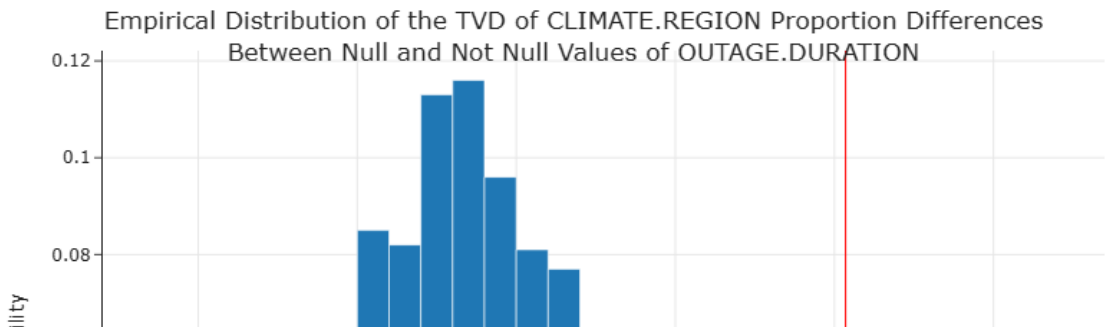
1. **Null**: The missingness of OUTAGE.DURATION is not MAR dependent on CLIMATE.REGION.
2. **Alternative**: The missingness of OUTAGE.DURATION is MAR dependent on CLIMATE.REGION.
3. **Test statistic**: The TVD of CLIMATE.REGION values for missing and non-missing values.

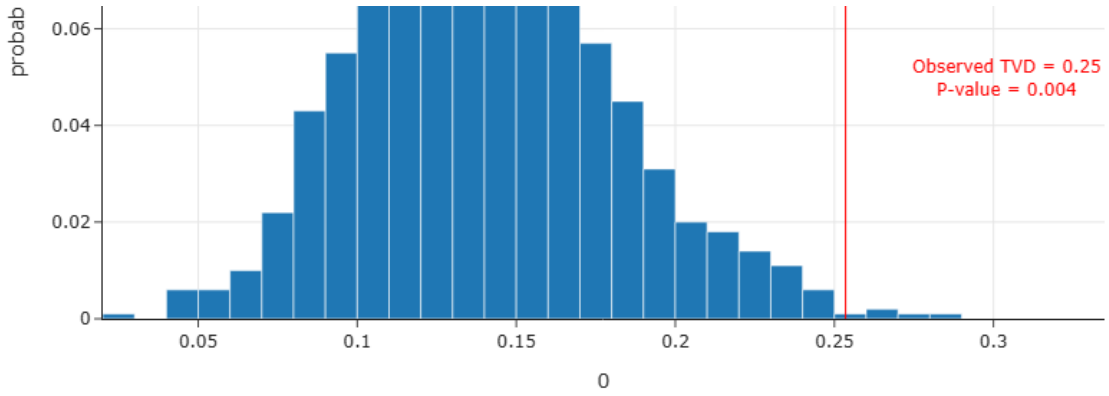
In [414…

```
observed_statistic, permuted_vals = perform_missingness_permutation_test(data,
'CLIMATE.REGION', 'OUTAGE.DURATION', n_repetitions=1000) # perform permutation
test
p_value = calculate_p_value(observed_statistic, permuted_vals) # calculate p-value
print(f'P-value: {p_value}')
fig = plot_permutation_test_distribution(permuted_vals, observed_statistic,
'CLIMATE.REGION', 'OUTAGE.DURATION', p_value=p_value) # plot permutation test
distribution
save_figure(fig, 'missingness_permutation_test_climate_region_outage_duration',
save_folder='hyp_perm_test_analysis') # save the figure
convert_to_html(fig,
'missingness_permutation_test_climate_region_outage_duration') # convert to HTML
for report
```

```
P-value: 0.004
Saved figure to hyp_perm_test_analysis\missingness_permutation_test_climate_region_o
utage_duration.png
```

If we run the code above, we find that the p-value of this permutation test is less than 0.05, the significance threshold used in data science. This means that we **reject the null** and can therefore state that the missingness of OUTAGE.DURATION is MAR dependent on CLIMATE.REGION. Here is the plot from the permutation test:





MAR Dependence on State

The following code performs the test checking if the missing values in outage duration depend on the state (or the postal code column). Specifically, the pair of hypotheses are:

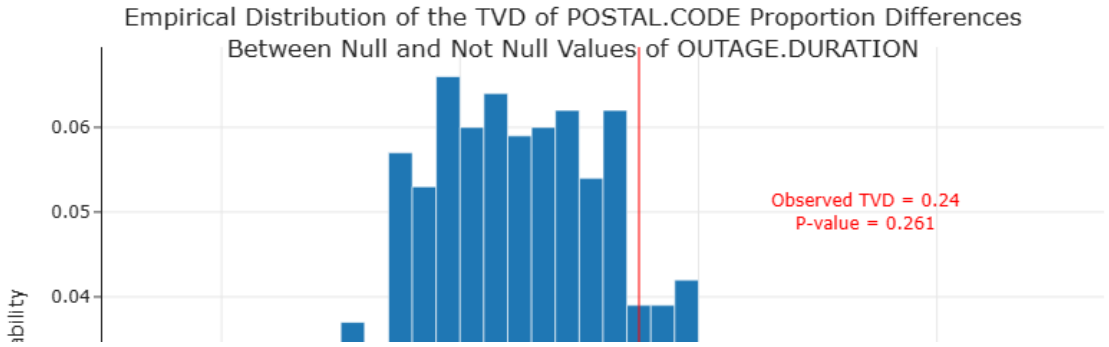
1. **Null**: The missingness of OUTAGE.DURATION is not MAR dependent on POSTAL.CODE.
2. **Alternative**: The missingness of OUTAGE.DURATION is MAR dependent on POSTAL.CODE.
3. **Test statistic**: The TVD of MONTH values for missing and non-missing values.

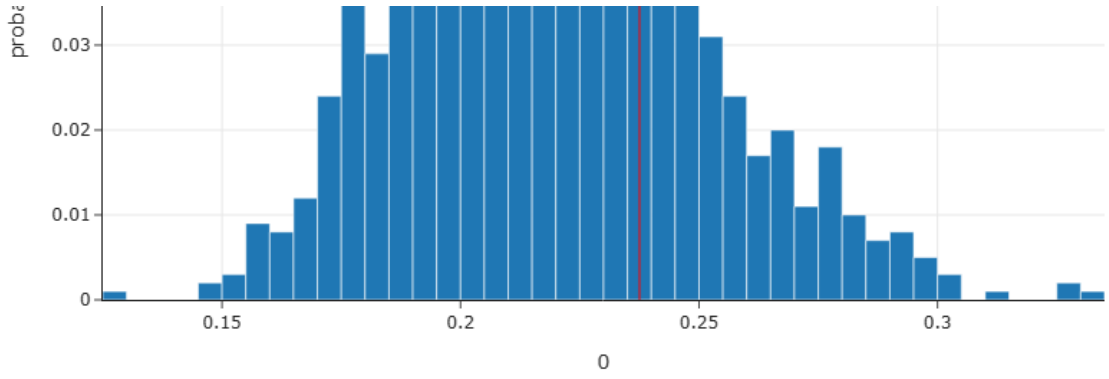
In [415…

```
observed_statistic, permuted_vals = perform_missingness_permutation_test(data,
'POSTAL.CODE', 'OUTAGE.DURATION', n_repetitions=1000) # perform permutation test
p_value = calculate_p_value(observed_statistic, permuted_vals) # calculate p-value
print(f'P-value: {p_value}')
fig = plot_permutation_test_distribution(permuted_vals, observed_statistic,
'POSTAL.CODE', 'OUTAGE.DURATION', p_value=p_value) # plot permutation test
distribution
save_figure(fig, 'missingness_permutation_test_postal_code_outage_duration',
save_folder='hyp_perm_test_analysis') # save the figure
convert_to_html(fig, 'missingness_permutation_test_postal_code_outage_duration') #
convert to HTML for report
```

```
P-value: 0.261
Saved figure to hyp_perm_test_analysis\missingness_permutation_test_postal_code_outa
ge_duration.png
```

If we run the code above, we find that the p-value of this permutation test is greater than 0.05, the significance threshold used in data science. This means that we **fail to reject the null** and can therefore state that the missingness of MONTH is not MAR dependent on POSTAL.CODE. Here is the plot from the permutation test:





Step 4: Hypothesis Testing

For the hypothesis test, I decided to test a pattern I observed in the bivariate analysis - the relationship with more Northern regions and increased OUTAGE.DURATION values. Specifically, my hypotheses and test statistic were:

1. **Null**: The outage duration for regions classified as "North" is not greater than the outage duration for all regions (i.e. is consistent with random sampling). In other words, Northern regions are **not special**.
2. **Alternative**: The outage duration for regions classified as "North" is greater than the outage duration for all regions.
3. **Test statistic**: The mean of the outage duration for "North" regions.

This was important for the question I am trying to answer which relates regional/climate information with the severity of outages because it establishes a clear relationship between geographically north (and usually colder) regions and a greater outage durations.

1. My null and alternate hypotheses directly targets the question I have about regional/climate impact by packaging a variety of climate regions into what is essentially a binary variable, allowing for a low granularity but important test regaridng whether geographic location matters. My null hypotheses represents the **baseline assumption** that any observed difference in outages across large scale regional/climate categories is random variation.
2. My test statistic, specifically just calculating the mean outage duration for randomly sampled/observed "North" regions, is also justifiable based on the way the p-value is calculated. Since we calculate the p-value by doing `np.mean(np.array(sampled_vals) >= observed_statistic)` , if the p-value is less than the significance threshold (0.05), then that means we reject the null hypothesis, and vice versa (i.e. we don't need to calculate differences due to the nature of our p-value calculation).
3. The p-value threshold is the standard threshold for hypothesis testing. There is not a compelling reason to change this value.

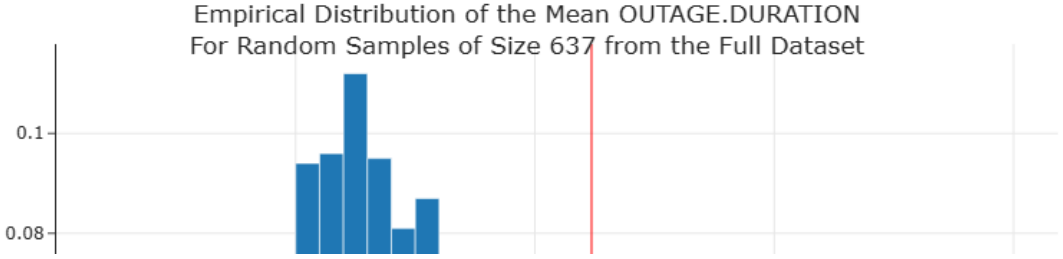
In [416…

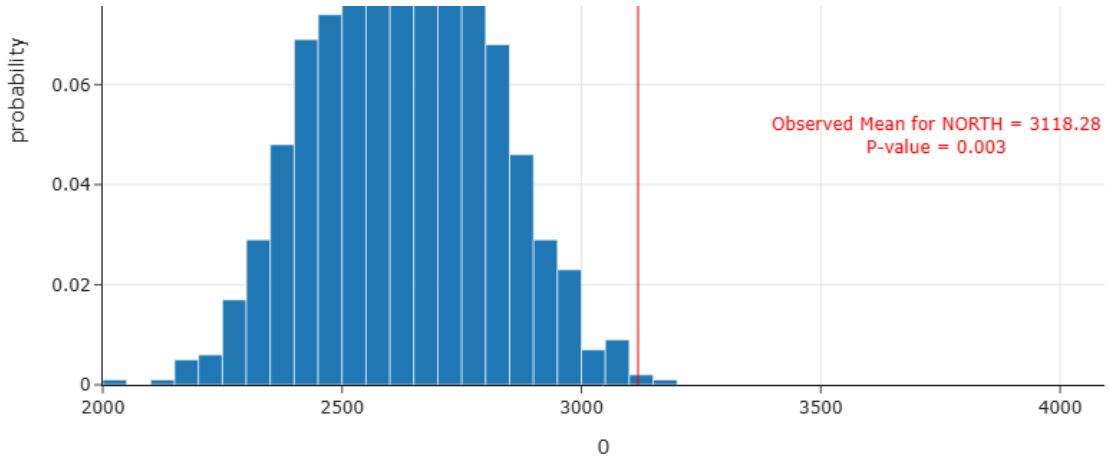
```
# Define a mapping from detailed climate regions to 'North' and 'Not North'
```

```
climate_region_map = {'North': ['Northeast', 'East North Central', 'Northwest',
'West North Central'], 'Not North': ['South', 'West', 'Central', 'Southeast',
'Southwest']}
# Reverse the map to easily map each region to its bin
bin_map = {region: group for group, regions in climate_region_map.items() for
region in regions}
# Map the CLIMATE.REGION to the binned version
data['CLIMATE.REGION.BINNED'] = data['CLIMATE.REGION'].map(bin_map)
# Determine the counts in each bin for sampling later on
binned_vals = data['CLIMATE.REGION.BINNED'].value_counts()
# Determine the observed mean for the 'North' bin
observed_mean = data.loc[data['CLIMATE.REGION.BINNED'] == 'North',
'OUTAGE.DURATION'].mean()
# Perform permutation test
n_repetitions = 1000
permuted_means = []
for _ in range(n_repetitions):
    sample_mean = data['OUTAGE.DURATION'].sample(n=binned_vals['North']).mean()
    permuted_means.append(sample_mean)
# Calculate p-value
p_value = calculate_p_value(observed_mean, permuted_means)
print(f'P-value: {p_value}')
# Plot the hypothesis test distribution
title = (
    f'Empirical Distribution of the Mean OUTAGE.DURATION<br>'
    f'For Random Samples of Size {binned_vals["North"]} from the Full Dataset'
)
fig = px.histogram(pd.DataFrame(permuted_means), x=0, nbins=50,
histnorm='probability',
                   title=title, width=800, height=500,) # create histogram of
permuted means
fig.add_vline(x=observed_mean, line_color='red', line_width=1, opacity=1)
fig.add_annotation(text=f'<span style="color:red">Observed Mean for NORTH =
{round(observed_mean, 2)}</span><br><span style="color:red">P-value =
{round(p_value, 3)}</span>',
                   x=1.2 * observed_mean, showarrow=False, y=0.05)
save_figure(fig, 'hypothesis_test_climate_region_binned_outage_duration',
save_folder='hyp_perm_test_analysis') # save the figure
convert_to_html(fig, 'hypothesis_test_climate_region_binned_outage_duration') #
convert to HTML for report
```

```
P-value: 0.003
Saved figure to hyp_perm_test_analysis\hypothesis_test_climate_region_binned_outage_
duration.png
```

We find that the p-value of this hypothesis test is indeed less than 0.05, causing us to **reject the null hypothesis** and demonstrating that Northern regions likely have a higher outage duration compared to the general data. Here is the plot for the hypothesis test:





Step 5: Framing a Prediction Problem

The prediction problem that I am trying to solve is to predict the cause of a major power outage based on information about the climate, environment, and region of the outage.

1. This is a **multiclass classification problem**, where each outage belongs to exactly one cause class.
2. I am choosing **accuracy** as the classification metric over precision, recall, or F1-score:
    A. While severe weather outages are more common, precision and recall are primarily meaningful in binary classification or when some classes are more critical. Here, all outage causes are equally important, and we want to correctly identify the cause regardless of class.
    B. In multiclass classification, each instance belongs to only one class. Prioritizing precision for one class reduces false positives for that class but increases false negatives for it, which in turn increases false positives for other classes. Similarly, prioritizing recall for one class reduces its false negatives but increases its false positives, shifting errors to other classes. Therefore, precision and recall metrics are difficult to interpret in this zero-sum scenario.
    C. F1-score addresses the precision-recall tradeoff but still requires averaging across classes and is less intuitive. Since all classes are equally important, accuracy is simpler and easier to interpret, and therefore is what I am using.
    D. Overall, accuracy provides a clear, interpretable, and holistic measure of model performance for this multiclass task.
3. The **response variable** I am using is the cause category, as it directly aligns with my research problem on investigating outage information (such as cause and severity) using regional/climate information. Furthermore, since causes of outages are discrete and mutually exclusive, it lends itself to a classification problem.
    A. In terms of real world impact, predicting the cause allows stakeholders to take preventative actions depending on the type of outage, or make policy decisions regarding mitigative measures.
4. My baseline model uses the following columns for prediction `['POSTAL.CODE',`

`'NERC.REGION', 'MONTH', 'YEAR', 'CLIMATE.REGION', 'ANOMALY.LEVEL', 'CLIMATE.CATEGORY', 'HURRICANE.NAMES', 'PCT_LAND', 'PCT_WATER_TOT', 'PCT_WATER_INLAND']` (and my final model uses a subset of these). If we are trying to predict the cause of an outage, all of these are pieces of infomration we would know beforehand - i.e. there is no feature on here that is directly impacted by the fact that there IS an outage.

Step 6: Baseline Model

This baseline model is a **decision tree** uses a series of climate relevant models using the following columns:

1. `POSTAL.CODE` : nominal data that provides us with the state.
2. `NERC.REGION` : nominal data that provides us with the North American Electric Reliability Corporation regions.
3. `MONTH` : quantiative data that indicates the month that the outage occurred.
4. `YEAR` : quantiative data that indicates the year that the outage occurred.
5. `CLIMATE.REGION` : nominal data that provides us with one of nine climatically consistent regions in the continental US.
6. `ANOMALY.LEVEL` : quantitative data that provides us with the oceanic El Nino/La Nina index referring to the cold and warm episodes by season.
7. `CLIMATE.CATEGORY` : ordinal data that categorizes climate episodes corresponding to years (cold, normal, and warm have an inherent ordering to them).
8. `HURRICANE.NAMES` : nominal data that indicates the hurricane that was occurring at the time of the outage (if there was one)
9. `PCT_LAND` : quantitative data incidating the percentage of the land area in the state.
10. `PCT_WATER_TOT` : quantitative data inciduating the percentage of water area.
11. `PCT_WATER_INLAND` : quantitative data indicating the percentage of inland water area in the state.

Overall, I had 4 pieces of nominal data, 1 pieces of ordinal data, and 6 pieces of ordinal data. I performed the necessary encodings by using a one hot encoder (`OneHotEncoder(drop='first', handle_unknown='ignore')`) for the nominal values and an ordinal encoder (`OrdinalEncoder(categories=[<categories>], dtype=int)`) for the ordinal column.

In [386…

```
# Step 1 - define functions for imputation of null data. This is to ensure that
the model can be fit without errors due to missing data.

def prob_impute_categorical(series): # function to impute missing categorical
values based on observed distribution
    series = series.copy() # create a copy of the series to avoid modifying the
```

```
original
    num_null = series.isnull().sum() # count number of null values
    fill_values = np.random.choice(series.dropna(), size=num_null, replace=True) #
sample from non-null values
    series[series.isnull()] = fill_values # fill null values with sampled values
    return series

def prob_impute_numerical(series): # function to impute missing numerical values
with mean
    return series.fillna(series.mean())
```

In [387…

```
# Step 2 - X data preparation. Get columns to be used in the model and impute any
missing values for ordinal columns
climate_relevant_columns = ['POSTAL.CODE', 'NERC.REGION', 'MONTH', 'YEAR',
                    'CLIMATE.REGION', 'ANOMALY.LEVEL', 'CLIMATE.CATEGORY',
                    'HURRICANE.NAMES', 'PCT_LAND', 'PCT_WATER_TOT',
'PCT_WATER_INLAND'] # columns to use for modeling
model_data = data[climate_relevant_columns] # subset data to only model columns

# Impute missing values
for col in model_data.columns:
    if model_data[col].dtype == 'object': # if column is categorical
        model_data[col] = prob_impute_categorical(model_data[col]) # impute using
categorical imputation
    else: # if column is numerical
        model_data[col] = prob_impute_numerical(model_data[col]) # impute using
numerical imputation

# Step 3 - y data preparation - cause of the outage
y = data['CAUSE.CATEGORY']

# Step 4 - create preprocessing and modeling pipeline
col_trans = make_column_transformer( # create column transformer for preprocessing
    (OneHotEncoder(drop='first', handle_unknown='ignore'), ['POSTAL.CODE',
'NERC.REGION', 'CLIMATE.REGION', 'HURRICANE.NAMES']), # one-hot encode nominal
categorical columns
    (OrdinalEncoder(categories=[['cold', 'normal', 'warm']], dtype=int),
['CLIMATE.CATEGORY']), # ordinal encode the CLIMATE.CATEGORY column because it has
an inherent order to it
    remainder='passthrough', # pass through any remaining columns without
transformation
)

pipeline = make_pipeline( # create pipeline with preprocessing and decision tree
classifier model
    col_trans,
    DecisionTreeClassifier()
)

# Step 5 - split data into training and testing sets
X = model_data
y = y # target variable defined above
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # split data into training and testing sets
```

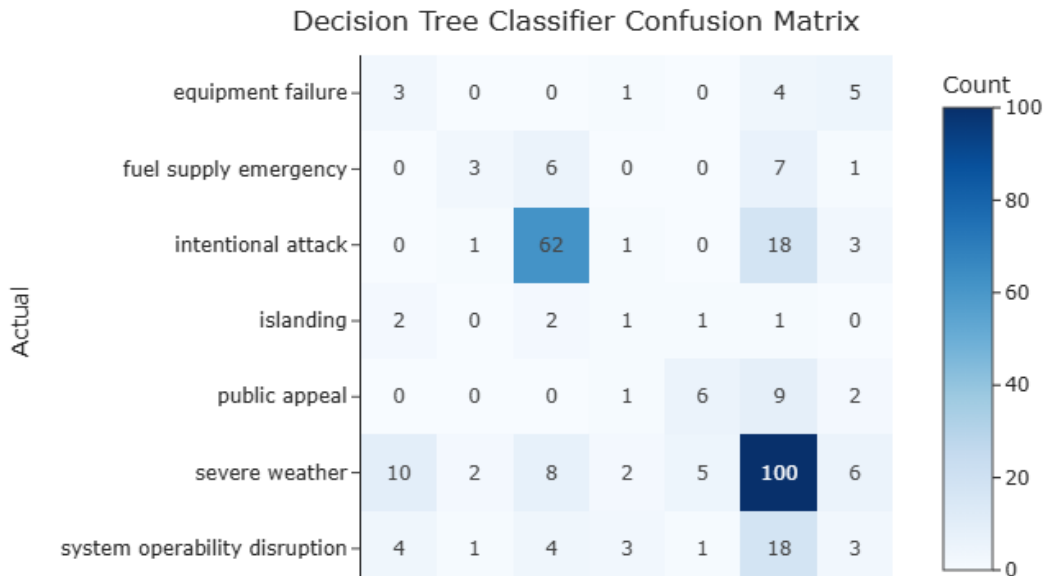
In [389…
```
# Step 6 - fit the pipeline and evaluate the model
pipeline.fit(X_train, y_train) # fit the pipeline on training data

accuracy = (pipeline.predict(X_test) == y_test).mean() # compute accuracy on test
data of the pipeline, which includes preprocessing and model prediction
print(f'Accuracy: {accuracy}') # print accuracy
cm = confusion_matrix(y_test, pipeline.predict(X_test), labels=pipeline.classes_)
# compute confusion matrix
fig = px.imshow(
    cm,
    text_auto=True,
    labels=dict(x="Predicted", y="Actual", color="Count"),
    x=pipeline.classes_,
    y=pipeline.classes_,
    color_continuous_scale='Blues',
    width=700,
    height=500
) # create heatmap of confusion matrix
fig.update_layout(title="Decision Tree Classifier Confusion Matrix") # update
layout of figure to have a title
save_figure(fig, 'decision_tree_confusion_matrix', save_folder='model_evaluation')
# save the confusion matrix figure
convert_to_html(fig, 'decision_tree_confusion_matrix') # convert confusion matrix
figure to HTML for report
```

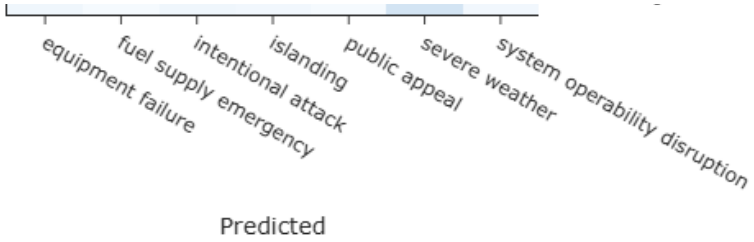
```
Accuracy: 0.5798045602605864
Saved figure to model_evaluation\decision_tree_confusion_matrix.png
```

In [390…
```
# Calculate class proportions for CAUSE.CATEGORY
class_counts = data['CAUSE.CATEGORY'].value_counts(normalize=True)
# Expected accuracy for random selection proportional to class distribution
expected_random_accuracy = sum(class_counts ** 2)

print(f"Expected random accuracy: {expected_random_accuracy:.3f}")
```

```
Expected random accuracy: 0.334
```

Overall, I was able to set my model accuracy at ~57.9% (this willl vary based on each run), with the following confusion matrix.





Evaluating my current model, I do not believe that it is good. This is because of a few reasons:

1. There are some columns that can cause multicollinearity. Specifically, since you can predict `PCT_WATER_TOT = 1 - PCT_LAND`, the mdoel is multicollinear.
2. I can make current features more useful by transforming them.
   A. Currently I impute values into `HURRICANE.NAMES`, but since most outages aren't related to hurricanes, this is a misrepresentative column when making predictions. It could be made better by a `Binarizer`.
   B. I also simply use the quantitative `MONTH` values. However, it might be more useful to encode these as seasons after a mapping, to generalize the model better and make it a nominal categorical variable. One issue with quantitative months is that numerically far months in winter (ex. December, January), might actually want to be adjacent to each other.
3. Finally, I believe my model is not good because currently it sits at around a 57% accuracy, which although is better than the expected random accuracy of 33.4%, seems like it can be significantly improved to ensure that policy/mitigation actions ar taken with more confidence.

Step 7: Final Model

The final model improves on the baseline model. Specifically, I performed the following for the readability of the model:

1. I applied a `StandardScaler` on my quantitative columns. Although this does not specifically help with model accuracy or changes how decisions are made, it does make the model more interpretable.
2. I removed columns that are highly correlated to reduce model multicollinarity. The column that this was relevant to was `PCT_WATER_TOT` and `PCT_LAND`, as you could calculate `PCT_WATER_TOT = 1 - PCT_LAND`.

In terms of new features, I added the following:

1. I **binarized** the `HURRICANE.NAMES` column to be `HURRICANE_PRESENT`. This is because a) imputing `NaN` values with randomly selected hurricane names is misrepresentative of the climate conditions leading up to the power outage and b) the presence or absence of a hurricane is a relevant piece of data that can be used to

predict the cause of an outage, especially considering its impact on climate data at that specific time point.
2. I **mapped** the quantitative column of `MONTH` to a categorical column of `SEASON`. This is because a) encoding the `SEASON` as opposed to `MONTH` allows for better generalization to make our predictions (i.e. whether a month is June, July, or August might be less relevant than the fact that these are summer months and therefore have specific climate characteristics) and b) numerically far months (like December and January) are able to be categorically adjacent to one another and therefore more representative of climate patterns.

From there, I tested a second model (**Random Forest**) and used **GridSearchCV** with both models in order to determine the best-performing model and set of hyperparameters.

For the **Decision Tree** model, I made sure to optimize for the following hyperparameters:

1. `max_depth`: this controls the maximum depth of the tree, and I hope to prevent overfitting by limiting it and finding the optimal depth for maximum accuracy.
2. `min_samples_split`: this controls the minimum number of samples required to split an internal node, and I hope to prevent overfitting by requiring a higher number of samples to make splits.

For the **Random Forest** model, I made sure to optimize for the same hyperparameters as the decision tree plus one more:

1. `n_estimators`: this controls how MANY decision trees I am using to make the general decision. If I have too few, it can lead to unstable predictions, but too many can lead to unnecessary computation and no gain.

`modify_data_for_modeling` function

In the interest of making data preprocessing simpler in Step 8, where I perform a fairness analysis, I made a function for the initial data modification that needs to happen for the model.

In [391…

```
def modify_data_for_modeling(data):
    # Step 1 - re-select relevant columns for climate modeling
    climate_relevant_columns = ['YEAR', 'MONTH', 'POSTAL.CODE', 'NERC.REGION',
                        'CLIMATE.REGION', 'ANOMALY.LEVEL', 'CLIMATE.CATEGORY',
                        'HURRICANE.NAMES', 'PCT_LAND', 'PCT_WATER_TOT',
'PCT_WATER_INLAND']
    model_data_final = data[climate_relevant_columns] # subset data to only model
columns

    # Step 2 - analyze numerical columns and decide which ones to remove based on
correlation
    numerical_cols =
model_data_final.select_dtypes(include=['number']).columns.tolist() # select all
```

```
numerical columns
    correlation_matrix = model_data_final[numerical_cols].corr().abs() # compute
absolute correlation matrix
    # It is shown that PCT_LAND has a high correlation with PCT_WATER_TOT (100 -
PCT_LAND), so we will remove that column
    model_data_final = model_data_final.drop(columns=['PCT_WATER_TOT'])

    # # Step 3 - binarize the HURRICANE.NAMES column
    # The name of a hurricane does not provide useful information, but whether or
not there was a hurricane does
    model_data_final['HURRICANE_PRESENT'] =
model_data_final['HURRICANE.NAMES'].notnull().astype(int) # create binary feature
for hurricane presence
    model_data_final = model_data_final.drop(columns=['HURRICANE.NAMES']) # drop
the original

    # Step 4 - convert MONTH to categorical type using seasons
    def month_to_season(month):
        if month in [12, 1, 2]:
            return 'Winter'
        elif month in [3, 4, 5]:
            return 'Spring'
        elif month in [6, 7, 8]:
            return 'Summer'
        elif month in [9, 10, 11]:
            return 'Fall'
    model_data_final['SEASON'] = model_data_final['MONTH'].apply(month_to_season)
# map month to season
    model_data_final = model_data_final.drop(columns=['MONTH']) # drop the
original MONTH column

    # Step 5 - impute missing values using previously defined functions
    for col in model_data_final.columns:
        if col in ['POSTAL.CODE', 'NERC.REGION', 'CLIMATE.REGION',
'CLIMATE.CATEGORY', 'SEASON']: # if column is categorical
            model_data_final[col] = prob_impute_categorical(model_data_final[col])
# impute using categorical imputation
        elif col in ['YEAR', 'ANOMALY.LEVEL', 'PCT_LAND', 'PCT_WATER_INLAND']: #
if column is numerical
            model_data_final[col] = prob_impute_numerical(model_data_final[col]) #
impute using numerical imputation

    return model_data_final
```

In [392…

```
# Step 1 - re-prepare X and y data for modeling with modified function
X = modify_data_for_modeling(data) # features
y = data['CAUSE.CATEGORY'] # target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) # split
data into training and testing sets
```

In [393…

```
# Step 2 - create preprocessing and modeling pipelines for decision tree and
random forest classifiers
col_trans = make_column_transformer( # create column transformer for preprocessing
    (OneHotEncoder(drop='first', handle_unknown='ignore'), ['POSTAL.CODE',
'NERC.REGION', 'CLIMATE.REGION', 'HURRICANE_PRESENT', 'SEASON']), # one-hot encode
```

```
nominal categorical columns
    (OrdinalEncoder(categories=[['cold', 'normal', 'warm']], dtype=int),
['CLIMATE.CATEGORY']), # ordinal encode the CLIMATE.CATEGORY column because it has
an inherent order to it
    (StandardScaler(), ['YEAR', 'ANOMALY.LEVEL', 'PCT_LAND', 'PCT_WATER_INLAND']),
# standard scale numerical columns
    remainder='passthrough', # pass through any remaining columns without
transformation
)

pipeline_dt = make_pipeline( # create pipeline with preprocessing and decision
tree classifier model
    col_trans,
    DecisionTreeClassifier()
)

pipeline_rf = make_pipeline( # create pipeline with preprocessing and random
forest classifier model
    col_trans,
    RandomForestClassifier()
)

# Step 3 - perform grid search cross-validation to find best hyperparameters for
both models
param_grid_dt = {
    'decisiontreeclassifier__max_depth': list(range(5, 45)) + [None],
    'decisiontreeclassifier__min_samples_split': list(range(2, 10)),
}

param_grid_rf = {
    'randomforestclassifier__n_estimators': list(range(10, 110, 10)),
    'randomforestclassifier__max_depth': list(range(5, 45)) + [None],
    'randomforestclassifier__min_samples_split': list(range(2, 10)),
}

dt_grid_search = GridSearchCV(
    pipeline_dt,
    param_grid=param_grid_dt,
    cv=5,
    scoring='accuracy',
)

rf_grid_search = GridSearchCV(
    pipeline_rf,
    param_grid=param_grid_rf,
    cv=5,
    scoring='accuracy',
)
```

`fit_and_evaluate_model` function

In order to ensure that we don't have to repeat code for the Random Forest and Decision Trees grid searches, I set up this function to evaluate and plot the confusion matrices of both models.

In [394…
```
def fit_and_evaluate_model(grid_search, model_name, X_train, y_train, X_test, y_test): # function to fit and evaluate model
    grid_search.fit(X_train, y_train) # fit the grid search on training data
    best_model = grid_search.best_estimator_ # get the best model from grid search
    accuracy = (best_model.predict(X_test) == y_test).mean() # calculate accuracy on test data
    print("Model: ", model_name) # print model name
    print(f'Best Parameters: {grid_search.best_params_}') # print best hyperparameters
    print(f'Accuracy: {accuracy}') # print accuracy
    fig = px.imshow(
        cm,
        text_auto=True,
        labels=dict(x="Predicted", y="Actual", color="Count"),
        x=pipeline.classes_,
        y=pipeline.classes_,
        color_continuous_scale='Blues',
        width=700,
        height=500
    ) # create heatmap of confusion matrix
    fig.update_layout(title=f"{model_name} Confusion Matrix") # update layout of figure to have a title
    save_figure(fig, f'{model_name.lower().replace(" ", "_")}_confusion_matrix', save_folder='model_evaluation') # save the confusion matrix figure
    convert_to_html(fig, f'{model_name.lower().replace(" ", "_")}_confusion_matrix') # convert confusion matrix figure to HTML for report
    return best_model
```

In [395…
```
best_dt_model = fit_and_evaluate_model(dt_grid_search, "Decision Tree Grid Search", X_train, y_train, X_test, y_test) # fit and evaluate decision tree model
best_rf_model = fit_and_evaluate_model(rf_grid_search, "Random Forest Grid Search", X_train, y_train, X_test, y_test) # fit and evaluate random forest model
```

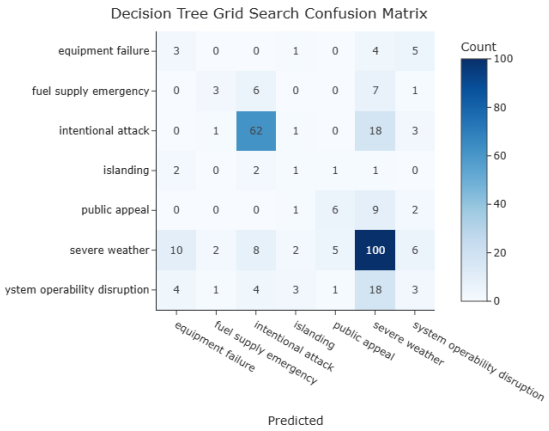
```
Model:  Decision Tree Grid Search
Best Parameters: {'decisiontreeclassifier__max_depth': 8, 'decisiontreeclassifier__min_samples_split': 5}
Accuracy: 0.6482084690553745
Saved figure to model_evaluation\decision_tree_grid_search_confusion_matrix.png
Model:  Random Forest Grid Search
Best Parameters: {'randomforestclassifier__max_depth': 36, 'randomforestclassifier__min_samples_split': 6, 'randomforestclassifier__n_estimators': 90}
Accuracy: 0.6938110749185668
Saved figure to model_evaluation\random_forest_grid_search_confusion_matrix.png
```

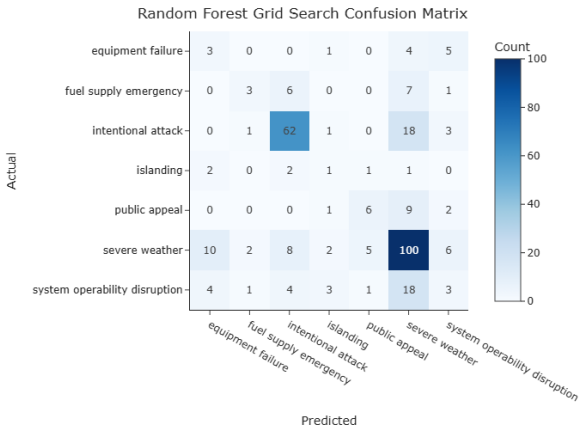
Through Grid Search and testing two modeling techniques, I got a decision tree model at ~64.8% accuracy, with the optimal hyperparameters being a max depth of 8 and a minimum samples split of 5. I also got a random forest model with a ~69.4% accuracy, with the optimal

hyperparameters being a max depth of 36, a minimum samples split of 6, and an estimators count of 90.

What is interesting about these models is that in the decision tree, we had a relatively smaller max depth than in the random forest model, but the random forest model used 90 trees, relatively close to the maximum we were testing for. This implies that the random forest model used more in-depth trees to vote on a final decision, even if the individual trees might have overfit on their bootstrapped and aggregated data.

The confusion matrices for both models are available here:

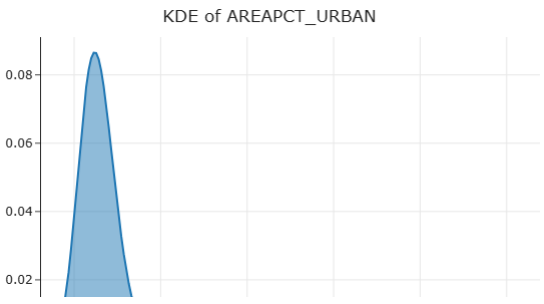


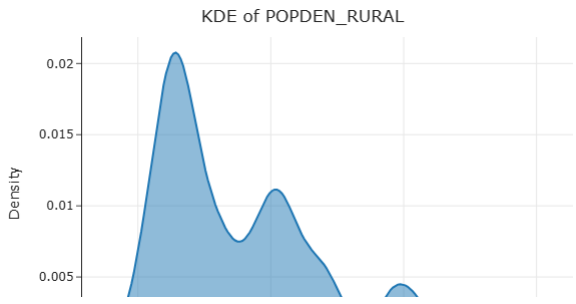


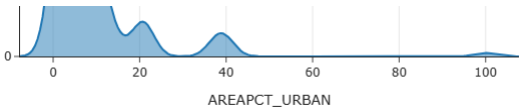
Step 8: Fairness Analysis

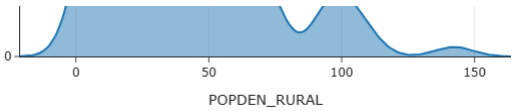
For the fairness analysis, I chose to measure whether my model would perform better or worse for individuals who were in areas where a greater median percentage of their population was rural compared to areas where a greater median percentage of the population is urban. I theorize that **urban populations** will have a lower accuracy due to the diverse causes of outages/incidents correlated with the large population, whereas rural populations have fewer, more predictable causes.

The reason I used `POPDEN_RURAL` instead of `AREAPCT_URBAN` to binarize is because cities are usually very dense, leading to the majority of land in the U.S. to be rural. Therefore there is less variation (and therefore a worse split) in `AREAPCT_URBAN` as compared to `POPDEN_RURAL`, which while also might tend to be right skewed, does tend itself to an increased variance and therefore a more valid binarization. This can be verified using the univariate plots of the two columns:









In order to binarize our `POPDEN_RURAL` column into "rural" and "urban", I split them with the median of that column as the binarization threshold. Then, I formed the test:

1. **Null**: our model is fair. The accuracy for "rural" and "urban" population is roughly the same, and any differences are due to random chance.
2. **Alternative**: our model is unfair. The accuracy for "urban" populations is lower than the accuracy for "rural" populations.
3. **Test statistic**: the signed difference in accuracy between "urban" and "rural" populations.

The significance level was kept at 0.05, the industry standard for these tests, as there is no compelling reason to make the test more or less selective.

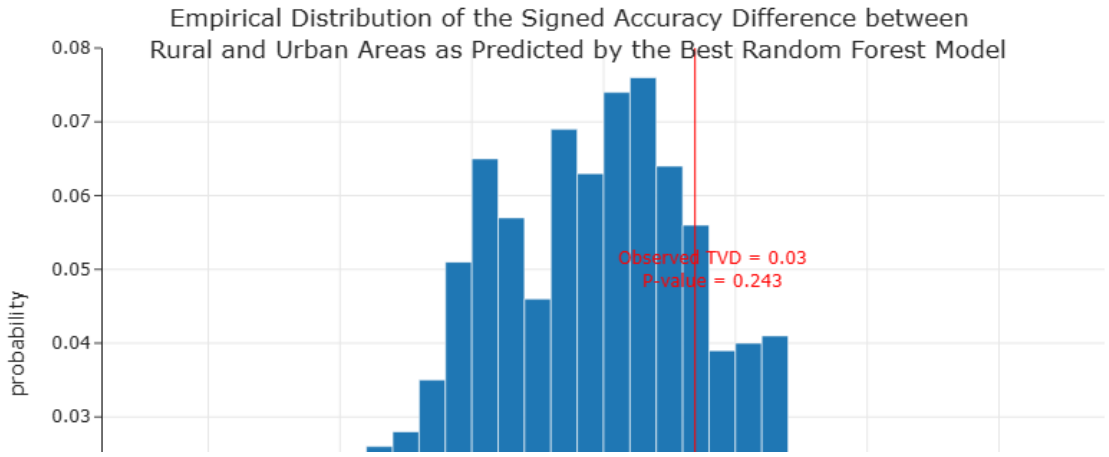
In [397…

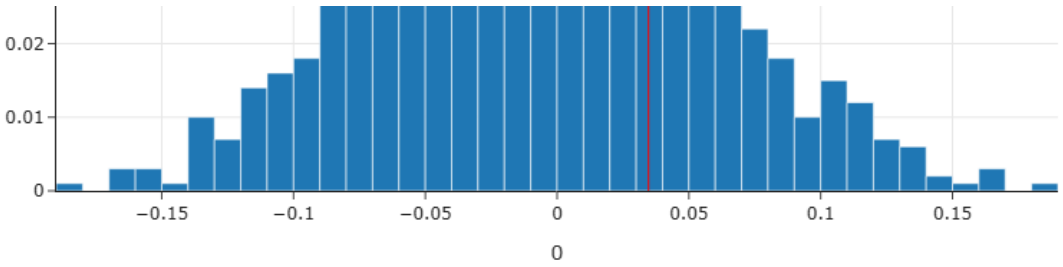
```
# Create a copy of the data to prevent modifying the original
data_binarized = data.copy()

# Step 1 - define the statistic to binarize
binarization_threshold = data['POPDEN_RURAL'].median()
data_binarized['POPDEN_RURAL'] = (data_binarized['POPDEN_RURAL'] >
binarization_threshold).astype(int)

# Step 2 - define function to calculate recall TVD between rural and urban areas
def calculate_signed_acc_difference_rural_urban(best_model, data):
    # Step 2.1 - define X and y for modeling
    data_rural = data[data['POPDEN_RURAL'] == 1]
    X_rural = modify_data_for_modeling(data_rural)
    y_rural = data_rural['CAUSE.CATEGORY']

    data_urban = data[data['POPDEN_RURAL'] == 0]
    X_urban = modify_data_for_modeling(data_urban)
    y_urban = data_urban['CAUSE.CATEGORY']

    # Step 2.2 - split data into training and testing sets for rural and urban
    X_rural_train, X_rural_test, y_rural_train, y_rural_test =
train_test_split(X_rural, y_rural, test_size=0.2)
    X_urban_train, X_urban_test, y_urban_train, y_urban_test =
train_test_split(X_urban, y_urban, test_size=0.2)

    # Step 2.3 - create and fit the model pipelines for rural and urban
    best_model.fit(X_rural_train, y_rural_train)
    predictions_rural = best_model.predict(X_rural_test)
    accuracy_rural = (predictions_rural == y_rural_test).mean()

    best_model.fit(X_urban_train, y_urban_train)
    predictions_urban = best_model.predict(X_urban_test)
    accuracy_urban = (predictions_urban == y_urban_test).mean()

    # Step 2.4 - calculate and return the TVD between rural and urban recall
distributions
```

```
    return accuracy_rural - accuracy_urban

# Step 3 - define the observed statistic
observed_statistic = calculate_signed_acc_difference_rural_urban(best_rf_model,
data_binarized)

# Step 4 - perform permutation test
n_repetitions = 1000
permuted_tvds = []
for _ in range(n_repetitions):
    # Step 4.1 - Shuffle the POPDEN_RURAL column to break association
    data_shuffled = data_binarized.copy()
    data_shuffled['POPDEN_RURAL'] =
np.random.permutation(data_shuffled['POPDEN_RURAL'])

    # Step 4.2 - Calculate the recall TVD for the shuffled data
    tvd = calculate_signed_acc_difference_rural_urban(best_dt_model,
data_shuffled)
    permuted_tvds.append(tvd)

# Step 5 - graph the permutation test distribution
p_value = calculate_p_value(observed_statistic, permuted_tvds)
print(f'P-value: {p_value}')
fig = plot_permutation_test_distribution(permuted_tvds, observed_statistic,
'POPDEN_RURAL', 'Signed Accuracy Difference', p_value=p_value, title='Empirical
Distribution of the Signed Accuracy Difference between <br> Rural and Urban Areas
as Predicted by the Best Random Forest Model')
save_figure(fig,
'fairness_permutation_test_popden_rural_signed_accuracy_difference',
save_folder='fairness_analysis') # save the figure
convert_to_html(fig,
'fairness_permutation_test_popden_rural_signed_accuracy_difference') # convert to
HTML for report
```

```
P-value: 0.243
Saved figure to fairness_analysis\fairness_permutation_test_popden_rural_signed_accu
racy_difference.png
```

Through this, we were able to determine that the resulting p-value was 0.26, implying that I **fail to reject the null hypothesis** and therefore cannot reliably state that the signed accuracy difference between rural and urban areas is significant enough to prove urban areas have an overall lower accuracy. Here is the figure from the permutation test:




0.02
0.01
0
−0.15
−0.1
−0.05
0
0.05
0.1
0.15
0