# Alaa and Tristan's Epic Analysis of Frequency of Power Outages in Different Regions

**Name(s)**: Alaa Fadhl-Allah and Tristan Romanov

**Website Link**: [https://romanov360.github.io/alaa-tristan-epic-energy-eda/ (https://romanov360.github.io/alaa-tristan-epic-energy-eda/)](https://romanov360.github.io/alaa-tristan-epic-energy-eda/)

## Code

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

import plotly.express as px
import matplotlib.pyplot as plt

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

# Introduction

In this project, we studied the power outages dataset. Each row of the dataset is a U.S. power outage, each column describes something about that outage. The dataset includes geographical (where did the outages occur) and temporal (when did the outages occur + how long did they last) data, and we connected the two in our analysis.

The dataset has 1534 rows and 55 columns.

We were interested in the 'YEAR','MONTH','CLIMATE.REGION','OUTAGE.DURATION','OUTAGE.START.TIME', 'OUTAGE.START.DATE','CAUSE.CATEGORY', and 'CAUSE.CATEGORY.DETAIL' columns. Their descriptions follow.

### Cleaning and EDA

We write all in functions to keep processing modular.

This is what the Excel (.xlsx) file initially looks like post download from Purdue:

| | A | B | C | D | E | F | G | H | I | CLI |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **Major power outage events in the continental U.S.** | | | | | | | | | |
| 2 | **Time period: January 2000 - July 2016** | | | | | | | | | |
| 3 | **Regions affected: Outages reported in this data file affected a single U.S. state at the time of occurrence** | | | | | | | | | |
| 4 | | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | **variables** | OBS | YEAR | MONTH | U.S._STATE | POSTAL.CODE | NERC.REGION | CLIMATE.REGION | ANOMALY.LEVEL | CLI |
| 7 | *Units* | | | | | | | | *numeric* | |
| 8 | | 1 | 2011 | 7 | Minnesota | MN | MRO | East North Central | -0.3 | |
| 9 | | 2 | 2014 | 5 | Minnesota | MN | MRO | East North Central | -0.1 | |
| 10 | | 3 | 2010 | 10 | Minnesota | MN | MRO | East North Central | -1.5 | |
| 11 | | 4 | 2012 | 6 | Minnesota | MN | MRO | East North Central | -0.1 | |
| 12 | | 5 | 2015 | 7 | Minnesota | MN | MRO | East North Central | 1.2 | |
| 13 | | 6 | 2010 | 11 | Minnesota | MN | MRO | East North Central | -1.4 | |
| 14 | | 7 | 2010 | 7 | Minnesota | MN | MRO | East North Central | -0.9 | |
| 15 | | 8 | 2005 | 6 | Minnesota | MN | MRO | East North Central | 0.2 | |

This is not in tidy data format (one row per observation, one column per feature) since there is a header and columns have descriptors below the names. We removed the first five rows in Excel and saved it as a .csv file. The following function reads the file, removes the descriptors, resets the index, and returns a tidy DataFrame.

In [2]:
```
def get_data():
    """returns data as dataframe"""
    return pd.read_csv('outage_chopped.csv').iloc[1:].reset_index().drop(columns=['OBS','index'])
```

We look at the data and its shape to grow familiarity.

In [3]:
```
df=get_data()
```

In [4]:
```
df.shape
```

Out[4]: (1534, 55)

In [5]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    display(df.head())
```

| | YEAR | MONTH | U.S._STATE | POSTAL.CODE | NERC.REGION | CLIMATE.REGION | ANOMALY.LEVEL | CLIMATE.CATEGORY | OUTAGE.START.DATE | OU |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011.0 | 7.0 | Minnesota | MN | MRO | East North Central | -0.3 | normal | Friday, July 1, 2011 | |
| 1 | 2014.0 | 5.0 | Minnesota | MN | MRO | East North Central | -0.1 | normal | Sunday, May 11, 2014 | |
| 2 | 2010.0 | 10.0 | Minnesota | MN | MRO | East North Central | -1.5 | cold | Tuesday, October 26, 2010 | |
| 3 | 2012.0 | 6.0 | Minnesota | MN | MRO | East North Central | -0.1 | normal | Tuesday, June 19, 2012 | |
| 4 | 2015.0 | 7.0 | Minnesota | MN | MRO | East North Central | 1.2 | warm | Saturday, July 18, 2015 | |

We look at the data types alongside a couple of rows of the data to check that the data types are what they ought to be. The first row contains the data types, the following three rows are a few observations.

In [6]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more options can be specified also
    display(pd.concat([pd.DataFrame(df.dtypes).T,df.iloc[:3]]))
```

| | YEAR | MONTH | U.S._STATE | POSTAL.CODE | NERC.REGION | CLIMATE.REGION | ANOMALY.LEVEL | CLIMATE.CATEGORY | OUTAGE.START.DATE | OU |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | float64 | float64 | object | object | object | object | object | object | object | |
| 0 | 2011.0 | 7.0 | Minnesota | MN | MRO | East North Central | -0.3 | normal | Friday, July 1, 2011 | |
| 1 | 2014.0 | 5.0 | Minnesota | MN | MRO | East North Central | -0.1 | normal | Sunday, May 11, 2014 | |
| 2 | 2010.0 | 10.0 | Minnesota | MN | MRO | East North Central | -1.5 | cold | Tuesday, October 26, 2010 | |

We notice that the data types are wrong. Most of the columns are of type object, which is for strings, but the data are supposed to be numeric or dates/timedeltas.

In [7]:
```
pd.concat(
    [
        pd.DataFrame(df.dtypes).reset_index().drop(columns='index').rename(columns={0:'type'}),
        pd.DataFrame(df.shape[1]*['counter']).rename(columns={0:'count'})
    ],
    axis=1
).groupby('type').count()
```

Out[7]:

| | count |
|---|---|
| **type** | |
| **float64** | 8 |
| **object** | 47 |

For example, per the data dictionary, ANOMALY.LEVEL represents the El Niño/La Niña (ONI) index, a numerical climatological measure, but the column is of type object.

Moreover, we notice that some of the float datatypes provided in the original dataset are not supposed to be floats but rather ints. See that the first and second column contain month and year as integer-valued floats. Thus we cannot ignore the eight float64-typed columns as we fix the columnar data types.

We try to fix the data types but run into two issues.

#### Issue One

We notice columns like OUTAGE.DURATION are integer-valued objects, so we try to cast them to int, but we get this error:

In [8]:
```
try:
    df['OUTAGE.DURATION'].astype(int)
except ValueError as error:
    print(error)
```

```
cannot convert float NaN to integer
```

It appears that int Series cannot contain NaNs.

We try to fix this by ignoring the errors.

In [9]:
```
df['OUTAGE.DURATION'].astype(int,errors='ignore').head(3)
```

Out[9]:
```
0    3060
1       1
2    3000
Name: OUTAGE.DURATION, dtype: object
```

However, we see that the cast failed: the column is still of type object.

We know that the Int32 type is helpful for cases like this, where we want to have an int-valued column that contains NaNs. We try direct conversion but get this error.

In [10]:
```
try:
    df['OUTAGE.DURATION'].astype('Int64') #see here we are using Int64 instead of int
except TypeError as error:
    print(error)
```

```
object cannot be converted to an IntegerDtype
```

We solved this by intermediately casting to float. We deduced this solution by applying two facts

1. Float-like object columns are convertible to the float type without problems
2. NaN-containing float columns are convertible to the Int32 type without problems

and observing

3. Conversion from object to Int32 fails

to perform steps 1 and 2 sequentially instead of 3.

This solution is implemented in the try-except block. It tries to convert to int, and if that fails, it tries the two-step approach.

#### Issue Two

How do we know which numeric-looking columns are supposed to be floats and which are supposed to be ints? There are over a thousand rows, so we should do this programmatically instead of applying the eye filter. For columns like CUSTOMERS.AFFECTED, we presume this should be an int, as a customer count should be whole number valued, but for columns like DEMAND.LOSS.MW, which measures the difference in the potential and the actual quantity of electricity sold due to outage in megawatts. Whether this measure be integer-valued or float-valued is up to convention. Megawatts are virtually continuously valued, but looking at the table, we see all integers:

In [11]:
```
df['DEMAND.LOSS.MW'].unique()[:10]
```

Out[11]:
```
array([nan, '250', '75', '20', '0', '926', '100', '300', '1', '494'],
      dtype=object)
```

As the logic of convention is forgotten but its' artifacts remain, we should perform programmatic typecasting depending on the contents of the columns instead of the column names.

We can check whether a float or object-typed column of floats is integer-valued by the built-in float type method float.is_integer(). We deploy the integer check on the TOTAL.SALES column, which can be float or integer valued a priori.

In [12]:
```
np.all(
    df['TOTAL.SALES'].astype(float).apply(
        lambda num: num.is_integer() or np.isnan(num)
    )
)
```

Out[12]:
```
True
```

We do this for all numeric-looking columns to infer their types. If the above test returns True, it should be int (or Int32, if it has a NaN); else it should be float.

Issues **One** and **Two** resolved, we can fix the data types.

In [13]:
```
def get_data_with_correct_types():
    """returns data with correct types as dataframe"""
    # data cleaning last 11 columns

    df = get_data()
    # casting as float
    df[['PCT_WATER_INLAND', 'PCT_WATER_TOT', 'PCT_LAND', 'AREAPCT_URBAN', 'POPDEN_RURAL', 'POPDEN_UC', 'POPDEN_URBAN',
        'POPPCT_UC', 'POPPCT_URBAN', \
        'PI.UTIL.OFUSA', 'UTIL.CONTRI', 'PC.REALGSP.CHANGE', 'PC.REALGSP.REL']] = \
        df[['PCT_WATER_INLAND', 'PCT_WATER_TOT', 'PCT_LAND', 'AREAPCT_URBAN', 'POPDEN_RURAL', 'POPDEN_UC',
            'POPDEN_URBAN', 'POPPCT_UC', 'POPPCT_URBAN', \
            'PI.UTIL.OFUSA', 'UTIL.CONTRI', 'PC.REALGSP.CHANGE', 'PC.REALGSP.REL']].astype(float)
    # casting as int
    df[['POPULATION', 'TOTAL.REALGSP', 'UTIL.REALGSP', 'PC.REALGSP.USA']] = \
        df[['POPULATION', 'TOTAL.REALGSP', 'UTIL.REALGSP', 'PC.REALGSP.USA']].astype(int)
    #from front
    #casting as int
    for column in ['YEAR','MONTH','DEMAND.LOSS.MW','CUSTOMERS.AFFECTED',
        'RES.SALES','COM.SALES','IND.SALES','TOTAL.SALES',
        'RES.CUSTOMERS', 'COM.CUSTOMERS', 'IND.CUSTOMERS', 'TOTAL.CUSTOMERS',
        'PC.REALGSP.STATE', 'PC.REALGSP.USA', 'OUTAGE.DURATION']:
        try:
            df[column] = df[column].astype(int)
        except:
            df[column] = df[column].astype(float).astype('Int32')
    #casting as float
    for column in ['ANOMALY.LEVEL','RES.PRICE','COM.PRICE','IND.PRICE','TOTAL.PRICE',
                   'RES.PERCEN', 'COM.PERCEN', 'IND.PERCEN',
                   'RES.CUST.PCT', 'COM.CUST.PCT', 'IND.CUST.PCT',
                   'AREAPCT_UC']:
        df[column] = df[column].astype(float)
    return df
```

In [14]:
```
df=get_data_with_correct_types()
```

We reprint the datatypes with some rows to visually verify our effort. Now all column types appear correct.

In [15]:
```
#(not yet. we still need to do some float conversion and all of the datetime/timedelta conversion).
```

In [16]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more options can be specified also
    display(pd.concat([pd.DataFrame(df.dtypes).T,df.iloc[:3]]))
```

| | YEAR | MONTH | U.S._STATE | POSTAL.CODE | NERC.REGION | CLIMATE.REGION | ANOMALY.LEVEL | CLIMATE.CATEGORY | OUTAGE.START.DATE | OU |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | int32 | Int32 | object | object | object | object | float64 | object | object | |
| **0** | 2011 | 7 | Minnesota | MN | MRO | East North Central | -0.3 | normal | Friday, July 1, 2011 | |
| **1** | 2014 | 5 | Minnesota | MN | MRO | East North Central | -0.1 | normal | Sunday, May 11, 2014 | |
| **2** | 2010 | 10 | Minnesota | MN | MRO | East North Central | -1.5 | cold | Tuesday, October 26, 2010 | |

We are interested in the columns Year, Month, Climate Region, and Outage Duration as they are the most analyzable.

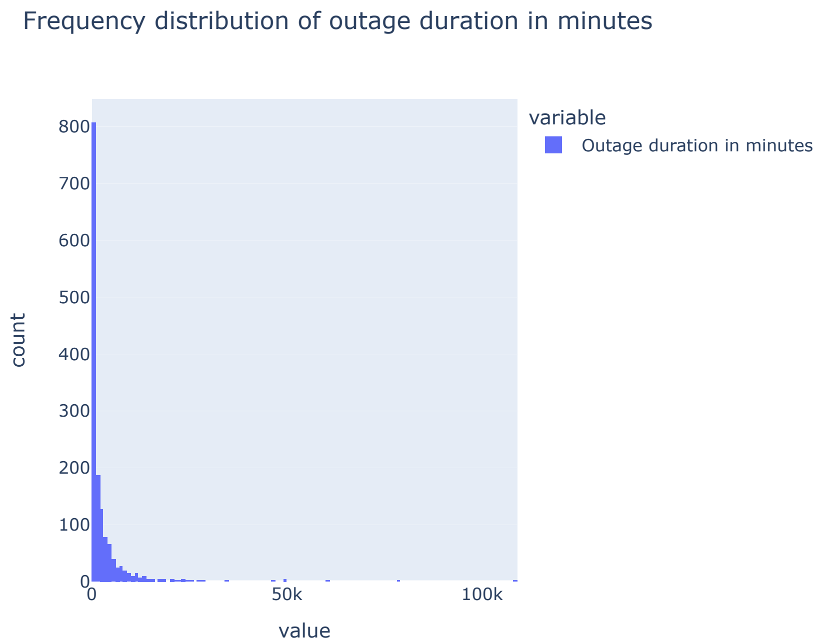
In [64]:
```
df=df[['YEAR','MONTH','CLIMATE.REGION','OUTAGE.DURATION','OUTAGE.START.TIME','OUTAGE.START.DATE','CAUSE.CATEGORY','CAUSI
```

## Univariate Analysis

We look at univariate distributions to get a sense of individual column values.

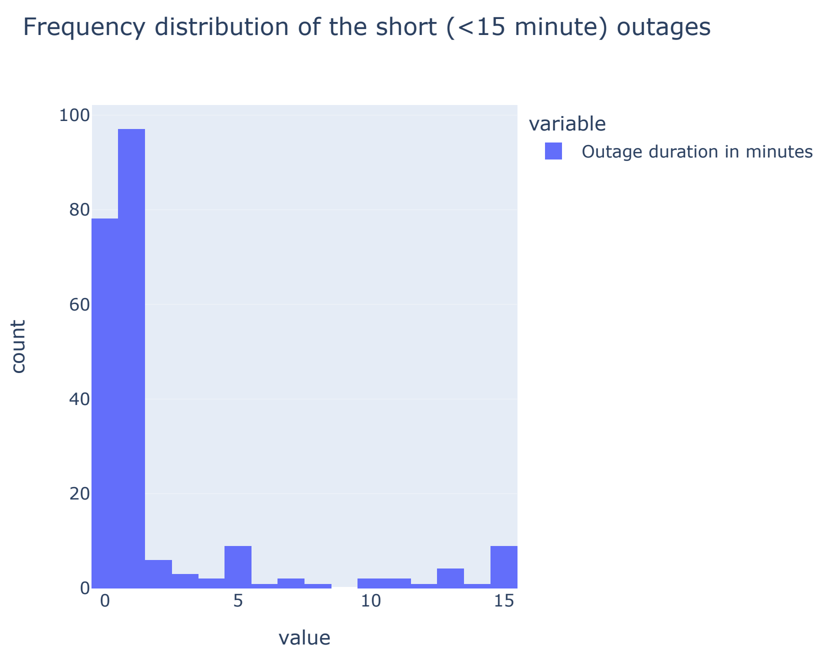
We see that most of the outages are short, but some are very long.

In [72]:
```
df[['OUTAGE.DURATION']].rename(columns={'OUTAGE.DURATION':'Outage duration in minutes'}).plot(kind='hist',title='Frequer
```



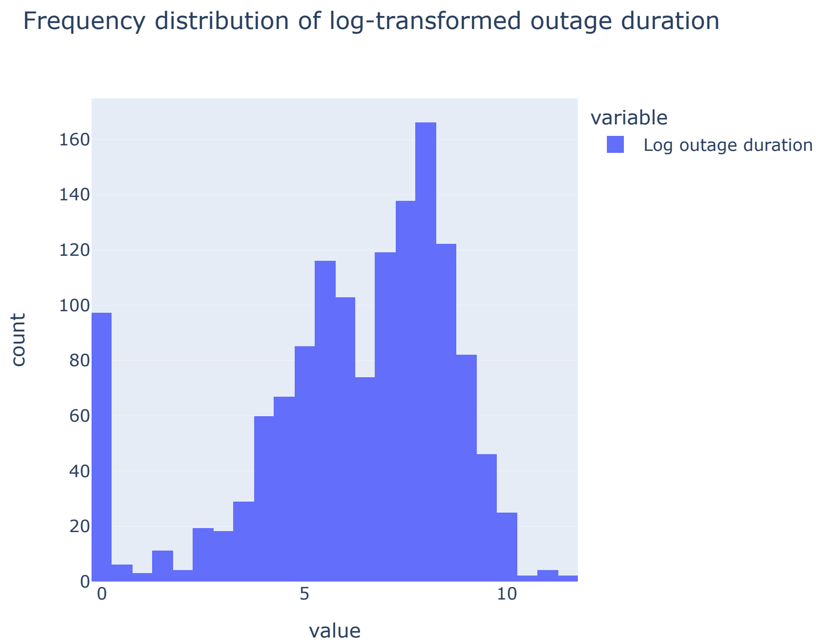

What if we look at the shorter outages only? It looks like a huge bulk of the outages last less than two minutes.

In [75]:
```
df[['OUTAGE.DURATION']][df['OUTAGE.DURATION']<=15].rename(columns={'OUTAGE.DURATION':'Outage duration in minutes'}).plot
```



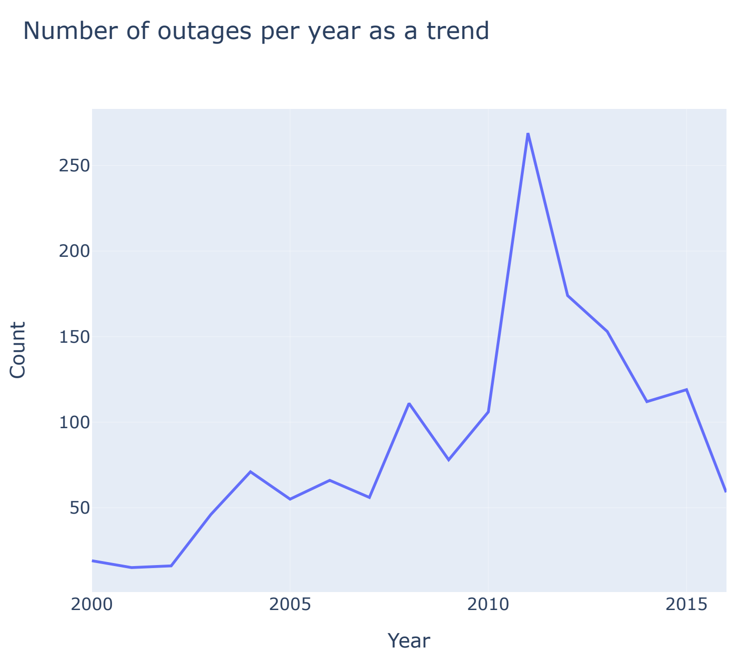

If we log-transform the outage duration, which is meant to give an idea of trend across scales of orders of magnitude, we can see a pattern. It appears to be a point mass at zero added to a bimodal bell, like the sum of two Gaussians with different means, with the lower Gaussian having a wider spread. The point mass at zero is due to the zero and one minute outages.

In [77]:
```
non_na_outage_times=df[['OUTAGE.DURATION']][~df['OUTAGE.DURATION'].isna()].reset_index()[['OUTAGE.DURATION']]
np.log(non_na_outage_times[non_na_outage_times['OUTAGE.DURATION']>0]).rename(columns={'OUTAGE.DURATION':'Log outage dura
```




We can look at how the number of outages per year changed over time. It looks like 2011-2013 have had a lot more power outages than the typical year. We may want to test this visually informed hypothesis statistically with a hypothesis test later on.

In [21]:
```
df[['YEAR','MONTH']].groupby('YEAR').count().reset_index().rename(columns={'YEAR':'Year','MONTH':'Count'}).plot(kind='l:
```

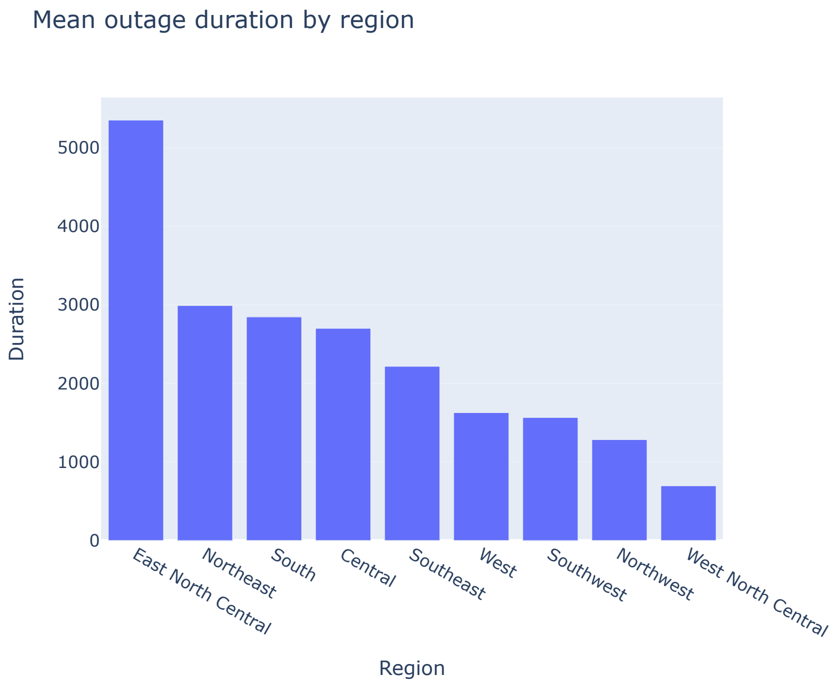


### Bivariate analysis

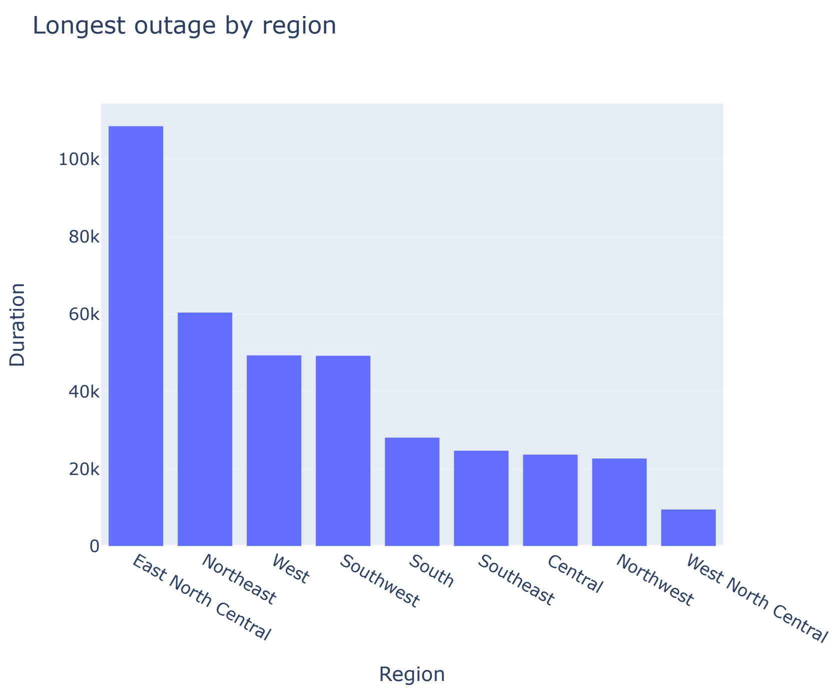
We can also perform some bivariate analysis to understand the associations between variables.

We can visualize outage duration statistics by region to identify patterns between regions and their power outages.

In [79]:
```
df[['OUTAGE.DURATION','CLIMATE.REGION']].groupby('CLIMATE.REGION').mean().reset_index().sort_values(by='OUTAGE.DURATION
```

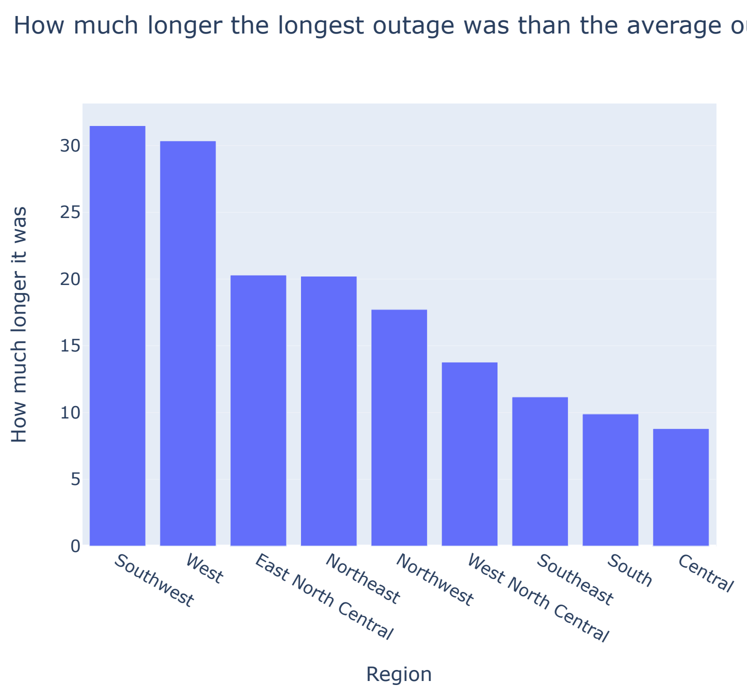



In [23]:
```
df[['OUTAGE.DURATION','CLIMATE.REGION']].groupby('CLIMATE.REGION').max().reset_index().sort_values(by='OUTAGE.DURATION'
```




By dividing these two distributions we can identify the relative duration of the longest power outage a region has had compared to their average power outage.

In [24]:
```
(df[['OUTAGE.DURATION','CLIMATE.REGION']].groupby('CLIMATE.REGION').max()/df[['OUTAGE.DURATION','CLIMATE.REGION']].grou
```




### Intererresting Aggregates

Now, we can choose different columns to grouping and pivoting, and examine the aggregate statistics of the data.

First, we combine different columns until we reach a trial that produces a valuable result.

In [25]:
```
d = df.copy()[['YEAR', 'MONTH','CLIMATE.REGION', 'OUTAGE.DURATION']]

region_year_prop = d.groupby(['CLIMATE.REGION','YEAR']).count()[['MONTH']] / d.groupby(['YEAR']).count()[['MONTH']]
region_year_prop
```

Out[25]:

| | | MONTH |
|---|---|---|
| **CLIMATE.REGION** | **YEAR** | |
| **Central** | **2000** | 0.210526 |
| | **2002** | 0.062500 |
| | **2003** | 0.130435 |
| | **2004** | 0.070423 |
| | **2005** | 0.090909 |
| **...** | **...** | ... |
| **West North Central** | **2009** | 0.038462 |
| | **2010** | 0.028302 |
| | **2011** | 0.011152 |
| | **2013** | 0.019608 |
| | **2014** | 0.008929 |

136 rows × 1 columns

In [26]:
```
year_region_prop = d.groupby(['YEAR','CLIMATE.REGION']).count()[['MONTH']] / d.groupby(['YEAR']).count()[['MONTH']]
year_region_prop
```

Out[26]:

| | | MONTH |
|---|---|---|
| **YEAR** | **CLIMATE.REGION** | |
| **2000** | **Central** | 0.210526 |
| | **Northeast** | 0.105263 |
| | **South** | 0.157895 |
| | **Southeast** | 0.315789 |
| | **Southwest** | 0.157895 |
| **...** | **...** | ... |
| **2016** | **Northwest** | 0.237288 |
| | **South** | 0.152542 |
| | **Southeast** | 0.050847 |
| | **Southwest** | 0.118644 |
| | **West** | 0.084746 |

136 rows × 1 columns

In [90]:
```
month_prop = (d.groupby('MONTH').count() / d.count())[['YEAR']]
month_prop
```

Out[90]:

| | YEAR |
|---|---|
| **MONTH** | |
| **1** | 0.088657 |
| **2** | 0.088657 |
| **3** | 0.065189 |
| **4** | 0.072360 |
| **5** | 0.082790 |
| **6** | 0.127119 |
| **7** | 0.117992 |
| **8** | 0.099739 |
| **9** | 0.061278 |
| **10** | 0.071056 |
| **11** | 0.046936 |
| **12** | 0.072360 |

In [104]:
```
print(pd.DataFrame(round(month_prop['YEAR']*100,2).apply(lambda percentage: str(percentage)+'%')).rename(columns={'YEAR
```

```
|   MONTH | Percent of power outages in this month   |
|--------:|:-----------------------------------------|
|       1 | 8.87%                                    |
|       2 | 8.87%                                    |
|       3 | 6.52%                                    |
|       4 | 7.24%                                    |
|       5 | 8.28%                                    |
|       6 | 12.71%                                   |
|       7 | 11.8%                                    |
|       8 | 9.97%                                    |
|       9 | 6.13%                                    |
|      10 | 7.11%                                    |
|      11 | 4.69%                                    |
|      12 | 7.24%                                    |
```

In [28]:
```python
year_prop = (d.groupby('YEAR').count() / d.count())[['MONTH']]
year_prop
```

Out[28]:

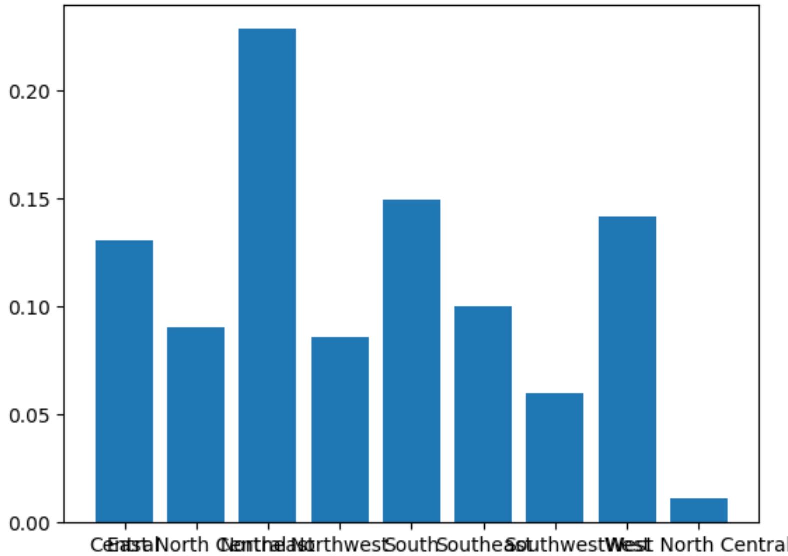
| | MONTH |
|---|---|
| **YEAR** | |
| **2000** | 0.012459 |
| **2001** | 0.009836 |
| **2002** | 0.010492 |
| **2003** | 0.030164 |
| **2004** | 0.046557 |
| **2005** | 0.036066 |
| **2006** | 0.043279 |
| **2007** | 0.036721 |
| **2008** | 0.072787 |
| **2009** | 0.051148 |
| **2010** | 0.069508 |
| **2011** | 0.176393 |
| **2012** | 0.114098 |
| **2013** | 0.100328 |
| **2014** | 0.073443 |
| **2015** | 0.078033 |
| **2016** | 0.038689 |

In [29]:
```python
region_prop = (d.groupby('CLIMATE.REGION').count() / d.count())[['YEAR']]
region_prop
```

Out[29]:

| | YEAR |
|---|---|
| **CLIMATE.REGION** | |
| **Central** | 0.130378 |
| **East North Central** | 0.089961 |
| **Northeast** | 0.228162 |
| **Northwest** | 0.086050 |
| **South** | 0.149283 |
| **Southeast** | 0.099739 |
| **Southwest** | 0.059974 |
| **West** | 0.141460 |
| **West North Central** | 0.011082 |

In [30]:
```python
region_prop_plot = plt.bar(region_prop.index, region_prop['YEAR'])
region_prop_plot;
```




In [31]:
```python
# PIVOT TABLE
region_year_prop_table = region_year_prop.pivot_table(index = 'CLIMATE.REGION', columns = 'YEAR', values = 'MONTH')
region_year_prop_table
```

Out[31]:

| YEAR | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CLIMATE.REGION** | | | | | | | | | | | | | | |
| **Central** | 0.210526 | NaN | 0.0625 | 0.130435 | 0.070423 | 0.090909 | 0.090909 | 0.089286 | 0.216216 | 0.256410 | 0.075472 | 0.148699 | 0.126437 | 0 |
| **East North Central** | NaN | NaN | 0.0625 | 0.217391 | 0.070423 | 0.181818 | 0.030303 | 0.160714 | 0.090090 | 0.115385 | 0.122642 | 0.063197 | 0.045977 | 0 |
| **Northeast** | 0.105263 | 0.133333 | 0.1875 | 0.260870 | 0.098592 | 0.109091 | 0.242424 | 0.196429 | 0.189189 | 0.064103 | 0.264151 | 0.308550 | 0.379310 | 0 |
| **Northwest** | NaN | NaN | NaN | 0.043478 | 0.042254 | NaN | 0.166667 | 0.035714 | 0.018018 | 0.012821 | 0.018868 | 0.137546 | 0.143678 | 0 |
| **South** | 0.157895 | 0.200000 | 0.1875 | 0.065217 | 0.154930 | 0.200000 | 0.151515 | 0.142857 | 0.126126 | 0.243590 | 0.169811 | 0.122677 | 0.132184 | 0 |
| **Southeast** | 0.315789 | 0.066667 | 0.1250 | 0.130435 | 0.366197 | 0.218182 | 0.090909 | 0.035714 | 0.126126 | 0.064103 | 0.066038 | 0.074349 | 0.063218 | 0 |
| **Southwest** | 0.157895 | NaN | 0.0625 | 0.043478 | 0.042254 | 0.018182 | 0.030303 | 0.017857 | 0.009009 | 0.038462 | 0.018868 | 0.063197 | 0.028736 | 0 |
| **West** | 0.052632 | 0.600000 | 0.3125 | 0.108696 | 0.140845 | 0.181818 | 0.136364 | 0.321429 | 0.207207 | 0.166667 | 0.235849 | 0.066914 | 0.080460 | 0 |
| **West North Central** | NaN | NaN | NaN | NaN | 0.014085 | NaN | 0.015152 | NaN | 0.009009 | 0.038462 | 0.028302 | 0.011152 | NaN | 0 |

In [32]:
```python
year_region_prop_table = year_region_prop.pivot_table(index = 'YEAR', columns = 'CLIMATE.REGION', values = 'MONTH')
year_region_prop_table
```

Out[32]:

| CLIMATE.REGION<br>YEAR | Central | East North Central | Northeast | Northwest | South | Southeast | Southwest | West | West North Central |
|---|---|---|---|---|---|---|---|---|---|
| **2000** | 0.210526 | NaN | 0.105263 | NaN | 0.157895 | 0.315789 | 0.157895 | 0.052632 | NaN |
| **2001** | NaN | NaN | 0.133333 | NaN | 0.200000 | 0.066667 | NaN | 0.600000 | NaN |
| **2002** | 0.062500 | 0.062500 | 0.187500 | NaN | 0.187500 | 0.125000 | 0.062500 | 0.312500 | NaN |
| **2003** | 0.130435 | 0.217391 | 0.260870 | 0.043478 | 0.065217 | 0.130435 | 0.043478 | 0.108696 | NaN |
| **2004** | 0.070423 | 0.070423 | 0.098592 | 0.042254 | 0.154930 | 0.366197 | 0.042254 | 0.140845 | 0.014085 |
| **2005** | 0.090909 | 0.181818 | 0.109091 | NaN | 0.200000 | 0.218182 | 0.018182 | 0.181818 | NaN |
| **2006** | 0.090909 | 0.030303 | 0.242424 | 0.166667 | 0.151515 | 0.090909 | 0.030303 | 0.136364 | 0.015152 |
| **2007** | 0.089286 | 0.160714 | 0.196429 | 0.035714 | 0.142857 | 0.035714 | 0.017857 | 0.321429 | NaN |
| **2008** | 0.216216 | 0.090090 | 0.189189 | 0.018018 | 0.126126 | 0.126126 | 0.009009 | 0.207207 | 0.009009 |
| **2009** | 0.256410 | 0.115385 | 0.064103 | 0.012821 | 0.243590 | 0.064103 | 0.038462 | 0.166667 | 0.038462 |
| **2010** | 0.075472 | 0.122642 | 0.264151 | 0.018868 | 0.169811 | 0.066038 | 0.018868 | 0.235849 | 0.028302 |
| **2011** | 0.148699 | 0.063197 | 0.308550 | 0.137546 | 0.122677 | 0.074349 | 0.063197 | 0.066914 | 0.011152 |
| **2012** | 0.126437 | 0.045977 | 0.379310 | 0.143678 | 0.132184 | 0.063218 | 0.028736 | 0.080460 | NaN |
| **2013** | 0.104575 | 0.098039 | 0.274510 | 0.065359 | 0.091503 | 0.084967 | 0.143791 | 0.117647 | 0.019608 |
| **2014** | 0.169643 | 0.160714 | 0.178571 | 0.062500 | 0.133929 | 0.107143 | 0.107143 | 0.071429 | 0.008929 |
| **2015** | 0.117647 | 0.067227 | 0.092437 | 0.134454 | 0.252101 | 0.033613 | 0.084034 | 0.218487 | NaN |
| **2016** | 0.067797 | 0.050847 | 0.237288 | 0.237288 | 0.152542 | 0.050847 | 0.118644 | 0.084746 | NaN |

We see that different regions are impacted by power outages differently. For instance, the Southwest typically has the third shortest outages on average but it once had a power outage 31 times longer than the average, which is the largest such ratio for any region.

### Assessment of Missingness

We looked at all of the columns to find a column that is NMAR. We found the column

$$'CAUSE.CATEGORY.DETAIL'$$

to be likely NMAR. We see that 30.7% of the values are missing.

In [33]:
```python
str(round(df['CAUSE.CATEGORY.DETAIL'].isna().mean()*100,2))+'%'
```

Out[33]:
```
'30.7%'
```

Though the preceding column, a column highly related in name and meaning, a column which is the categorization of the cause without the detail, is not missing at all.

In [34]:
```python
str(df['CAUSE.CATEGORY'].isna().sum())+' missing values'
```

Out[34]:
```
'0 missing values'
```

These two columns are related: the cause.category.detail column is a more detailed version of cause.category. The cause.category column is not missing at all, and the cause.category.detail column is missing a lot. Some of the nonmissing values look like this.

In [35]:
```python
for cause_detail in list(df[~df['CAUSE.CATEGORY.DETAIL'].isna()]['CAUSE.CATEGORY.DETAIL'].unique()[:10]):
    print(cause_detail, end=', ')
print('\b\b...')
```

```
vandalism, heavy wind, thunderstorm, winter storm, tornadoes, sabotage, hailstorm, uncontrolled loss, winter, wind sto
rm...
```

We postulate that it takes some effort to source the detailed cause of an outage, in a way that depends on the true details of the cause, thus cause.category.detail is NMAR. We cannot confirm this because we do not have the missing values, but we think that there are missing values for causes that are hard to explain. Not all power outage cause details are equally documentable/explainable, and we think that the harder to document/explain causes are filled in with NAN instead of properly described.

### Missingness Dependency

In [108]:
```
x = get_data()[['CLIMATE.REGION','U.S._STATE','NERC.REGION','CAUSE.CATEGORY']]
```

In [109]:

```
x['CR_missing'] = x['CLIMATE.REGION'].isna()
one_dist = (x.pivot_table(index='U.S._STATE', columns='CR_missing', aggfunc='size')
)

one_dist.columns = ['CLIMATE.REGION_missing = False', 'CLIMATE.REGION_missing = True']
one_dist = one_dist / one_dist.sum()
one_dist[~one_dist['CLIMATE.REGION_missing = True'].isna()]
one_dist
```

Out[109]:

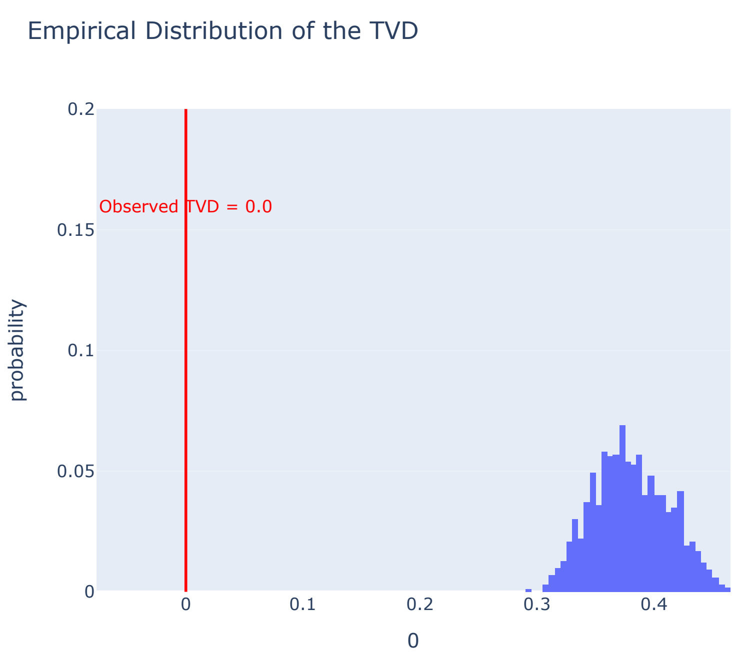
| U.S._STATE | CLIMATE.REGION_missing = False | CLIMATE.REGION_missing = True |
|---|---|---|
| Alabama | 0.003927 | NaN |
| Alaska | NaN | 0.166667 |
| Arizona | 0.018325 | NaN |
| Arkansas | 0.016361 | NaN |
| California | 0.137435 | NaN |
| Colorado | 0.009817 | NaN |
| Connecticut | 0.011780 | NaN |
| Delaware | 0.026832 | NaN |
| District of Columbia | 0.006545 | NaN |
| Florida | 0.029450 | NaN |
| Georgia | 0.011126 | NaN |
| Hawaii | NaN | 0.833333 |
| Idaho | 0.005890 | NaN |
| Illinois | 0.030105 | NaN |
| Indiana | 0.028141 | NaN |
| Iowa | 0.005236 | NaN |
| Kansas | 0.005890 | NaN |
| Kentucky | 0.008508 | NaN |
| Louisiana | 0.026178 | NaN |
| Maine | 0.012435 | NaN |
| Maryland | 0.037958 | NaN |
| Massachusetts | 0.011780 | NaN |
| Michigan | 0.062173 | NaN |
| Minnesota | 0.009817 | NaN |
| Mississippi | 0.002618 | NaN |
| Missouri | 0.011126 | NaN |
| Montana | 0.001963 | NaN |
| Nebraska | 0.002618 | NaN |
| Nevada | 0.004581 | NaN |
| New Hampshire | 0.009162 | NaN |
| New Jersey | 0.022906 | NaN |
| New Mexico | 0.005236 | NaN |
| New York | 0.046466 | NaN |
| North Carolina | 0.026178 | NaN |
| North Dakota | 0.001309 | NaN |
| Ohio | 0.028141 | NaN |
| Oklahoma | 0.015707 | NaN |
| Oregon | 0.017016 | NaN |
| Pennsylvania | 0.037304 | NaN |
| South Carolina | 0.005236 | NaN |
| South Dakota | 0.001309 | NaN |
| Tennessee | 0.022251 | NaN |
| Texas | 0.083115 | NaN |
| Utah | 0.026832 | NaN |
| Vermont | 0.005890 | NaN |
| Virginia | 0.024215 | NaN |
| Washington | 0.063482 | NaN |
| West Virginia | 0.002618 | NaN |
| Wisconsin | 0.013089 | NaN |
| Wyoming | 0.003927 | NaN |

In [110]:
```python
n_repetitions = 1000
shuffled = x.copy()

tvds = []

for _ in range(n_repetitions):
    shuffled['U.S._STATE'] =  np.random.permutation(shuffled['U.S._STATE'])

    pivoted = (
        shuffled
        .pivot_table(index='U.S._STATE', columns='CR_missing', aggfunc='size')
        .apply(lambda x: x / x.sum()))

    tvd = pivoted.diff(axis=1).iloc[:, -1].abs().sum() / 2
    tvds.append(tvd)
```

In [111]:
```python
observed_tvd = one_dist.diff(axis=1).iloc[:, -1].abs().sum() / 2
observed_tvd
```

Out[111]: 0.0

In [112]:
```python
fig = px.histogram(pd.DataFrame(tvds), x=0, nbins=50, histnorm='probability',
                   title='Empirical Distribution of the TVD')
fig.add_vline(x=observed_tvd, line_color='red')
fig.add_annotation(text=f'<span style="color:red">Observed TVD = {round(observed_tvd, 2)}</span>',
                   x=2.3 * observed_tvd, showarrow=False, y=0.16)
fig.update_layout(yaxis_range=[0, 0.2])
```




In [113]:
```python
fig.write_html('missingness-test.html', include_plotlyjs='cdn')
```

In [41]:
```
x
```

Out[41]:

| | CLIMATE.REGION | U.S._STATE | NERC.REGION | CAUSE.CATEGORY | CR_missing |
|---|---|---|---|---|---|
| **0** | East North Central | Minnesota | MRO | severe weather | False |
| **1** | East North Central | Minnesota | MRO | intentional attack | False |
| **2** | East North Central | Minnesota | MRO | severe weather | False |
| **3** | East North Central | Minnesota | MRO | severe weather | False |
| **4** | East North Central | Minnesota | MRO | severe weather | False |
| **...** | ... | ... | ... | ... | ... |
| **1529** | West North Central | North Dakota | MRO | public appeal | False |
| **1530** | West North Central | North Dakota | MRO | fuel supply emergency | False |
| **1531** | West North Central | South Dakota | RFC | islanding | False |
| **1532** | West North Central | South Dakota | MRO | islanding | False |
| **1533** | NaN | Alaska | ASCC | equipment failure | True |

1534 rows × 5 columns

In [42]:
```
y = x
y['CR_missing'] = y['CLIMATE.REGION'].isna()
two_dist = (y.pivot_table(index='NERC.REGION', columns='CR_missing', aggfunc='size')
)

two_dist.columns = ['CLIMATE.REGION_missing = False', 'CLIMATE.REGION_missing = True']
two_dist = two_dist / two_dist.sum()
two_dist[~two_dist['CLIMATE.REGION_missing = True'].isna()]
two_dist
```

Out[42]:

| NERC.REGION | CLIMATE.REGION_missing = False | CLIMATE.REGION_missing = True |
|---|---|---|
| **ASCC** | NaN | 0.166667 |
| **ECAR** | 0.022251 | NaN |
| **FRCC** | 0.028796 | NaN |
| **FRCC, SERC** | 0.000654 | NaN |
| **HECO** | NaN | 0.500000 |
| **HI** | NaN | 0.166667 |
| **MRO** | 0.030105 | NaN |
| **NPCC** | 0.098168 | NaN |
| **PR** | NaN | 0.166667 |
| **RFC** | 0.274215 | NaN |
| **SERC** | 0.134162 | NaN |
| **SPP** | 0.043848 | NaN |
| **TRE** | 0.072644 | NaN |
| **WECC** | 0.295157 | NaN |

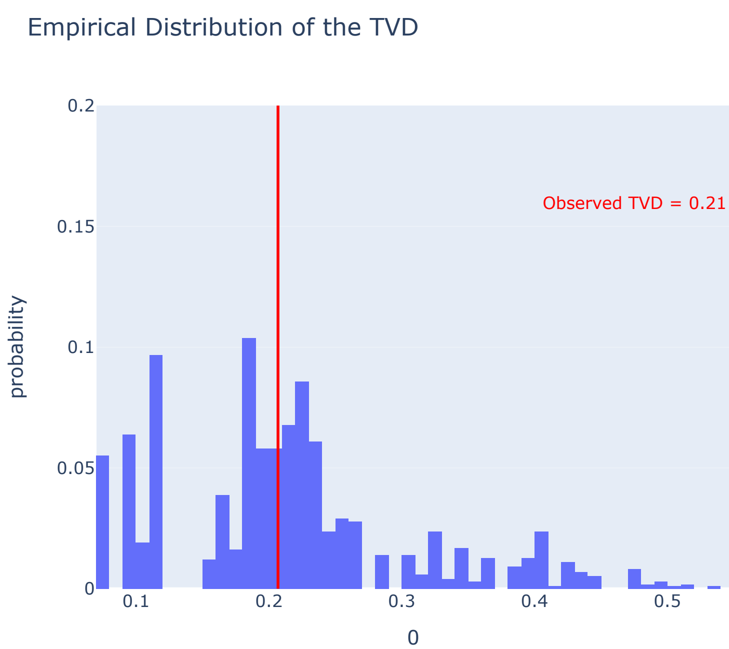
In [43]:
```
n_repetitions = 500
shuffled = y.copy()

tvds = []

for _ in range(n_repetitions):
    shuffled['NERC.REGION'] =  np.random.permutation(shuffled['NERC.REGION'])

    pivoted = (
        shuffled
        .pivot_table(index='NERC.REGION', columns='CR_missing', aggfunc='size')
        .apply(lambda y: y / y.sum()))

    tvd = pivoted.diff(axis=1).iloc[:, -1].abs().sum() / 2
    tvds.append(tvd)
```

In [44]:
```python
observed_tvd = two_dist.diff(axis=1).iloc[:, -1].abs().sum() / 2
observed_tvd
two_dist.diff(axis=1)
```

Out[44]:

| NERC.REGION | CLIMATE.REGION_missing = False | CLIMATE.REGION_missing = True |
|---|---|---|
| ASCC | NaN | NaN |
| ECAR | NaN | NaN |
| FRCC | NaN | NaN |
| FRCC, SERC | NaN | NaN |
| HECO | NaN | NaN |
| HI | NaN | NaN |
| MRO | NaN | NaN |
| NPCC | NaN | NaN |
| PR | NaN | NaN |
| RFC | NaN | NaN |
| SERC | NaN | NaN |
| SPP | NaN | NaN |
| TRE | NaN | NaN |
| WECC | NaN | NaN |

In [105]:
```python
fig = px.histogram(pd.DataFrame(tvds), x=0, nbins=50, histnorm='probability',
                   title='Empirical Distribution of the TVD')
fig.add_vline(x=observed_tvd, line_color='red')
fig.add_annotation(text=f'<span style="color:red">Observed TVD = {round(observed_tvd, 2)}</span>',
                   x=2.3 * observed_tvd, showarrow=False, y=0.16)
fig.update_layout(yaxis_range=[0, 0.2])
```




In [ ]:

In [ ]:

In [ ]:

In [46]:
```python
z = x
z['CR_missing'] = z['CLIMATE.REGION'].isna()
three_dist = (z.pivot_table(index='CAUSE.CATEGORY', columns='CR_missing', aggfunc='size')
)

three_dist.columns = ['CLIMATE.REGION_missing = False', 'CLIMATE.REGION_missing = True']
three_dist = three_dist / three_dist.sum()
three_dist[~three_dist['CLIMATE.REGION_missing = True'].isna()]
three_dist
```

Out[46]:

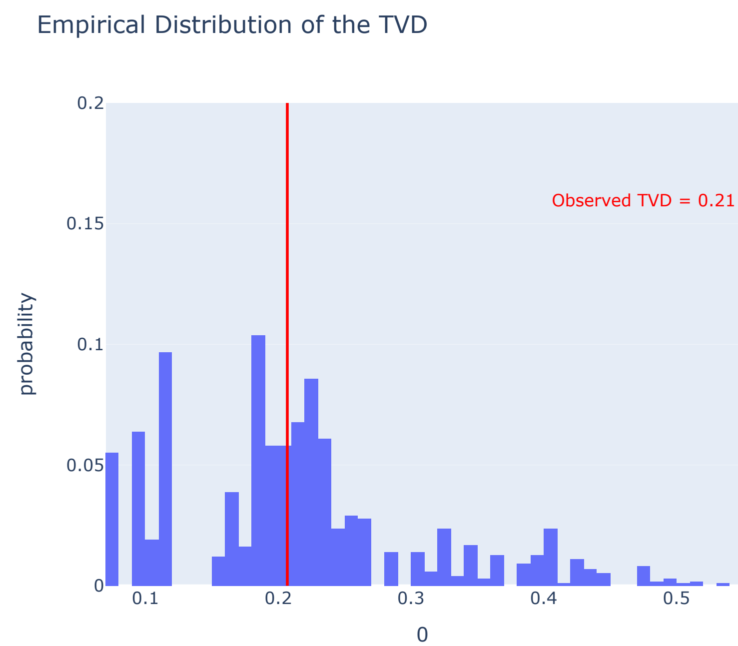
| | CLIMATE.REGION_missing = False | CLIMATE.REGION_missing = True |
|---|---|---|
| **CAUSE.CATEGORY** | | |
| **equipment failure** | 0.038613 | 0.166667 |
| **fuel supply emergency** | 0.033377 | NaN |
| **intentional attack** | 0.273560 | NaN |
| **islanding** | 0.030105 | NaN |
| **public appeal** | 0.045157 | NaN |
| **severe weather** | 0.496728 | 0.666667 |
| **system operability disruption** | 0.082461 | 0.166667 |

In [47]:
```python
n_repetitions = 1000
shuffled = z.copy()

tvds = []

for _ in range(n_repetitions):
    shuffled['CAUSE.CATEGORY'] =  np.random.permutation(shuffled['CAUSE.CATEGORY'])

    pivoted = (
        shuffled
        .pivot_table(index='CAUSE.CATEGORY', columns='CR_missing', aggfunc='size')
        .apply(lambda y: y / y.sum()))

    tvd = pivoted.diff(axis=1).iloc[:, -1].abs().sum() / 2
    tvds.append(tvd)
```

In [48]:
```python
observed_tvd = three_dist.diff(axis=1).iloc[:, -1].abs().sum() / 2
observed_tvd
```

Out[48]: 0.19109947643979056

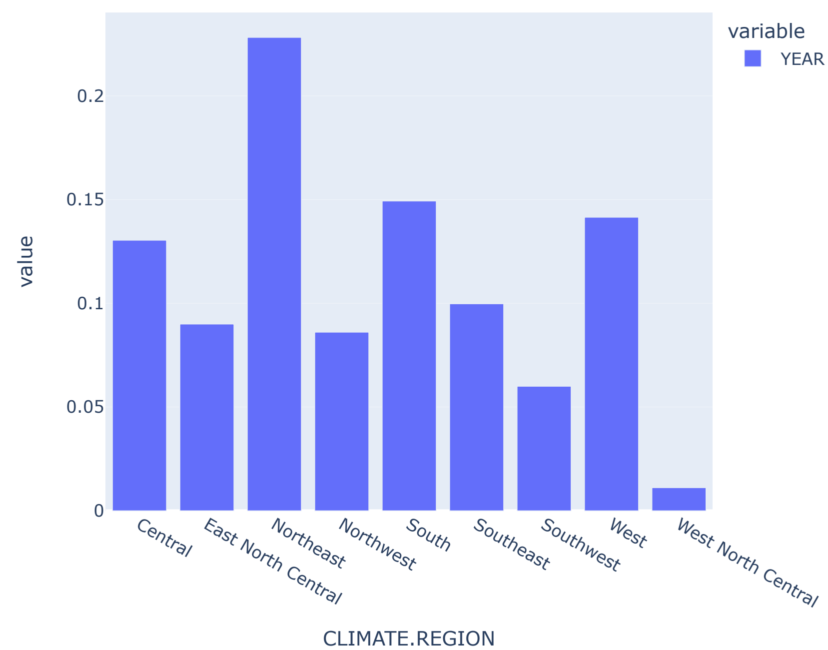
In [106]:
```python
fig = px.histogram(pd.DataFrame(tvds), x=0, nbins=50, histnorm='probability',
                   title='Empirical Distribution of the TVD')
fig.add_vline(x=observed_tvd, line_color='red')
fig.add_annotation(text=f'<span style="color:red">Observed TVD = {round(observed_tvd, 2)}</span>',
                   x=2.3 * observed_tvd, showarrow=False, y=0.16)
fig.update_layout(yaxis_range=[0, 0.2])
```




### Hypothesis Testing

Let's look at the region outage bar plot again. It gives the proportion of outages observed in each region. It sure looks like different regions have different rates of power outages.

In [83]:
```python
region_prop.plot(kind='bar')
```



Is this difference significant or can it be due to chance? We propose a hypothesis test for this claim.

Let $\{p_i : i \in Regions\}$ denote the set of proportion of power outages that occur in each region. That is, $p_r$ for any region $r$ was observed to be equal to the height of its bar. The following table shows the values of $p_i$ for all $i$.

In [86]:
```
region_prop.reset_index().rename(columns={'CLIMATE.REGION':'Climate region (i)','YEAR':'Proportion (p_i)'})
```

Out[86]:

| | Climate region (i) | Proportion (p_i) |
|---|---|---|
| 0 | Central | 0.130378 |
| 1 | East North Central | 0.089961 |
| 2 | Northeast | 0.228162 |
| 3 | Northwest | 0.086050 |
| 4 | South | 0.149283 |
| 5 | Southeast | 0.099739 |
| 6 | Southwest | 0.059974 |
| 7 | West | 0.141460 |
| 8 | West North Central | 0.011082 |

If all regions experience power outages at the same rate then we would expect to see the proportions close to $\frac{1}{\#regions} = \frac{1}{9}$.

So we test

$H_0 : p_i = \frac{1}{9}$ for all $i$

vs

$H_1$ : not all of the $p_i$ are $\frac{1}{9}$.

We can let $\alpha = 0.001$.

There were 1534 reported outages.

In [52]:
```
df.shape[0]
```

Out[52]:
```
1534
```

We can simulate a sample from the null distribution by randomly assigning 1534 samples to nine categories uniformly. We can estimate the $p_i$ to be the proportion of outages that land in each region $i$. The null distribution says that the proportions are all $\frac{1}{9}$, and we can compare these samples from the null distribution to the null distribution by the TVD statistic. We can compute the TVD of the observed region proportions to the null distribution of proportions to get an idea of how extreme our observed proportions were if they truly came from the null distribution.

In [53]:
```
n=df.shape[0]
```

In [54]:
```
def sample_1534_outages():
    return np.random.choice(range(9),size=1534)
```

In [55]:
```
def compute_proportions_in_each_region_of_sample(sample):
    return np.unique(sample,return_counts=True)[1]/1534
```

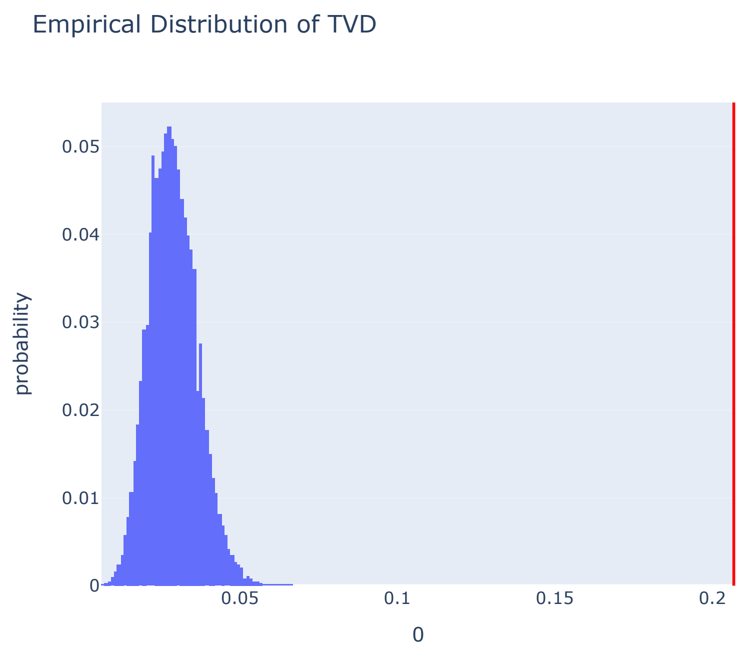
In [56]:
```
def compute_tvd(proportions):
    return np.abs(proportions - (1/9)).sum()/2
```

In [57]:
```
tvds_of_samples_under_null=[]
```

In [58]:
```
num_sims=10**5
for _ in range(num_sims):
    tvds_of_samples_under_null.append(
        compute_tvd(
            compute_proportions_in_each_region_of_sample(
                sample_1534_outages()
            )
        )
    )
```

In [59]:
```
observed_tvd=compute_tvd(np.array(region_prop['YEAR'].to_list()))
```

In [87]:
```
fig = px.histogram(pd.DataFrame(tvds_of_samples_under_null), x=0, nbins=75, histnorm='probability',
                   title='Empirical Distribution of TVD')
fig.add_vline(x=observed_tvd, line_color='red')
```




In [61]:
```
fig.write_html('test-file.html', include_plotlyjs='cdn')
```

The proportion of TVDS computed between the null distribution and its samples to the right (as TVD is unidirectional; bigger is more extreme) of the observed TVD is 0: none of the 100000 sample TVDs came close to the TVD that we saw. Thus since $0 < \alpha = 0.001$ we reject the null hypothesis that all regions get outages with the same rate, and we have evidence that some regions are more prone to power outages than others.