Power Outages

- **See the main project notebook for instructions to be sure you satisfy the rubric!**
- See Project 03 for information on the dataset.
- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
    - Predict the severity (number of customers, duration, or demand loss) of a major power outage.
    - Predict the cause of a major power outage.
    - Predict the number and/or severity of major power outages in the year 2020.
    - Predict the electricity consumption of an area.

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

Summary of Findings

Introduction

The dataset that we are using recorded the major electricity outage that occured in the US continent with different variables that might be related to the electricity outage and effects that brought with it. There are around 1500 cases of the electricity outage between 2000 and 2016. The main goal of our project is to predict the cause of outage base on the given dataset, which is a Classification question. Our target variable is 'CAUSE.CATEGORY' and it has already been catergorized into it's won unique values: 'severe weather','interntional attack','system operability disruption', 'public appreal','equipment failure', 'fuel supply emergency','islanding'.

Baseline Model

The total number of features is 5. Three of them will be nomial features. The total number of ordinal feature is one. The total number of quantative will be one. We will be using 'CLIMATE.CATEGORY', 'MONTH','OUTAGE.DURATION','CLIMATE.REGION', 'ANOMALY.LEVEL' to predict the CAUSE.CATEGORY of each outage. 'CLIMATE.CATEGORY', 'MONTH','CLIMATE.REGION' are nomial feature. 'ANOMALY.LEVEL' will be ordinal feature. The 'OUTAGE.DURATION' will be quatitive feature. The classifier that we use is decision tree classiier. We decide to use the accuracy score to be the metric of our prediction since we only want to the accuracy of our prediction on the cause of the outage and the weights for inaccurate is the same of accurate. The accuracy score that we got is less than 0.6. it isn't evry accurate, so we decide to add more features into our prediction.

Final Model

We were wondering is the climate region is related with outage duration. so we plot it out, and it seems like the north of the country seems to have longer duration. Normal and cold climates are the major climate for these region. Since the north region of the counry seems to have longer average duration, we decide to standardadize our duration by the climate region by using StdScalerByGroup. We also decide to treat the duration as a ordinal feature since there are several causes that are more likely to cause long duration. We realized that we are have to many inrelavant features, so we dropped the Month. We are still using decision tree classifier. We select the best parameter of the grid search cv {'classifier__max_depth': 7, 'classifier__max_leaf_nodes': 10}. This bring our accuracy score to about 0.65.

Fairness Evaluation

For the fairness evaluation, we want to discover whether our model is biased about different year of the data. We come up with the null hypothesis that our model is not fair that the difference between the two accuricies is not due to random chance. And the alternative hypothesis is that the differences between the accuricies is due to random choice, and our model is fair. We first find the median of the year of the outages data and we separate the data into 2 groups, one is before the median year, 2008, and the other is after the year of 2008. We tested the accuricies of the 2 groups and calculated the difference between the 2 groups as our observed difference. Then we run a permutation test to see whether our observed difference will fall into our significance level, which we set it to 0.05. Eventually, we got a p-value of 0.05 after 500 times permutation, thus we rejected our null hypothesis and concluded that our model is fair towards the two separated groups.

Code

In [644]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

In [645]:
```
df = pd.read_excel('outage.xlsx')
columns = np.array(df.iloc[4])
measurement = np.array(df.iloc[5])
df = df.drop(df.index[[0,1,2,3,4,5]]).set_axis(columns, axis=1, inplace=False)
x = df['OUTAGE.START.DATE'].fillna('x').astype(str).str.replace('00:00:00','').replace('x','1770-1-
y = df['OUTAGE.START.TIME'].fillna('x').astype(str).str.replace('00:00:00','').replace('x','01:01:0
df['OUTAGE.START'] = pd.to_datetime(x + ' ' + y)
z = df['OUTAGE.RESTORATION.DATE'].fillna('x').astype(str).str.replace('00:00:00','').replace('x','1
h = df['OUTAGE.RESTORATION.TIME'].fillna('x').astype(str).str.replace('00:00:00','').replace('x','0
df['OUTAGE.RESTORATION'] = pd.to_datetime(z + ' ' + h)
df = df.drop(columns = ['OUTAGE.START.DATE','OUTAGE.START.TIME','OUTAGE.RESTORATION.DATE','OUTAGE.R
df['OUTAGE.START'] = df['OUTAGE.START'].replace(pd.to_datetime('1770-01-01 01:01:01'),np.nan)
df['OUTAGE.RESTORATION'] = df['OUTAGE.RESTORATION'].replace(pd.to_datetime('1770-01-01 01:01:01'),n
df = df.reset_index().drop(columns = ['index'])
df['OUTAGE.DURATION'] = df['OUTAGE.DURATION'].agg(lambda x : x.replace(0,np.nan))
df = df.drop('variables', axis=1).set_index('OBS')
df = df.dropna(subset = ['OUTAGE.DURATION'])
outage = df.copy()
outage
```

	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVEL	CLIMATE.CATEGORY
OBS								
1	2011	7	Minnesota	MN	MRO	East North Central	-0.3	normal
2	2014	5	Minnesota	MN	MRO	East North Central	-0.1	normal
3	2010	10	Minnesota	MN	MRO	East North Central	-1.5	cold
4	2012	6	Minnesota	MN	MRO	East North Central	-0.1	normal
5	2015	7	Minnesota	MN	MRO	East North Central	1.2	warm
...	...	...	...	...	...	...	...	...
1526	2011	6	Idaho	ID	WECC	Northwest	-0.3	normal
1529	2016	7	Idaho	ID	WECC	Northwest	-0.3	normal
1530	2011	12	North Dakota	ND	MRO	West North Central	-0.9	cold
1532	2009	8	South Dakota	SD	RFC	West North Central	0.5	warm
1533	2009	8	South Dakota	SD	MRO	West North Central	0.5	warm

1398 rows × 53 columns

In [646]:

```
outage
# checking all cols in the dataframe and determine
# which is relevant to the number of customers affected by the outages
outage.dtypes
```

```
YEAR                              object
MONTH                             object
U.S._STATE                        object
POSTAL.CODE                       object
NERC.REGION                       object
CLIMATE.REGION                    object
ANOMALY.LEVEL                     object
CLIMATE.CATEGORY                  object
CAUSE.CATEGORY                    object
CAUSE.CATEGORY.DETAIL             object
HURRICANE.NAMES                   object
OUTAGE.DURATION                  float64
DEMAND.LOSS.MW                    object
CUSTOMERS.AFFECTED                object
RES.PRICE                         object
COM.PRICE                         object
IND.PRICE                         object
TOTAL.PRICE                       object
RES.SALES                         object
COM.SALES                         object
IND.SALES                         object
TOTAL.SALES                       object
RES.PERCEN                        object
COM.PERCEN                        object
IND.PERCEN                        object
RES.CUSTOMERS                     object
COM.CUSTOMERS                     object
IND.CUSTOMERS                     object
TOTAL.CUSTOMERS                   object
RES.CUST.PCT                      object
COM.CUST.PCT                      object
IND.CUST.PCT                      object
PC.REALGSP.STATE                  object
PC.REALGSP.USA                    object
PC.REALGSP.REL                    object
PC.REALGSP.CHANGE                 object
UTIL.REALGSP                      object
TOTAL.REALGSP                     object
UTIL.CONTRI                       object
PI.UTIL.OFUSA                     object
POPULATION                        object
POPPCT_URBAN                      object
POPPCT_UC                         object
POPDEN_URBAN                      object
POPDEN_UC                         object
POPDEN_RURAL                      object
AREAPCT_URBAN                     object
AREAPCT_UC                        object
PCT_LAND                          object
PCT_WATER_TOT                     object
PCT_WATER_INLAND                  object
OUTAGE.START              datetime64[ns]
OUTAGE.RESTORATION        datetime64[ns]
dtype: object
```

In [647]:

```
# checking null values

df.isna().sum(axis = 0)
df.head()
```

OBS	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVEL	CLIMATE.CATEGORY
1	2011	7	Minnesota	MN	MRO	East North Central	-0.3	normal
2	2014	5	Minnesota	MN	MRO	East North Central	-0.1	normal
3	2010	10	Minnesota	MN	MRO	East North Central	-1.5	cold
4	2012	6	Minnesota	MN	MRO	East North Central	-0.1	normal
5	2015	7	Minnesota	MN	MRO	East North Central	1.2	warm

5 rows × 53 columns

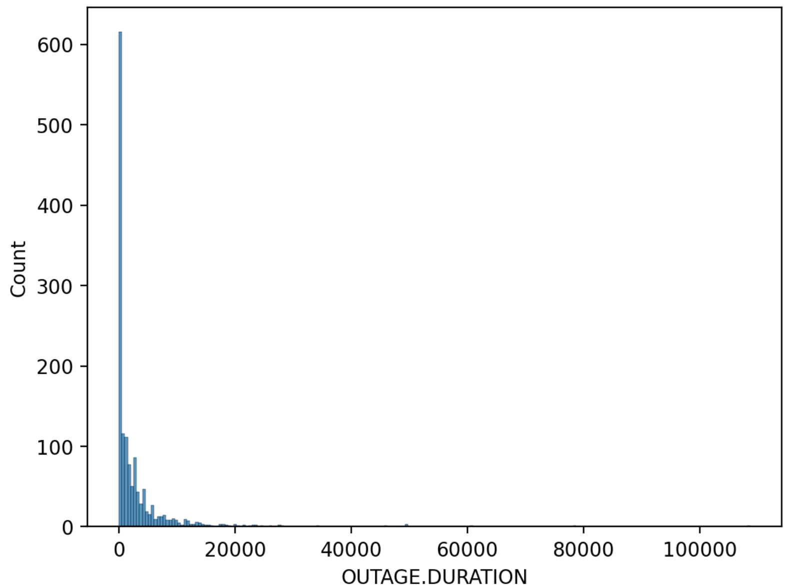
In [648]:

```
durations = outage['OUTAGE.DURATION'].value_counts()
durations
```

```
1.0        97
2880.0     15
300.0      14
60.0       14
1440.0     13
           ..
565.0       1
7298.0      1
11700.0     1
7987.0      1
1548.0      1
Name: OUTAGE.DURATION, Length: 851, dtype: int64
```

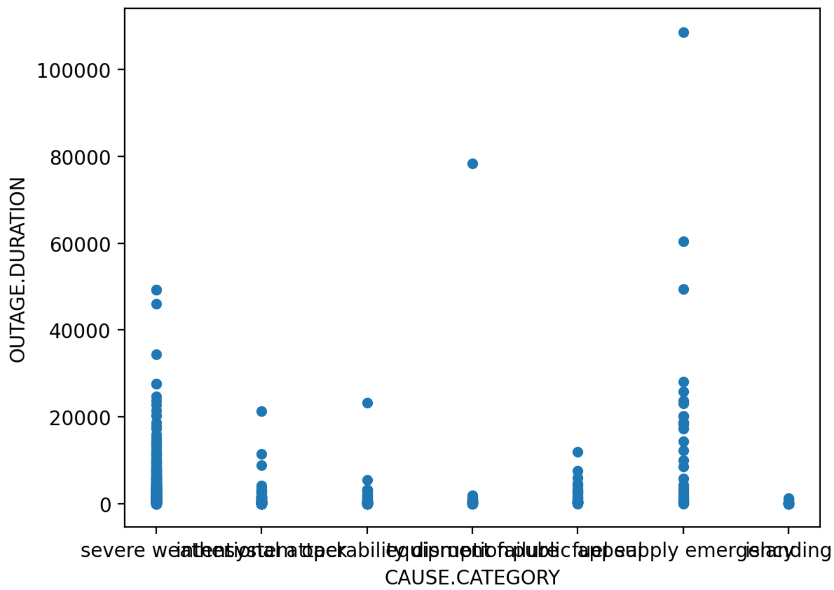
In [649]:
```
sns.histplot(outage['OUTAGE.DURATION'])
```

```
<AxesSubplot:xlabel='OUTAGE.DURATION', ylabel='Count'>
```



In [650]:
```
df.loc[:,['OUTAGE.DURATION','CAUSE.CATEGORY']].plot.scatter(x = 'CAUSE.CATEGORY',y = 'OUTAGE.DURATI
```

```
<AxesSubplot:xlabel='CAUSE.CATEGORY', ylabel='OUTAGE.DURATION'>
```



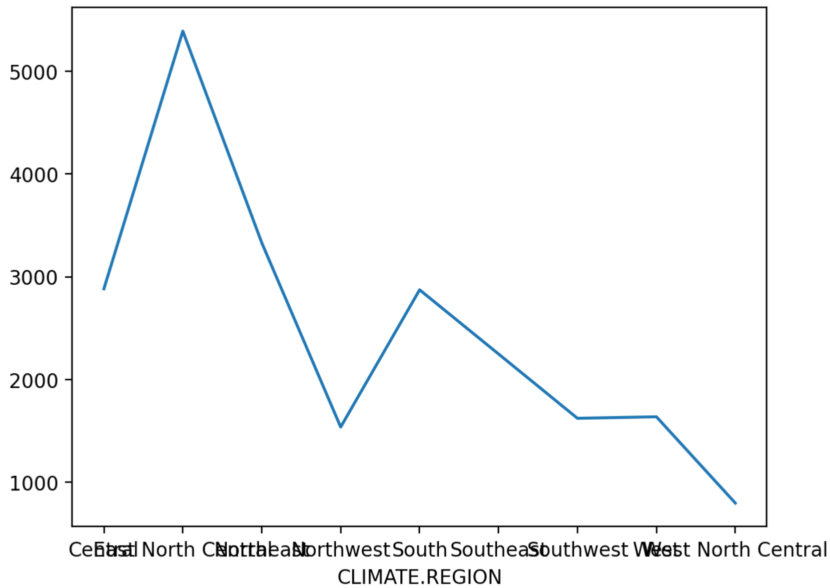
In [651]:
```
# duration_weather_cat = outage.groupby('CLIMATE.CATEGORY')['OUTAGE.DURATION'].mean()
# duration_weather_cat
```

In [652]:
```
# outage['CAUSE.CATEGORY'].value_counts()
```

In [653]:

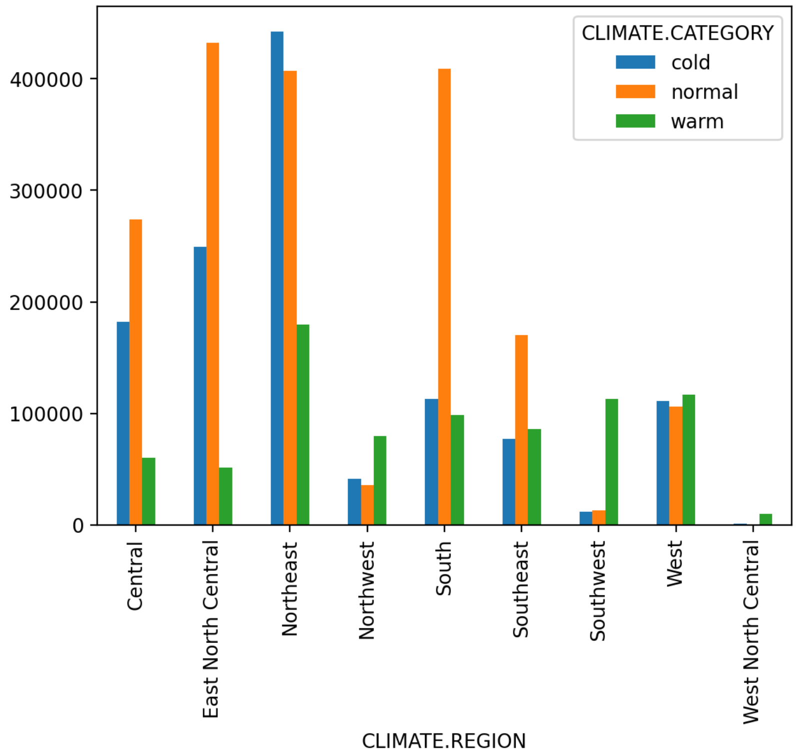
```
df.groupby('CLIMATE.REGION')['OUTAGE.DURATION'].mean().plot()
```

```
<AxesSubplot:xlabel='CLIMATE.REGION'>
```



In [654]:
```
pi = df.pivot_table(
    index = 'CLIMATE.REGION',
    columns = 'CLIMATE.CATEGORY',
    values = 'OUTAGE.DURATION',
    aggfunc = 'sum'
)
pi.plot(kind = 'bar')
```

```
<AxesSubplot:xlabel='CLIMATE.REGION'>
```



In [655]:
```
outage = outage[outage['OUTAGE.DURATION'].notna()]
outage.head()
```

	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVEL	CLIMATE.CATEGORY
OBS								
1	2011	7	Minnesota	MN	MRO	East North Central	−0.3	normal
2	2014	5	Minnesota	MN	MRO	East North Central	−0.1	normal
3	2010	10	Minnesota	MN	MRO	East North Central	−1.5	cold
4	2012	6	Minnesota	MN	MRO	East North Central	−0.1	normal
5	2015	7	Minnesota	MN	MRO	East North Central	1.2	warm

5 rows × 53 columns

Baseline Model

In [686]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, train_test_split
```

In [687]:
```
.DURATION','CLIMATE.REGION','ANOMALY.LEVEL','MONTH','CLIMATE.CATEGORY','YEAR']].dropna(axis =0).index
```

In [688]:
```
## split the data into training and testing
X = outage.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL','MONTH','CLIMATE.CATEGORY', 'Y
y = outage.loc[indx,'CAUSE.CATEGORY']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3)
```

In [689]:
```
## transform the columns
column_tranformer = ColumnTransformer(
    transformers = [
        ('std',StandardScaler(), ['OUTAGE.DURATION','ANOMALY.LEVEL', 'YEAR']),
        ('ohh',OneHotEncoder(),[ 'CLIMATE.CATEGORY','CLIMATE.REGION','MONTH']),

    ], remainder = 'passthrough'
)
```

In [690]:
```
### combine transformer and randome forest classifier
pred = Pipeline([
    ('col_trans', column_tranformer),
    ('classifier', DecisionTreeClassifier())
])
y_pred = pred.fit(X_train,y_train).predict(X_test)
print('Accuracy',metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy 0.6578947368421053
```

Final Model

In [692]:
```
pred.get_params().keys()
```

```
dict_keys(['memory', 'steps', 'verbose', 'col_trans', 'classifier', 'col_trans__n_jobs', 'col_trans__remainder', 'col_tr
ans__sparse_threshold', 'col_trans__transformer_weights', 'col_trans__transformers', 'col_trans__verbose', 'col_trans__v
erbose_feature_names_out', 'col_trans__std', 'col_trans__ohh', 'col_trans__std__copy', 'col_trans__std__with_mean', 'col
_trans__std__with_std', 'col_trans__ohh__categories', 'col_trans__ohh__drop', 'col_trans__ohh__dtype', 'col_trans__ohh__
handle_unknown', 'col_trans__ohh__sparse', 'classifier__ccp_alpha', 'classifier__class_weight', 'classifier__criterion',
'classifier__max_depth', 'classifier__max_features', 'classifier__max_leaf_nodes', 'classifier__min_impurity_decrease',
'classifier__min_samples_leaf', 'classifier__min_samples_split', 'classifier__min_weight_fraction_leaf', 'classifier__ra
ndom_state', 'classifier__splitter'])
```

In [693]:
```
## for ordinal encoder
outage['OUTAGE.DURATION'].max()
ran = [[i for i in range(108654)]]
```

In [694]:
```
# stdscalerbygroup
from sklearn.base import BaseEstimator, TransformerMixin
import numpy
class StdScalerByGroup(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def fit(self, X, y=None):
        """
        :Example:
        >>> cols = {'g': ['A', 'A', 'B', 'B'], 'c1': [1, 2, 2, 2], 'c2': [3, 1, 2, 0]}
        >>> X = pd.DataFrame(cols)
        >>> std = StdScalerByGroup().fit(X)
        >>> std.grps_ is not None
        True
        """
        # X might not be a pandas DataFrame (e.g. a np.array)
        df = pd.DataFrame(X)

        # Compute and store the means/standard-deviations for each column (e.g. 'c1' and 'c2'),
        # for each group (e.g. 'A', 'B', 'C').
        # (Our solution uses a dictionary)
        mean_values=df.groupby(df.columns[0]).mean().values
        std_values=df.groupby(df.columns[0]).std().values
        self.grps_ = {'mean': mean_values , 'std':std_values }
        return self

    def transform(self, X, y=None):
        """
        :Example:
        >>> cols = {'g': ['A', 'A', 'B', 'B'], 'c1': [1, 2, 3, 4], 'c2': [1, 2, 3, 4]}
        >>> X = pd.DataFrame(cols)
        >>> std = StdScalerByGroup().fit(X)
        >>> out = std.transform(X)
        >>> out.shape == (4, 2)
        True
        >>> np.isclose(out.abs(), 0.707107, atol=0.001).all().all()
        True
        """

        # Hint: Define a helper function here!
        df=pd.DataFrame(X)

        res=pd.DataFrame()
        first_col = df.columns[0]
        unique_group=df[df.columns[0]].unique()



        row=0
        for i in unique_group:
            col=0

            temp=df.loc[df[first_col]==i]
```

```
            data=temp.copy()


            for j in temp.columns[1:].tolist():
                tmean=self.grps_['mean'][row][col]
                tstd=self.grps_['std'][row][col]

                data[j]=(temp[j]-tmean)/tstd

                col=col+1


            data=data.drop(columns=data.columns[0])
            row  = row +1

            res=pd.concat([res,data],ignore_index=True)

        return res
```

In [695]:
```
## column transformers
## make outage duration into ordinal

column_tranformer = ColumnTransformer(
    transformers = [
        ('std',StandardScaler(), ['YEAR']),
        ('sbg',StdScalerByGroup(),['CLIMATE.REGION','OUTAGE.DURATION']),
        ('ordi',OrdinalEncoder(categories = ran), ['OUTAGE.DURATION']),
        ('ohh',OneHotEncoder(),['CLIMATE.CATEGORY', 'CLIMATE.REGION'])
    ], remainder = 'passthrough'
)
```

In [696]:
```
pred = Pipeline([
    ('col_trans', column_tranformer),
    ('classifier', DecisionTreeClassifier(max_depth = 2))
])
y_pred = pred.fit(X_train,y_train).predict(X_test)
print('Accuracy',metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy 0.6602870813397129
```

In [697]:
```
## setting hyperparameters,
hyperparameters = {
    'classifier__max_depth': [3,5,7,10],
    'classifier__max_leaf_nodes' : [4,6,8,10,15,20]
#     'classifier__max_depth': [i for i in np.arange(5,1300,100)]
#     'classifier__class_weight': [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1},None]

}
```

In [698]:
```
pred.set_params(classifier__max_depth = 7)
pred.set_params(classifier__max_leaf_nodes = 10)
y_pred = pred.fit(X_train,y_train).predict(X_test)
print('Accuracy',metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy 0.6818181818181818
```

Fairness Evaluation

Null Hypothesis: The differences between prediction accuricies from before 2008 and after is due to random choice.

Alternative Hypothesis: The differences between the predicted accuricies are not randomly distibuted.

Significace level: 0.05

In [724]:
```
# finding the median of the year in dataframe
year_median = np.median(list(outage['YEAR'].unique()))
year_median
```

```
2008.0
```

In [725]:
```
# grouping the year
early_year = outage[outage['YEAR'] < year_median]
late_year = outage[outage['YEAR'] >= year_median]
print(early_year.size, late_year.size)
```

```
17543 56551
```

In [767]:
```
column_tranformer = ColumnTransformer(
    transformers = [
        ('std',StandardScaler(), ['YEAR', 'OUTAGE.DURATION']),
        #('sbg',StdScalerByGroup(),['CLIMATE.REGION','OUTAGE.DURATION']),
        ('ordi',OrdinalEncoder(categories = ran), ['OUTAGE.DURATION']),
        ('ohh',OneHotEncoder(),['CLIMATE.CATEGORY', 'CLIMATE.REGION'])
    ], remainder = 'passthrough'
)
```

In [768]:
```
pred = Pipeline([
    ('col_trans', column_tranformer),
    ('classifier', DecisionTreeClassifier(max_depth = 2))
])
pred.set_params(classifier__max_depth = 7)
pred.set_params(classifier__max_leaf_nodes = 10)
```

```
Pipeline(steps=[('col_trans',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('std', StandardScaler(),
                                                  ['YEAR', 'OUTAGE.DURATION']),
                                                 ('ordi',
                                                  OrdinalEncoder(categories=[[0,
                                                                              1,
                                                                              2,
                                                                              3,
                                                                              4,
                                                                              5,
                                                                              6,
                                                                              7,
                                                                              8,
                                                                              9,
                                                                              10,
                                                                              11,
                                                                              12,
                                                                              13,
                                                                              14,
                                                                              15,
                                                                              16,
                                                                              17,
                                                                              18,
                                                                              19,
                                                                              20,
                                                                              21,
                                                                              22,
                                                                              23,
                                                                              24,
                                                                              25,
                                                                              26,
                                                                              27,
                                                                              28,
                                                                              29, ...]]),
                                                  ['OUTAGE.DURATION']),
                                                 ('ohh', OneHotEncoder(),
                                                  ['CLIMATE.CATEGORY',
                                                   'CLIMATE.REGION'])])),
                ('classifier',
                 DecisionTreeClassifier(max_depth=7, max_leaf_nodes=10))])
```

In [769]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, train_test_split
```

In [770]:
```
# predictions on early years
X = early_year.drop('CAUSE.CATEGORY', axis=1)
y = early_year['CAUSE.CATEGORY']
X = X.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL','MONTH','CLIMATE.CATEGORY', 'YEAR']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
pred.fit(X_train, y_train)
early_predictions = pred.predict(X_test)
early = metrics.accuracy_score(y_test, early_predictions)
print("Accuracy on early years: ", early)
```

```
Accuracy on early years:  0.7951807228915663
```

In [771]:
```
# predictions on late years
X = late_year.drop('CAUSE.CATEGORY', axis=1)
y = late_year['CAUSE.CATEGORY']
X = X.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL','MONTH','CLIMATE.CATEGORY', 'YEAR']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
pred.fit(X_train, y_train)
late_predictions = pred.predict(X_test)
late = metrics.accuracy_score(y_test, late_predictions)
print("Accuracy on late years: ", late)
```

```
Accuracy on late years:  0.6666666666666666
```

Hypothesis Test

In [742]:
```
obs = abs(early - late)
obs
```

```
0.1426830919182347
```

In [847]:
```
column_tranformer = ColumnTransformer(
    transformers = [
        ('std',StandardScaler(), ['YEAR']),
        #('sbg',StdScalerByGroup(),['CLIMATE.REGION','OUTAGE.DURATION']),
        ('ordi',OrdinalEncoder(categories = ran), ['OUTAGE.DURATION']),
        ('ohh',OneHotEncoder(handle_unknown='ignore'),['CLIMATE.CATEGORY', 'CLIMATE.REGION'])
    ], remainder = 'passthrough'
)
```

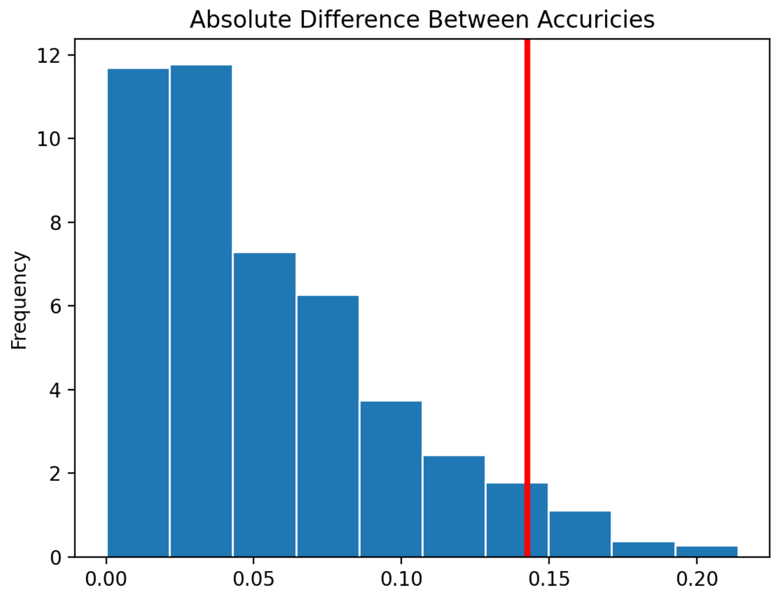
In [848]:

```
pred = Pipeline([
    ('col_trans', column_tranformer),
    ('classifier', DecisionTreeClassifier(max_depth = 6))
])
pred.set_params(classifier__max_depth = 7)
pred.set_params(classifier__max_leaf_nodes = 10)
```

```
Pipeline(steps=[('col_trans',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('std', StandardScaler(),
                                                  ['YEAR']),
                                                 ('ordi',
                                                  OrdinalEncoder(categories=[[0,
                                                                              1,
                                                                              2,
                                                                              3,
                                                                              4,
                                                                              5,
                                                                              6,
                                                                              7,
                                                                              8,
                                                                              9,
                                                                              10,
                                                                              11,
                                                                              12,
                                                                              13,
                                                                              14,
                                                                              15,
                                                                              16,
                                                                              17,
                                                                              18,
                                                                              19,
                                                                              20,
                                                                              21,
                                                                              22,
                                                                              23,
                                                                              24,
                                                                              25,
                                                                              26,
                                                                              27,
                                                                              28,
                                                                              29, ...]]),
                                                  ['OUTAGE.DURATION']),
                                                 ('ohh',
                                                  OneHotEncoder(handle_unknown='ignore'),
                                                  ['CLIMATE.CATEGORY',
                                                   'CLIMATE.REGION'])])),
                ('classifier',
                 DecisionTreeClassifier(max_depth=7, max_leaf_nodes=10))])
```

In [860]:
```
n_repetitions = 500

differences = []
for _ in range(n_repetitions):

    # Shuffle the year
    shuffled_year = (
        outage['YEAR']
        .sample(frac=1)
        .reset_index(drop=True)
    )

    shuffled = (
        outage
        .assign(**{'Shuffled Year': shuffled_year})
    )

    early = shuffled[shuffled['Shuffled Year'] < 2008]
    late = shuffled[shuffled['Shuffled Year'] >= 2008]

    eindx = (early.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL',
                          'MONTH','CLIMATE.CATEGORY','YEAR']].dropna(axis =0).index
           )
    lindx = (late.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL',
                          'MONTH','CLIMATE.CATEGORY','YEAR']].dropna(axis =0).index
           )
    X = early.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL','MONTH','CLIMATE.CATEGORY',
    e_idx = X.index
    y = early.loc[e_idx,'CAUSE.CATEGORY']
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25)
    pred.fit(X_train, y_train)
    early_predictions = pred.predict(X_test)
    early_accuracy = metrics.accuracy_score(y_test, early_predictions)

    X = late.loc[:,['OUTAGE.DURATION','CLIMATE.REGION','ANOMALY.LEVEL','MONTH','CLIMATE.CATEGORY',
    y = late.loc[lindx,'CAUSE.CATEGORY']
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25)
    pred.fit(X_train, y_train)
    late_predictions = pred.predict(X_test)
    late_accuracy = metrics.accuracy_score(y_test, late_predictions)

    # Store the result
    diff = abs(early_accuracy - late_accuracy)
    differences.append(diff)
```

In [861]:
```
title = 'Absolute Difference Between Accuricies'
pd.Series(differences).plot(kind='hist', density=True, ec='w', bins=10, title=title)
plt.axvline(x=obs, color='red', linewidth=3);
```



In [862]:
```
p_val = np.mean(np.array(differences) > obs)
p_val
```

```
0.05
```

We end up rejected the null hypothesis, and concluded that the difference distribution of the accuricies between prediction of the cause categories of the outages before 2008 and the after is due to randomly choice under the 0.05 significance level. Thus, we believe our model is fair for the statistics before 2008 and after.