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

For our project we are trying to predict whether or not a power outage is measured as a "major power outage". A major outage is defined as a power outage that impacted at least 50,000 customers or caused an unplanned firm load loss of atleast 300MW. Our data did not come with a variable indicating whether or not a power outage observation was defined as major. After pulling in our data and cleaning it we added a column to the dataframe that indicates to us whether or not a given power outage was defined as major. To do this, we classified those observations that impacted at least 50,000 customers or caused an unplanned firm load loss of atleast 300 MW as a 1 in the "major_outage" column and a 0 for those observations that did not meet either of the specified requirements. The problem we are attempting to solve is a classification problem where we are trying to predict whether any given observation has either a 0 or a 1 in the "major_outage" column based on other variables in our dataset. Our objective is to improve the accuracy between our baseline model and our final model.

In our outages dataset 785/1534 of our power outages were classified as major, or about 51%. After considering the real world implications of our model, we decided to chose accuracy as our objective because our data was pretty evenly split between major power outages and non major power outages. Since we ended up using the "outage duration" column in our model, our model would not likely be one used to predict a major power outage before the outage has ended, consequently telling us that in this case a false positive is just as bad as a false negative.

Baseline Model

For our baseline model we decided to use a Decision Tree Classifier with a max depth of 8 as our only parameter. We picked this model as it was the classification algorithm that we understood the most. We picked the number 8 as our baseline depth parameter based on the number of features we used and also just intuition. For this original model we decided to use US state, Year, Climate Region, Anomaly Level, Climate Category, and Outage Duration as our features to try and predict

the classification of a power outage. Year, Anomaly Level, and Outage Duration were orignally quantitative so they did not need to be engineered. US State, Climate Region, and Climate Category, however, were categorical variables and we decided to one hot encode them. This is a total of six overall features, three of which were originally quantitative and three of which were categorical. Out of these features Outage Duration and Anomaly Level were quantitative, Year was ordinal, and US State, Climate Region, and Climate Category were nominal.

After testing our baseline model we got a baseline accuracy of about 0.85 when training on our entire dataset. Overall, our accuracy from the baseline model was alright but we thought it could be improved by adding and engineering a few features and modifying the decision tree parameters. An accuracy of 0.85 is pretty high but we also wanted to test our model to make sure we were not overfitting and that our model could generalize to potentially unseen data.

Final Model

To improve the accuracy of our model we went through the following procedure:

First, we addded in and engineered the month column in an sklearn pipeline to categorize the months by season. We then onehotencoded the season column. We made this modification under the logic that some seasons of the year that have harsher weather might be more prone to having major power outages.

Second, we added in the Cause Category feature and binarized this column to give a 1 when the cause category was "severe weather" and a 0 otherwise. We made this modification after we noticed that out of the 763 power outages in our dataset that had severe weather as their cause, 662 of them were major power outages. And out of the 771 power outages that did not have severe weather as their cause only 123 were major power outages. This told us that the cause of a power outage being severe weather is likely a strong predictor of whether the power outage was major or not.

After engineering these features our accuracy improved to 0.92 when training on our entire dataset.

Third, we investigated how our model worked with a train test split along with the engineered features and some modified parameters. For our modified Decision Tree Classifier parameters we arbitrarily picked a max depth of 4, a min samples leaf of 2, and min samples split of 2. After running this a few times with train test split we found that we were getting a test accuracy of around 0.83.

Fourth, we performed a search for the best parameters of the Decision Tree Classifier. To do this we used the Grid Search Cross Validation with a cv of 8. After running this we found that a max depth of 3, min samples leaf of 7, and min samples split of 20 gave us a consistent accuracy on our test data of 0.86. These parameters reduced the accuracy when using training on our entire dataset, but they gave us the best accuracy when we generalized using train test split.

Lastly, we tried to search for a better classification algorithm. We tried to build a model with the KNeighbors Classifier and found the best parameters to see if this model could help improve the accuracy of our predictions. We thought that this model might make better predictions since some observations might be very similar to each other in our dataset. Overall, this model did not improve our accuracy, coming in around 0.73 when generalized, even after searching for the best parameters.

Ultimately we decided on the Decision Tree Classifier with max depth 3, min samples leaf 7, and min samples split of 20. Our final features were: US State (one hot encoded), Year (as is), Climate Region (one hot encoded), Anomaly Level (as is), Climate Category (one hot encoded), Outage Duration (as is), Month (transformed to season + one hot encoded), Cause Category (binarized). This model and feature combination consistently gave us the best accuracy overall.

Fairness Evaluation

For our fairness evaluation we wanted to test whether our final model worked better for long power outages or short power outages. Here we defined a long power outage as one that had an outage duration over the mean outage duration among all of the power outages and a short power outage as one that had an outage duration less than the mean outage duration among all of the power outages. We set a 0.05 significance level and our null + alternative hypotheses to be:

{use of accuracy justified in introduction}

Null Hypothesis: my model is fair, the accuracy for our two subsets are roughly the same

Alternative Hypothesis: my model is unfair; the accuracy for the long outage subset is higher than the short outage subset

Before running our test we looked at the observed accuracy difference between models we saw that the accuracy of the long outage dataset was 0.89 and the observed accuracy of the short outage dataset was 0.83. When we ran our permutation test we got a pval of 0.12. This tells us that it is pretty likely under random chance that we would see a difference this large. We fail to reject the null hypothesis that the accuracy of our two subsets are roughly the same, but we cannot be certain that the model is completely fair.

Code

In [23]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
from scipy.stats import pearsonr
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

sns.set_theme(style= 'whitegrid')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

In [24]:
```
#We first read in the file from excel and keep the columns relevant to our
def read_data(fp):
    """
    This is a function that reads in the Excel file with outages data.
    Only the appropriate rows/columns are taken in.
    fp = os.path.join('data', 'outage.xlsx')
    """

    #reads the excel file
    data = pd.read_excel(fp, header = 5, usecols = "B:T")

    # drop the first row with the descriptions
    data = data.drop([0])

    return data
```

In [25]:
```
fp = os.path.join('data', 'outage.xlsx')
data = read_data(fp).head()
```

In [26]:
```
# we then clean the dataset to only keep the variables necessary to our que
## helper function to convert into seasons
def season_helper(obs):
    """
    Helper functions converts month into
    respective season.
    """
    if obs >= 3 and obs <= 5:
        return 'Spring'
    elif obs >= 6 and obs <= 8:
        return 'Summer'
    elif obs >= 9 and obs <= 11:
        return 'Fall'
    elif pd.isnull(obs):
        return np.NaN
    else:
        return 'Winter'
def data_cleaning(data):
    """
    Takes in a table like the one produced by read_data(fp) and
    does the necessary stuff to clean the data set. This includes:
    -> combining the outage time/date columns into one
    -> combining the restoration time/date columns into one
    -> type casting the values in columns to their proper types
    """
    data_copy = data.copy(deep = True)

    # combine outage start time/date
    data_copy["OUTAGE.START.DATE"] = pd.to_datetime(
        data_copy["OUTAGE.START.DATE"])
    data_copy["OUTAGE.START.TIME"] = pd.to_timedelta(
        data_copy["OUTAGE.START.TIME"].astype(str))
    data_copy["OUTAGE.START"] = (data_copy["OUTAGE.START.DATE"] +
                                 data_copy["OUTAGE.START.TIME"])
    data_copy = data_copy.drop(
        columns = ["OUTAGE.START.DATE","OUTAGE.START.TIME"])

    # combine outage restoration time/date
    data_copy["OUTAGE.RESTORATION.DATE"] = pd.to_datetime(
        data_copy["OUTAGE.RESTORATION.DATE"])
    data_copy["OUTAGE.RESTORATION.TIME"] = pd.to_timedelta(
        data_copy["OUTAGE.RESTORATION.TIME"].astype(str))
    data_copy["OUTAGE.RESTORATION"] = (data_copy["OUTAGE.RESTORATION.DATE"]
                                       data_copy["OUTAGE.RESTORATION.TIME"]
    data_copy = data_copy.drop(
        columns = ["OUTAGE.RESTORATION.DATE", "OUTAGE.RESTORATION.TIME"])

    # cast values to 'correct' types + set index
    data_copy = data_copy.set_index('OBS')
    data_copy["YEAR"] = data_copy["YEAR"].astype(int)
    data_copy["OUTAGE.DURATION"] = data_copy["OUTAGE.DURATION"].astype(floa
    data_copy['DEMAND.LOSS.MW'] = data_copy['DEMAND.LOSS.MW'].astype(float)
    #add on a season column based on the month
    data_copy['SEASON'] = data_copy['MONTH'].apply(season_helper)
```

```
    data_copy = data_copy.reset_index().drop(columns = ['OBS'])

    return data_copy
```

In [27]:
```
# now let us read in the clean data
fp = os.path.join('data', 'outage.xlsx')
data = read_data(fp)
clean_data = data_cleaning(data)
```

In [28]:
```
# conditional function to determine if a power
# outage is classified as 'major'
def detect_major(row):
    if row["CUSTOMERS.AFFECTED"] > 50000:
        return 1
    elif row["DEMAND.LOSS.MW"] > 300:
        return 1
    else:
        return 0
```

In [29]:
```
# classify each power outage observation as a
# 0 or 1 in the major outage column
clean_data["Major_Outage"] = clean_data.apply(detect_major, axis = 1)
clean_data.head()
```

Out[29]:

	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVEL
0	2011	7.0	Minnesota	MN	MRO	East North Central	-0.3
1	2014	5.0	Minnesota	MN	MRO	East North Central	-0.1
2	2010	10.0	Minnesota	MN	MRO	East North Central	-1.5
3	2012	6.0	Minnesota	MN	MRO	East North Central	-0.1
4	2015	7.0	Minnesota	MN	MRO	East North Central	1.2

In [30]:
```
# 785/1534 (0.51) power outages were classified as being "Major"
print(clean_data["Major_Outage"].sum())
print(len(clean_data["Major_Outage"]))
```

```
785
1534
```

Baseline Model

In [31]:
```
# here we computed the accuracy score for baseline model by
# making a pipeline that first one hot encodes the necessary vars
# and then predicts using a decision tree classifier

def baseline_accuracy(data):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month']

    preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    X = features[preds]
    y = features["major_outage"]

    one_hot = ['u.s._state', 'climate.region', 'climate.category']
    ct = ColumnTransformer(transformers = [
        ('onehot', OneHotEncoder(handle_unknown = 'ignore'), one_hot)]
                           , remainder = 'passthrough')
    pl = Pipeline(steps = [('pp', ct),
                           ('tree', DecisionTreeClassifier(max_depth = 8))])

    pl.fit(X, y)
    preds = pl.predict(X)

    return pl.score(X, y)
```

In [32]:
```
baseline_accuracy(clean_data)
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-31-2ef320daefab>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[32]:
```
0.8599592114208022
```

Final Model

Adding in the Engineered Features

In [33]:
```
# logic behind binarizing cause category
print("# major power outages with cause severe:",
      clean_data[clean_data["CAUSE.CATEGORY"] == "severe weather"]["Major_O
print("# outages with cause severe:",
    len(clean_data[clean_data["CAUSE.CATEGORY"] == "severe weather"]["Major

print("# major power outages with other cause:",
    clean_data[clean_data["CAUSE.CATEGORY"] != "severe weather"]["Major_Out
print("# outages with other cause",
    len(clean_data[clean_data["CAUSE.CATEGORY"] != "severe weather"]["Major
```

```
# major power outages with cause severe: 662
# outages with cause severe: 763
# major power outages with other cause: 123
# outages with other cause 771
```

In [34]:
```
# a function that transfroms our engineered features
# and computes the accuracy score of the model on the whole dataset
# have not changed any parameters yet

def add_features_accuracy(data):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month', 'cause.category']

    preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'month', 'cause.category']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    def season_helper(df):
        """
        Helper functions converts month into
        respective season.
        """
        def helper(obs):
            if obs >= 3 and obs <= 5:
                return 'Spring'
            elif obs >= 6 and obs <= 8:
                return 'Summer'
            elif obs >= 9 and obs <= 11:
                return 'Fall'
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 'Winter'

        df["month"] = df["month"].apply(helper)
        return df

    def cause_helper(df):
        """
        Helper functions converts cause
        into binary variable.
        """
        def helper2(obs):
            if obs == 'severe weather':
                return 1
            elif pd.isnull(obs):
```

```
            return np.NaN
        else:
            return 0

    df["cause.category"] = df["cause.category"].apply(helper2)
    return df

X = features[preds]
y = features["major_outage"]

season_scale = ("convert season", Pipeline([
    ("pass to helper", FunctionTransformer(season_helper)),
    ("one_hot", OneHotEncoder(handle_unknown = 'ignore'))]), ["month"])

cause_scale = (
    "scale cause", FunctionTransformer(cause_helper),
    ["cause.category"])

one_hot = ['u.s._state', 'climate.region', 'climate.category']
ct = ColumnTransformer(
    transformers = [
        ('onehot', OneHotEncoder(handle_unknown = 'ignore'), one_hot),
                                        season_scale, cause_scale],
    remainder = 'passthrough')
pl = Pipeline(
    steps = [('pp', ct), ('tree', DecisionTreeClassifier(max_depth = 8)

pl.fit(X, y)
preds = pl.predict(X)

return pl.score(X, y)
```

In [35]:
```
add_features_accuracy(clean_data)
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-34-353f9eaad84e>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[35]:
```
0.9259007477906186
```

Testing Using Train/Test Split

In [36]:
```
# same thing as above adding train test split to see how our model generali
def trntst_split_accuracy(data, max_depth, min_leaf, min_split):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month', 'cause.category'

    preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'month', 'cause.category']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    def season_helper(df):
        """
        Helper functions converts month into
        respective season.
        """
        def helper(obs):
            if obs >= 3 and obs <= 5:
                return 'Spring'
            elif obs >= 6 and obs <= 8:
                return 'Summer'
            elif obs >= 9 and obs <= 11:
                return 'Fall'
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 'Winter'

        df["month"] = df["month"].apply(helper)
        return df

    def cause_helper(df):
        """
        Helper functions converts cause
        into binary variable.
        """
        def helper2(obs):
            if obs == 'severe weather':
                return 1
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 0
```

```
    df["cause.category"] = df["cause.category"].apply(helper2)
    return df

X = features[preds]
y = features["major_outage"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0

season_scale = ("convert season",
                Pipeline([
    ("pass to helper", FunctionTransformer(season_helper)),
    ("one_hot", OneHotEncoder(handle_unknown = 'ignore'))]),
                ["month"])

cause_scale = (
    "scale cause", FunctionTransformer(cause_helper),
    ["cause.category"])

one_hot = ['u.s._state', 'climate.region', 'climate.category']
ct = ColumnTransformer(transformers = [
    ('onehot', OneHotEncoder(handle_unknown = 'ignore'), one_hot),
                                        season_scale, cause_scale],
                        remainder = 'passthrough')
pl = Pipeline(
    steps = [('pp', ct), ('tree', DecisionTreeClassifier(
        max_depth = max_depth, min_samples_leaf = min_leaf,
        min_samples_split = min_split))])

pl.fit(X_train, y_train)

return pl.score(X_train, y_train), pl.score(X_test, y_test)
```

In [37]:
```
trntst_split_accuracy(clean_data, 4, 2, 2)
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-36-2b3108d33f4b>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[37]:
```
(0.8746355685131195, 0.8506787330316742)
```

Search for the Best Decision Classifier Parameters

In [38]:
```
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
```

In [39]:
```
# a function to do a grid search for the best model parameters
def gridsearch_cv(data):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month', 'cause.category']

    preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'month', 'cause.category']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    def season_helper(df):
        """
        Helper functions converts month into
        respective season.
        """
        def helper(obs):
            if obs >= 3 and obs <= 5:
                return 'Spring'
            elif obs >= 6 and obs <= 8:
                return 'Summer'
            elif obs >= 9 and obs <= 11:
                return 'Fall'
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 'Winter'

        df["month"] = df["month"].apply(helper)
        return df

    def cause_helper(df):
        """
        Helper functions converts cause
        into binary variable.
        """
        def helper2(obs):
            if obs == 'severe weather':
                return 1
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 0
```

```
    df["cause.category"] = df["cause.category"].apply(helper2)
    return df

X = features[preds]
y = features["major_outage"]

season_scale = ("convert season", Pipeline([
    ("pass to helper", FunctionTransformer(season_helper)),
    ("one_hot", OneHotEncoder(handle_unknown = 'ignore'))]), ["month"])

cause_scale = (
    "scale cause", FunctionTransformer(cause_helper),
    ["cause.category"])

one_hot = ['u.s._state', 'climate.region', 'climate.category']
ct = ColumnTransformer(
    transformers = [('onehot', OneHotEncoder(handle_unknown = 'ignore')
                                        season_scale, cause_scale],
    remainder = 'passthrough')

transformed_X = ct.fit(X).transform(X)

X_train, X_test, y_train, y_test = train_test_split(transformed_X, y, t

parameters = {"max_depth": [2, 3, 4, 5, 7, 10, 13, 15, 18, None],
          "min_samples_split": [2, 3, 5, 7, 10, 15, 20],
          "min_samples_leaf": [2, 3, 5, 7, 10, 15, 20]
          }
clf = GridSearchCV(DecisionTreeClassifier(), parameters, cv = 8)

clf.fit(X_train, y_train)

return clf.best_params_, plt.hist(clf.cv_results_["mean_test_score"], b
```

In [58]:
```
# our grid search gives us different results each time
# but the best parameters all seem to be in the ballpark of 3, 7, 20
params, plot = gridsearch_cv(clean_data)
params
```

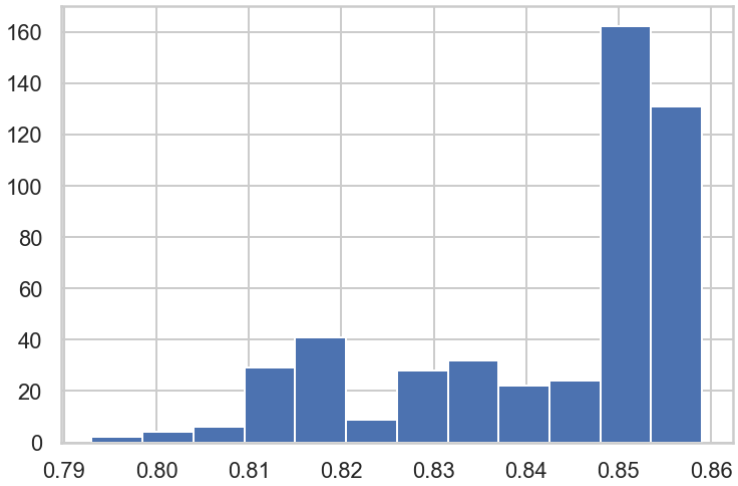
```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-39-4314c57f03b0>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[58]:
```
{'max_depth': 3, 'min_samples_leaf': 20, 'min_samples_split': 2}
```



In [59]:
```
# our model accuracy using the best parameters found above
trntst_split_accuracy(clean_data,
                      params["max_depth"],
                      params["min_samples_leaf"],
                     params["min_samples_split"])
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-36-2b3108d33f4b>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[59]:
```
(0.8561710398445093, 0.8710407239819005)
```

Testing KNeighbors Model as an Alternative

In [42]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [43]:
```
# a function that computes the baseline accuracy score
# using the kneighbors classifier

def kneighbors_baseline_accuracy(data, neighbors):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
               'anomaly.level', 'climate.category',
               'outage.duration', 'major_outage', 'month']

    preds = ['u.s._state', 'year', 'climate.region',
               'anomaly.level', 'climate.category',
               'outage.duration']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    X = features[preds]
    y = features["major_outage"]

    one_hot = ['u.s._state', 'climate.region', 'climate.category']
    ct = ColumnTransformer(
        transformers = [('onehot', OneHotEncoder(handle_unknown = 'ignore')
                         , remainder = 'passthrough')
    pl = Pipeline(
        steps = [('pp', ct),
                 ('kneighbors', KNeighborsClassifier(n_neighbors = neighbor

    pl.fit(X, y)
    preds = pl.predict(X)

    return pl.score(X, y)
```

In [44]:
```
kneighbors_baseline_accuracy(clean_data, 5)
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-43-d021cb0c4fc8>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[44]:
```
0.7987763426240653
```

In [45]:
```
# a function that computes the accuracy score
# using the kneighbors classifier with engineered features

def kneighbors_trtstsplit_accuracy(data, neighbors):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
               'anomaly.level', 'climate.category',
               'outage.duration', 'major_outage', 'month', 'cause.category']

    preds = ['u.s._state', 'year', 'climate.region',
               'anomaly.level', 'climate.category',
               'outage.duration', 'month', 'cause.category']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    def season_helper(df):
        """
        Helper functions converts month into
        respective season.
        """
        def helper(obs):
            if obs >= 3 and obs <= 5:
                return 'Spring'
            elif obs >= 6 and obs <= 8:
                return 'Summer'
            elif obs >= 9 and obs <= 11:
                return 'Fall'
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 'Winter'

        df["month"] = df["month"].apply(helper)
        return df

    def cause_helper(df):
        """
        Helper functions converts cause
        into binary variable.
        """
        def helper2(obs):
            if obs == 'severe weather':
                return 1
            elif pd.isnull(obs):
                return np.NaN
```

```
            else:
                return 0

        df["cause.category"] = df["cause.category"].apply(helper2)
        return df

    X = features[preds]
    y = features["major_outage"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0

    season_scale = ("convert season", Pipeline([
        ("pass to helper", FunctionTransformer(season_helper)),
        ("one_hot", OneHotEncoder(handle_unknown = 'ignore'))]), ["month"])

    cause_scale = (
        "scale cause", FunctionTransformer(cause_helper), ["cause.category"

    one_hot = ['u.s._state', 'climate.region', 'climate.category']
    ct = ColumnTransformer(
        transformers = [('onehot', OneHotEncoder(handle_unknown = 'ignore')
                                            season_scale, cause_scale],
        remainder = 'passthrough')

    pl = Pipeline(
        steps = [('pp', ct),
                 ('kneighbors', KNeighborsClassifier(n_neighbors = neighbor

    pl.fit(X_train, y_train)

    return pl.score(X_train, y_train), pl.score(X_test, y_test)
```

In [46]:
```
kneighbors_trtstsplit_accuracy(clean_data, 7)
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-45-cce9308513b2>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[46]:
```
(0.793002915451895, 0.7398190045248869)
```

In [47]:
```
# grid search with kneighbors to find the best number of neighbors
def gridsearch_cv_kneighbors(data):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month', 'cause.category']

    preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'month', 'cause.category']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    def season_helper(df):
        """
        Helper functions converts month into
        respective season.
        """
        def helper(obs):
            if obs >= 3 and obs <= 5:
                return 'Spring'
            elif obs >= 6 and obs <= 8:
                return 'Summer'
            elif obs >= 9 and obs <= 11:
                return 'Fall'
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 'Winter'

        df["month"] = df["month"].apply(helper)
        return df

    def cause_helper(df):
        """
        Helper functions converts cause
        into binary variable.
        """
        def helper2(obs):
            if obs == 'severe weather':
                return 1
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 0
```

```
    df["cause.category"] = df["cause.category"].apply(helper2)
    return df

X = features[preds]
y = features["major_outage"]

season_scale = ("convert season", Pipeline([
    ("pass to helper", FunctionTransformer(season_helper)),
    ("one_hot", OneHotEncoder(handle_unknown = 'ignore'))]), ["month"])

cause_scale = (
    "scale cause", FunctionTransformer(cause_helper),
    ["cause.category"])

one_hot = ['u.s._state', 'climate.region', 'climate.category']
ct = ColumnTransformer(
    transformers = [('onehot', OneHotEncoder(handle_unknown = 'ignore')
                                        season_scale, cause_scale],
    remainder = 'passthrough')

transformed_X = ct.fit(X).transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    transformed_X, y, test_size = 0.3)

parameters = {"n_neighbors": [10, 12, 14, 16, 18, 20, 22, 24,
                              26, 28, 30,32, 34, 36, 38, 40, 42,
                              44, 46, 48, 50, 52, 54, 56, 58, 60]}
clf = GridSearchCV(KNeighborsClassifier(), parameters, cv = 8)

clf.fit(X_train, y_train)

return clf.best_params_, plt.hist(clf.cv_results_["mean_test_score"], b
```

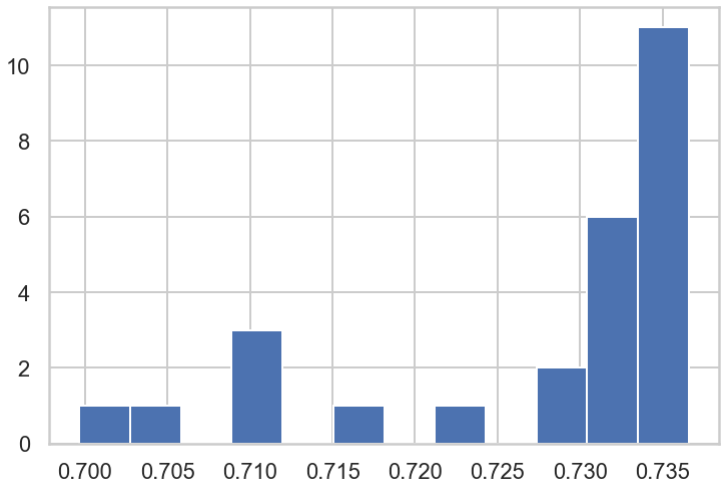
In [48]:
```
params, plot = gridsearch_cv_kneighbors(clean_data)
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-47-03fbc066dd0e>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```



In [49]:
```
kneighbors_trtstsplit_accuracy(clean_data, params["n_neighbors"])
```

```
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
/Users/devonromero/Library/Python/3.8/lib/python/site-packages/pandas/cor
e/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  return super().drop(
<ipython-input-45-cce9308513b2>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
```

Out[49]:
```
(0.7453838678328474, 0.7262443438914027)
```

Final Model:

In [50]:
```
# a function that returns our final pipeline
# with the engineered features
# and a decision tree classifier with our best found parameters

def final_pipeline(data):

    data_copy = data.copy(deep = True)

    feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month', 'cause.category']

    preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'month', 'cause.category']

    data_copy.columns = data_copy.columns.str.lower()

    features = data_copy[feature_cols]
    features.drop(
        features[features['anomaly.level'].isna() == True].index,
        inplace = True)
    features.drop(
        features[features['climate.region'].isna() == True].index,
        inplace = True)
    features.dropna(inplace = True)

    def season_helper(df):
        """
        Helper functions converts month into
        respective season.
        """
        def helper(obs):
            if obs >= 3 and obs <= 5:
                return 'Spring'
            elif obs >= 6 and obs <= 8:
                return 'Summer'
            elif obs >= 9 and obs <= 11:
                return 'Fall'
            elif pd.isnull(obs):
                return np.NaN
            else:
                return 'Winter'

        df["month"] = df["month"].apply(helper)
        return df

    def cause_helper(df):
        """
        Helper functions converts cause
        into binary variable.
        """
        def helper2(obs):
            if obs == 'severe weather':
                return 1
            elif pd.isnull(obs):
```

```
                return np.NaN
            else:
                return 0

        df["cause.category"] = df["cause.category"].apply(helper2)
        return df

    X = features[preds]
    y = features["major_outage"]

    season_scale = ("convert season", Pipeline([
        ("pass to helper", FunctionTransformer(season_helper)),
        ("one_hot", OneHotEncoder(handle_unknown = 'ignore'))]), ["month"])

    cause_scale = (
        "scale cause", FunctionTransformer(cause_helper),
        ["cause.category"])

    one_hot = ['u.s._state', 'climate.region', 'climate.category']

    ct = ColumnTransformer(
        transformers = [
            ('onehot', OneHotEncoder(handle_unknown = 'ignore'), one_hot),
                                            season_scale, cause_scale],
        remainder = 'passthrough')

    pl = Pipeline(
        steps = [('pp', ct), ('tree', DecisionTreeClassifier(max_depth = 3,
                                                min_samples_leaf = 7,
                                                min_samples_split = 20))])
    pl.fit(X, y)

    return pl
```

Fairness Evaluation

In [51]:
```
from sklearn import metrics
```

In [52]:
```
# we manually transformed the necessary variables so the result
# of our transformation was a dataframe
data_copy = clean_data.copy(deep = True)

feature_cols = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'major_outage', 'month', 'cause.category']

preds = ['u.s._state', 'year', 'climate.region',
                'anomaly.level', 'climate.category',
                'outage.duration', 'month', 'cause.category']

data_copy.columns = data_copy.columns.str.lower()

features = data_copy[feature_cols]
features.dropna(inplace = True)

def season_helper(obs):
    if obs >= 3 and obs <= 5:
        return 'Spring'
    elif obs >= 6 and obs <= 8:
        return 'Summer'
    elif obs >= 9 and obs <= 11:
        return 'Fall'
    elif pd.isnull(obs):
         return np.NaN
    else:
        return 'Winter'

def cause_helper(obs):
    if obs == 'severe weather':
        return 1
    elif pd.isnull(obs):
        return np.NaN
    else:
        return 0

X = features[preds]
y = features["major_outage"]
features['season'] = features['month'].apply(season_helper)
features['cause.category'] = features['cause.category'].apply(cause_helper)
# one hot season, state, climate.region, climate.category,
seasons = features.season.unique()
onehotseason = features['season'].apply(
    lambda x: pd.Series(x == seasons, index=seasons, dtype=float))
states = features['u.s._state'].unique()
regions = features['climate.region'].unique()
cats = features['climate.category'].unique()
onehotstate = features['u.s._state'].apply(
    lambda x: pd.Series(x == states, index=states, dtype=float))
onehotreg = features['climate.region'].apply(
    lambda x: pd.Series(x == regions, index=regions, dtype=float))
onehotcats = features['climate.category'].apply(
    lambda x: pd.Series(x == cats, index=cats, dtype=float))
new_features = pd.concat(
    [onehotseason,onehotstate,onehotreg,onehotcats,features], axis = 1)
```

```
new_features.drop(
    columns = ['climate.category', 'season', 'climate.region', 'u.s._state'
    inplace = True)
```

```
<ipython-input-52-f8e5e211252a>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features.dropna(inplace = True)
<ipython-input-52-f8e5e211252a>:40: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features['season'] = features['month'].apply(season_helper)
<ipython-input-52-f8e5e211252a>:41: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  features['cause.category'] = features['cause.category'].apply(cause_hel
per)
```

In [53]:
```
new_features.drop(
    columns = ['major_outage'], inplace = True)
new_features['anomaly.level'] = new_features['anomaly.level'].astype(float)
X_train, X_test, y_train, y_test = train_test_split(
    new_features, y, test_size = 0.3)
```

In [54]:
```
# permutation test
clf = DecisionTreeClassifier(
    max_depth = 3, min_samples_split = 20, min_samples_leaf = 7)
clf.fit(X_train, y_train)

preds = clf.predict(X_test)
results = X_test.copy()
results['preds'] = preds
results['tag'] = y_test
mean = new_features["outage.duration"].mean()
results['is_long'] = (
    results['outage.duration'] <= mean).replace(
    {True: 'short', False : 'long'})

obs = results.groupby('is_long').apply(
    lambda x: metrics.accuracy_score(x.tag, x.preds)).diff().iloc[-1]
metrs = []
for _ in range(100):
    s = (
        results[['is_long', 'preds', 'tag']]
        .assign(is_long=results.is_long.sample(
            frac=1.0, replace=False).reset_index(drop=True))
        .groupby('is_long')
        .apply(lambda x: metrics.accuracy_score(x.tag, x.preds))
        .diff()
        .iloc[-1]
    )

    metrs.append(s)
print(pd.Series(metrs <= obs).mean())
```

```
0.12
```