# Summary of Findings

## Introduction

For this project, I took a look at the outages dataset. This dataset has rows for each outage that occurred, and the columns represent the information on each outage, such as the date of the outage, duration, state, population, and much more.

The prediction problem that I investigated was the duration of the outage. I took a look at certain columns that are best in order to predict the length of a single outage, which is a regression model. In order to address this regression problem, I realized it would be possible to simply incorporate the quantitative values, but there are also categorical columns that help in the prediction, so they have to be put in quantitative inputs. Thus, I trained a baseline model with the set of features.

The target variables in order to predict outage duration I chose were month, US state, cause category, customers affected, and total customers. I thought these would be helpful in predicting the values as the location and time could affect weather that relate to power outages, and the length of the outage could relate to how it affects customers. The evaluation metric I used for my baseline model and final model was RMSE. RMSE is suitable for my prediction problem since my pipeline uses linear regression. The RMSE is useful in showing how much error there is from the actual dataset, thus can reflect whether the baseline or final model is better in prediction.

## Baseline Model

For my baseline model, I chose to model with a generic set of features from outage data. The columns that I saw that could possibly be helpful in predicting outage duration would be the outage's month, state, cause category, customers affected, and total customers. Since the categorical columns, the month, state, and cause do not have a particular order, being nominal, I chose to employ one-hot encoding on the pipeline and employing linear regression.

When I used train test split, I wanted to see how well the predictions were made, and firstly by finding the RMSE and R squared values. The RMSE of the baseline model I got was 5869, and R squared was 0.212. I think the RMSE reflects that my predictions are suitable given the values and range of outage durations, however, I knew that this could be improved.

## Final Model

In my final model, I decided to employ PCA on the categorical columns, and log-scaled the number of customers affected by the power outage. The reason for incorporating PCA, which is known to be Principal Component Analysis, is because it helps to remove and drop correlated features of the categorical variables in my dataset. It goes well with one-hot encoding. I thought one-hot encoding was helpful in predicting using categorical features, but the use of PCA will improve the predictions.

For the feature of number of customers affected, I log-scaled the column in the dataframe. In the code I added a demonstration, showing the association of the customers affected in relation to outage durations. The scatter plot showed that there seemed to be a pattern but it was extremely difficult to tell. When I log scaled the column and made the scatter plot, it was more easy to tell the relationship between customers affected and outage duration, thus, used log-scaling as the second engineered feature.

I used train test split on my final model to get the RMSE value of 4498 and R squared balue of 0.218. The RMSE value for this final model is lower than what I got from the baseline model, which shows improvement. The lower the RMSE value, we know that there is less error from the fitted dataset of the predicted values. Given this model, I went on to evaluate my model for 'fairness'.

### Fairness Evaluation

After developing my final model, I conducted a permutation test to evaluate the 'fairness'. In order to do so, I chose to evaluate the difference in RMSE values of the baseline model and the final model with a significance level of 0.05. My null hypothesis is that my model is fair, where the RMSE values of the baseline model and final model are roughly the same. My alternative hypothesis is that the predicted values from the final model has smaller RMSE values than the predicted values from the baseline model. The test statistic I used was difference in RMSE. I thought to compare predicted power outage durations, RMSE values reflect how well they were predicted.

I decided 500 simulations is an appropriate number to see if my model is fair. Within each simulation, I did train test split for the baseline model and final model, took their rmse values and got their differences to compare to the observed variable. The p-value I got was 0.22. Since the significance level is 0.05, my p-value was greater than the significance level. In conclusion, we fail to reject the null hypothesis. So we know that the final model seemed to be an improved model, but was not enough to be significantly different from my baseline model. The RMSE values of predictions made are roughly the same. Since my dataset of power outages was an observational study, there could potentially be confounding factors, and possibly columns that I dropped from the original dataframe that could have predicted outage durations better. With the predictions made from the baseline model and the final model, they are reasonable, but could be improved with more engineered features, or using other columns from the original dataset. Furthermore, these are predictions made from power outage data from 2000 to 2016, so it could be slightly outdated. Overall, my final model is fair, based on my permutation test.

# Code

In [1]:
```
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 figu
res
```

In [2]:
```
import sklearn.preprocessing as pp
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

In [3]:
```
filepath_outage = os.path.join('data', 'outage.xlsx')
outage_data = pd.read_excel(filepath_outage)
outage_data.head(10)
```

Out[3]:

| Unnamed: 48 | Unnamed: 49 | Unnamed: 50 | Unnamed: 51 | Unnamed: 52 | Unnamed: 53 | Un |
|---|---|---|---|---|---|---|
| NaN | NaN | NaN | NaN | NaN | NaN | |
| NaN | NaN | NaN | NaN | NaN | NaN | |
| NaN | NaN | NaN | NaN | NaN | NaN | |
| NaN | NaN | NaN | NaN | NaN | NaN | |
| POPPCT_UC | POPDEN_URBAN | POPDEN_UC | POPDEN_RURAL | AREAPCT_URBAN | AREAPCT_UC | PC |
| % | persons per square mile | persons per square mile | persons per square mile | % | % | |
| 15.28 | 2279 | 1700.5 | 18.2 | 2.14 | 0.6 | ! |
| 15.28 | 2279 | 1700.5 | 18.2 | 2.14 | 0.6 | ! |
| 15.28 | 2279 | 1700.5 | 18.2 | 2.14 | 0.6 | ! |
| 15.28 | 2279 | 1700.5 | 18.2 | 2.14 | 0.6 | ! |

In [4]:
```
# From Project03, getting the cleaned dataset of outage data, in a forma
t where we can decide the prediction model.
column_names_lst = outage_data.iloc[4].values.tolist()[2:]
outage_data_adjusted = outage_data.iloc[6:,2:]
outage_data_adjusted.reset_index(inplace=True, drop=True)
outage_data_adjusted.columns = column_names_lst
outage_data_adjusted.head()
```

Out[4]:

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

5 rows × 55 columns

## Baseline Model

We are taking a look at the duration of outage, so for the baseline model, I dropped the columns that seem to be irrelevant. Instead, I used features that seem relevant to predicting the length of the outage.

The features that seem to affect duration are the number of customers affected, total number of customers, the cause category of the outage, state, month, and population of the state.

In [5]:
```
selected_cols = outage_data_adjusted[['MONTH','U.S._STATE','CAUSE.CATEGO
RY','OUTAGE.DURATION','CUSTOMERS.AFFECTED','TOTAL.CUSTOMERS']]
selected_cols.head()
```

Out[5]:

| | MONTH | U.S._STATE | CAUSE.CATEGORY | OUTAGE.DURATION | CUSTOMERS.AFFECTED | TOTAL |
|---|---|---|---|---|---|---|
| **0** | 7 | Minnesota | severe weather | 3060 | 70000 | |
| **1** | 5 | Minnesota | intentional attack | 1 | NaN | |
| **2** | 10 | Minnesota | severe weather | 3000 | 70000 | |
| **3** | 6 | Minnesota | severe weather | 2550 | 68200 | |
| **4** | 7 | Minnesota | severe weather | 1740 | 250000 | |

Using this dataframe above would not work in order to create a baseline model. If there are NaN values, we cannot use sklearn models. The columns that have null values are 'MONTH', 'OUTAGE.DURATION', 'CUSTOMERS.AFFECTED'. Since 'OUTAGE.DURATION' is the column that we are predicting the values for, it seems like a bad idea to impute the null values. It could potentially affect the results of our predictions, so we will be removing those columns, and input the month and customers affected columns. This would not be too big of a problem since we have many rows to work with, we are only removing information on 58 outages.

In [6]:
```
selected_cols.isnull().sum()
```

Out[6]:
```
MONTH                   9
U.S._STATE              0
CAUSE.CATEGORY          0
OUTAGE.DURATION        58
CUSTOMERS.AFFECTED    443
TOTAL.CUSTOMERS         0
dtype: int64
```

In [7]:
```
selected_cols.shape[0]
```

Out[7]:
```
1534
```

In [8]:
```
selected_cols.dropna(subset=['OUTAGE.DURATION'],inplace=True)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [9]:
```
selected_cols.head()
```

Out[9]:

| | MONTH | U.S._STATE | CAUSE.CATEGORY | OUTAGE.DURATION | CUSTOMERS.AFFECTED | TOTAL |
|---|---|---|---|---|---|---|
| **0** | 7 | Minnesota | severe weather | 3060 | 70000 | |
| **1** | 5 | Minnesota | intentional attack | 1 | NaN | |
| **2** | 10 | Minnesota | severe weather | 3000 | 70000 | |
| **3** | 6 | Minnesota | severe weather | 2550 | 68200 | |
| **4** | 7 | Minnesota | severe weather | 1740 | 250000 | |

In [10]:
```
selected_cols.isnull().sum()
```

Out[10]:
```
MONTH                  0
U.S._STATE             0
CAUSE.CATEGORY         0
OUTAGE.DURATION        0
CUSTOMERS.AFFECTED   420
TOTAL.CUSTOMERS        0
dtype: int64
```

To impute the customers affected, I will be using imputation with distribution, since missing values are MAR, conditionally ignorable.

In [11]:
```
num_null = selected_cols['CUSTOMERS.AFFECTED'].isnull().sum()
```

In [12]:
```
# draw fill values from distribution
fill_vals = selected_cols['CUSTOMERS.AFFECTED'].dropna().sample(num_null
, replace=True)
```

In [13]:
```
# align index
fill_vals.index = selected_cols.loc[selected_cols['CUSTOMERS.AFFECTED'].
isnull()].index
```

In [14]:
```
# fill the values
selected_cols_filled = selected_cols.fillna({'CUSTOMERS.AFFECTED':fill_v
als.to_dict()})
```

Now that we have a proper dataset with no null values, I will train this baseline model with the features of selected columns. To predict, I am using one-hot encoding for the nominal columns: categorical features, which are month, state, cause category, and using the quantitative features, customers affected, total customers, and population as is.

Although the values for the month column are numbers, they indicate months and are not quantitative. They are nominal values.

In [15]:
```
selected_cols_filled.head()
```

Out[15]:

| | MONTH | U.S._STATE | CAUSE.CATEGORY | OUTAGE.DURATION | CUSTOMERS.AFFECTED | TOTAL |
|---|---|---|---|---|---|---|
| **0** | 7 | Minnesota | severe weather | 3060 | 70000.0 | |
| **1** | 5 | Minnesota | intentional attack | 1 | 0.0 | |
| **2** | 10 | Minnesota | severe weather | 3000 | 70000.0 | |
| **3** | 6 | Minnesota | severe weather | 2550 | 68200.0 | |
| **4** | 7 | Minnesota | severe weather | 1740 | 250000.0 | |

In [16]:
```
cat_feat = ['MONTH','U.S._STATE','CAUSE.CATEGORY']
# pipeline to one-hot encode
cat_transformer = Pipeline([
    ('one-hot', OneHotEncoder())
])
preproc = ColumnTransformer(transformers=[(
    'cat', cat_transformer, cat_feat
)])
pl = Pipeline(steps=[('preprocessor', preproc),('regressor', LinearRegre
ssion())])
pl.fit(selected_cols_filled.drop('OUTAGE.DURATION',axis=1),selected_cols
_filled['OUTAGE.DURATION'])
predictionspl = pl.predict(selected_cols_filled.drop('OUTAGE.DURATION',a
xis=1))
```

In [17]:
```
pl
```

Out[17]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('cat',
                                                  Pipeline(memory=None,
                                                           steps=[('one
-hot',

OneHotEncoder(categories='auto',

drop=None,

dtype=<class 'numpy.float64'>,

handle_unknown='error',

sparse=True))],
                                                           verbose=Fals
e),
                                                  ['MONTH', 'U.S._STAT
E',
                                                   'CAUSE.CATEGORY'])],
                                   verbose=False)),
                ('regressor',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jo
bs=None,
                                  normalize=False))],
         verbose=False)
```

In [18]:
```
# predictions from the baseline model
predictionspl
```

Out[18]:
```
array([3281.30391896,  310.8064098 , 5041.29410067, ...,  720.00034355,
        120.00805566,  120.00805566])
```

Now, with this baseline model, I am using train-test split in order to check the test scores. The reason for this is to check if my model overfits the data, and it is a goodness-of-fit test. Then we can retrieve the RMSE and $R^2$ values for how good the model is.

In [21]:
```
# using train-test split and checking rmse and r squared
X = selected_cols_filled.drop('OUTAGE.DURATION',axis=1)
y = selected_cols_filled['OUTAGE.DURATION']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
)
cat_feat = ['MONTH','U.S._STATE','CAUSE.CATEGORY']
cat_transformer = Pipeline([
    ('one-hot', OneHotEncoder(handle_unknown='ignore'))
])
preproc = ColumnTransformer(transformers=[(
    'cat', cat_transformer, cat_feat
)])

pl = Pipeline(steps=[('preprocessor', preproc),('regressor', LinearRegre
ssion())])
pl.fit(X_train, y_train)
preds = pl.predict(X_test)
rmse = np.sqrt(np.mean((preds - y_test)**2))
```

In [22]:
```
# rmse from the train test split of baseline model
rmse
```

Out[22]:
```
5869.019647305163
```

In [23]:
```
# r squared for the train test split of baseline model
r_squared_baseline = pl.score(selected_cols_filled.drop('OUTAGE.DURATIO
N',axis=1), selected_cols_filled['OUTAGE.DURATION'])
r_squared_baseline
```

Out[23]:
```
0.21193223891535118
```

## Final Model

In order to improve the baseline model, I will be engineering two new features, which are to use PCA on categorical data that drops correlated features, and secondly to log scale number of customers affected.

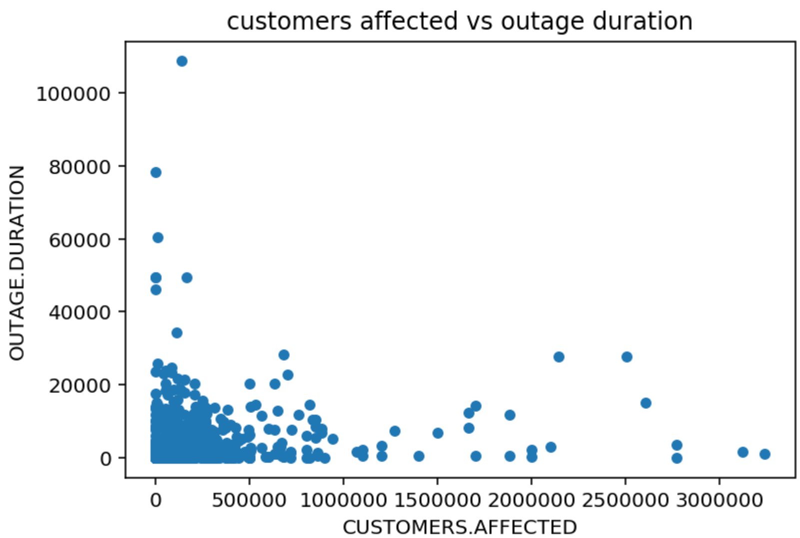
1. applying PCA to month, state, and cause category
2. log-scaling the number of customers affected, which is the column 'CUSTOMERS.AFFECTED'

In [24]:
```
# making copy of the baseline model dataframe
# this will be used to demonstrate reason for how the features are being
engineered
final_model_demo = selected_cols_filled.copy()
```

In [25]:
```
final_model_demo['OUTAGE.DURATION'] = final_model_demo['OUTAGE.DURATION'
].astype('int64')
```

In [26]:
```
# firstly, I am taking a look at how number of customers affected relate
s to outage duration
final_model_demo.plot(kind='scatter',x='CUSTOMERS.AFFECTED',y='OUTAGE.DU
RATION',title='customers affected vs outage duration')
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa09a967518>
```

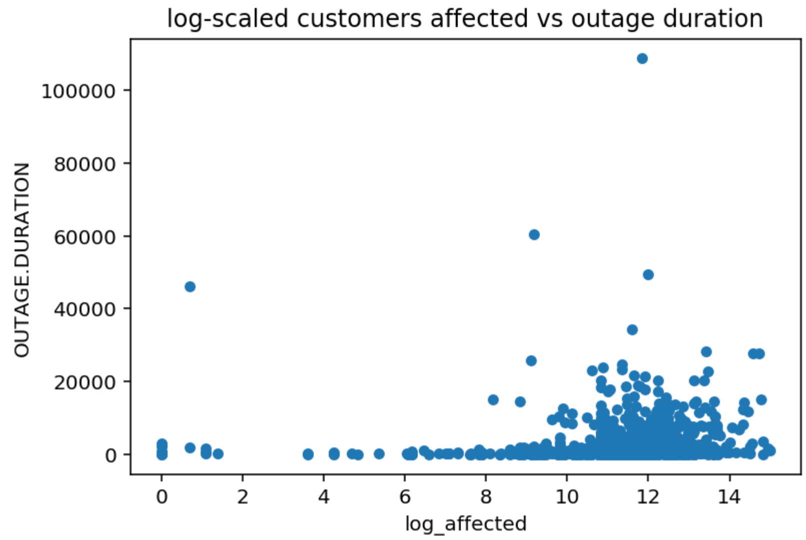



In [27]:
```
final_model_demo['log_affected'] = np.log(final_model_demo['CUSTOMERS.AF
FECTED'])
```

```
/opt/conda/lib/python3.7/site-packages/pandas/core/series.py:856: Runti
meWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [28]:
```
final_model_demo.plot(kind='scatter',x='log_affected',y='OUTAGE.DURATIO
N',title='log-scaled customers affected vs outage duration')
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa0934e25f8>




As we can see in the differences in the two scatter plots, it is better to log-scale the column of customers affected, since data points are more spread apart and better to interpret.

Now, I will be creating an sklearn ML-pipeline that will hopefully have better results to make predictions of the duration of power outages.

In [107]:
```
# making a copy that can be used for the final model
final_model = selected_cols_filled.copy()
```

In [108]:
```
final_model.head()
```

Out[108]:

| | MONTH | U.S._STATE | CAUSE.CATEGORY | OUTAGE.DURATION | CUSTOMERS.AFFECTED | TOTAL |
|---|---|---|---|---|---|---|
| **0** | 7 | Minnesota | severe weather | 3060 | 70000.0 | |
| **1** | 5 | Minnesota | intentional attack | 1 | 0.0 | |
| **2** | 10 | Minnesota | severe weather | 3000 | 70000.0 | |
| **3** | 6 | Minnesota | severe weather | 2550 | 68200.0 | |
| **4** | 7 | Minnesota | severe weather | 1740 | 250000.0 | |

In [57]:
```
# using train test split to get rmse and r squared value for the final m
odel that has 2 engineered features
X = final_model.drop('OUTAGE.DURATION',axis=1)
y = final_model['OUTAGE.DURATION']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
)
cat_feat_final_cols = ['MONTH','U.S._STATE','CAUSE.CATEGORY']
cat_feat_final = Pipeline([
    ('one-hot', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    # adding pca
    ('pca', PCA(svd_solver='full'))
])
num_feat_final_cols = ['CUSTOMERS.AFFECTED']
# using FunctionTransformer to log scale the number of customers affecte
d
num_feat_final = FunctionTransformer(lambda x: np.log(x))
ct_final = ColumnTransformer(transformers=[
    ('cat', cat_feat_final, cat_feat_final_cols),
    ('num', num_feat_final, num_feat_final_cols)
])
pl_final = Pipeline(steps=[('ct-feats', ct_final),('regressor',LinearReg
ression())])
pl_final.fit(X_train, y_train)
preds_final = pl_final.predict(X_test)
rmse_final = np.sqrt(np.mean((preds_final - y_test)**2))
```

In [58]:
```
rmse_final
```

Out[58]: 4497.7853807567435

In [59]:
```
r_squared_final = pl_final.score(final_model.drop('OUTAGE.DURATION',axis
=1), final_model['OUTAGE.DURATION'])
r_squared_final
```

Out[59]: 0.21798660515804458

## Fairness Evaluation

For inference analysis on results, I will be evaluating the 'fairness', by taking a look at the RMSE values of both baseline and final models. I will be using a permutation test and a significance level of 0.05. The reason why I chose RMSE is because it measures the error of the model in predicting the outage durations. The whole purpose of the permutation test is to compare the RMSE of the subset of predictions from baseline model to the RMSE of the predictions from the final model and see if the improved model really makes a difference, has less error in predicting outage durations. I will use train test split and perform 500 repetitions.

Null Hypothesis: The final model is fair, RMSE values from the predicted values of the baseline model and final model are roughly the same.

Alternative Hypothesis: Predicted values from the final model has smaller RMSE values than the predicted values from the from the baseline model.

Test Statistic: Difference in RMSE

In [60]:
```
# predicted values from baseline model
preds[:20]
```

Out[60]:
```
array([-1189.31490376,  3436.68553804,  4108.74099873, -1056.32338854,
        4947.54392307,  3751.45497254,   730.77732174,  2560.40878839,
        5003.96320052,  5051.27772308,  4237.56488218,  1279.59036569,
        2222.96493761,   765.6530333 ,  -126.31440551,   907.1904003 ,
        3464.80863417,  3180.43782827,   976.34995042,  -356.88916266])
```

In [61]:
```
# predicted values from the final model
preds_final[:20]
```

Out[61]:
```
array([ 3437.91027241,  2474.05186405, -1789.34417311,  1569.84267804,
        1771.26155664,   120.71870742,  3558.73483959,  1013.89357241,
        4217.09767891,  1349.82846795,  6350.920502  ,  3145.3274493 ,
        2942.48131624, -3298.33840976,  1153.75590279, 15611.72590701,
        6427.39970304,   955.22677973,  3172.20668547,   804.17439326])
```

In [78]:
```
rmse_baselinemodel = np.sqrt(np.mean((preds - y_test)**2))
rmse_baselinemodel
```

Out[78]:
```
5133.7975644597445
```

In [85]:
```
rmse_finalmodel = np.sqrt(np.mean((preds_final2 - y_test)**2))
rmse_finalmodel
```

Out[85]:
```
4497.7853807567435
```

In [86]:
```
# observed test statistic: taken from the predictions made earlier
difference_rmse_observed = np.abs(rmse_baselinemodel - rmse_finalmodel)
difference_rmse_observed
```

Out[86]:
```
636.012183703001
```

In [94]:
```
# Permutation test - performing 500 simulations to get predicted valued
 of outage durations
n_repetitions = 500

# setting up lists to take in the rmse values from baseline and final mo
dels
rmse_baseline = []
rmse_final = []


for i in range(n_repetitions):
    # firstly setting up train test split
    X = final_model.drop('OUTAGE.DURATION',axis=1)
    y = final_model['OUTAGE.DURATION']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.25)
    # this is for the baseline model
    cat_feat = ['MONTH','U.S._STATE','CAUSE.CATEGORY']
    cat_transformer = Pipeline([
        ('one-hot', OneHotEncoder(handle_unknown='ignore'))
    ])
    preproc = ColumnTransformer(transformers=[(
        'cat', cat_transformer, cat_feat
    )])

    pl_baseline = Pipeline(steps=[('preprocessor', preproc),('regressor'
, LinearRegression())])
    pl_baseline.fit(X_train, y_train)
    preds_baseline = pl_baseline.predict(X_test)
    one_rmse_baseline = np.sqrt(np.mean((preds_baseline - y_test)**2))
    rmse_baseline.append(one_rmse_baseline)

    # this is for the final model - engineering features
    cat_feat_final_cols = ['MONTH','U.S._STATE','CAUSE.CATEGORY']
    cat_feat_final = Pipeline([
        ('one-hot', OneHotEncoder(handle_unknown='ignore', sparse=False
)),
        ('pca', PCA(svd_solver='full'))
    ])
    num_feat_final_cols = ['CUSTOMERS.AFFECTED']
    num_feat_final = FunctionTransformer(lambda x: np.log(x))
    ct_final = ColumnTransformer(transformers=[
        ('cat', cat_feat_final, cat_feat_final_cols),
        ('num', num_feat_final, num_feat_final_cols)
    ])
    pl_final = Pipeline(steps=[('ct-feats', ct_final),('regressor',Linea
rRegression())])
    pl_final.fit(X_train, y_train)
    preds_final = pl_final.predict(X_test)
    one_rmse_final = np.sqrt(np.mean((preds_final - y_test)**2))
    rmse_final.append(one_rmse_final)
```

In [98]:
```
rmse_baseline[:5]
```

Out[98]:
```
[5460.24360631834,
 4801.123768950495,
 4112.222049648121,
 5915.80339410297,
 6448.244838515151]
```

In [99]:
```
rmse_final[:5]
```

Out[99]:
```
[5459.456941015309,
 4835.2670004019055,
 4113.262334852791,
 7888637008507.801,
 6450.838185141994]
```

In [101]:
```
# now that we have rmse values, taking difference in rmse and saving it
 into an array
diff_rmse = np.abs(np.array(rmse_baseline) - np.array(rmse_final))
```

In [104]:
```
diff_rmse[:5]
```

Out[104]:
```
array([7.86665303e-01, 3.41432315e+01, 1.04028520e+00, 7.88863700e+12,
       2.59334663e+00])
```

In [105]:
```
# now that we have our simulated data, we are calculating p-value,
# comparing our calculated test statistics to observed difference in rms
e
p_value = np.count_nonzero(diff_rmse >= difference_rmse_observed) / n_re
petitions
p_value
```

Out[105]:
```
0.22
```

In [ ]:
```
# larger significance level so we fail to reject the null hypothesis
```

As a result of the permutation test, we found the p-value to be 0.22. I chose the appropriate significance level to be 0.05. Because my p-value is greater than the significance level, we know that we fail to reject the null hypothesis.

So in conclusion, the distribution of predicted values from the baseline model and the final model are approximately the same.