Predicting Building Energy Use

Project Understanding and Technical Overview

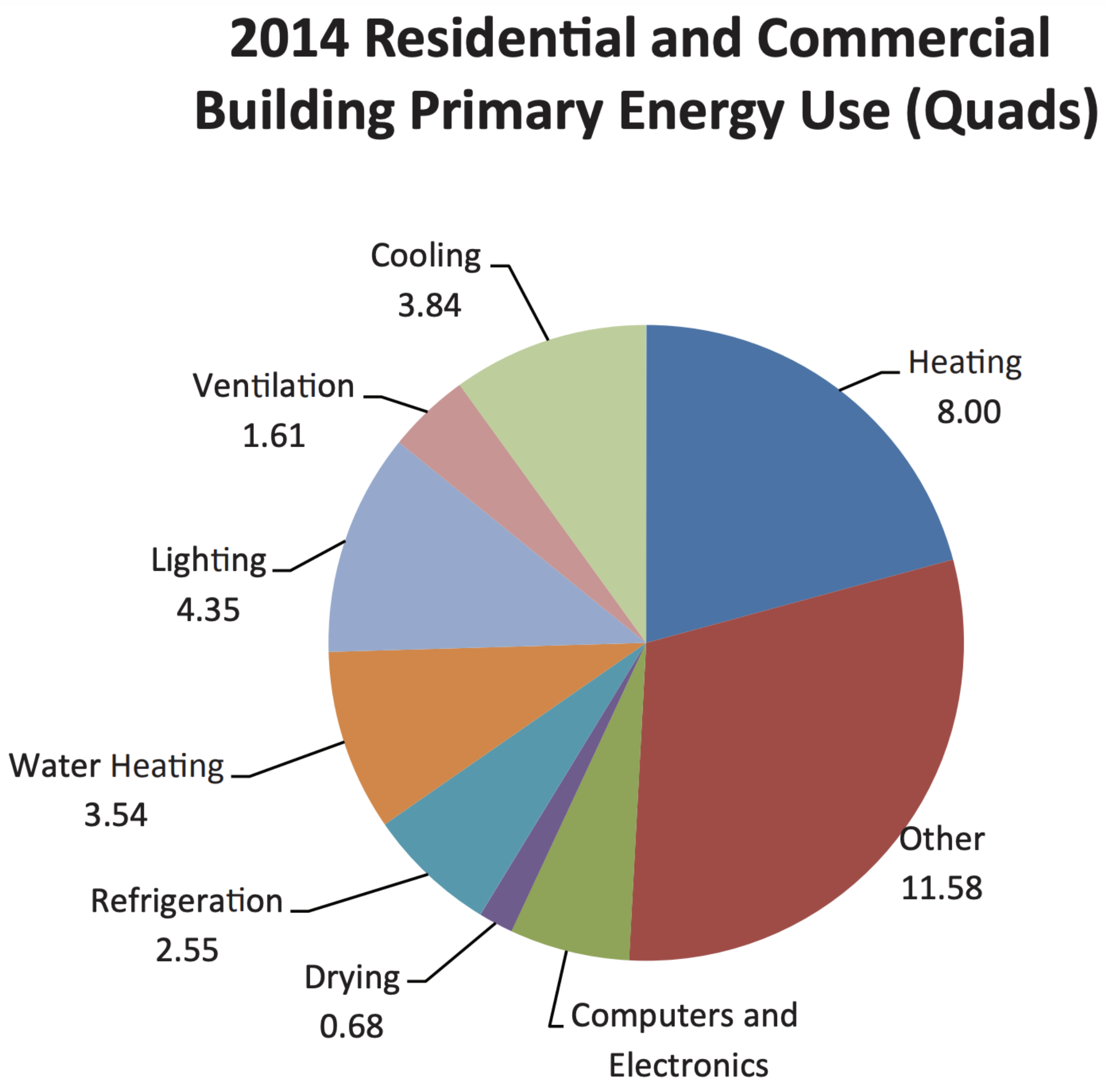
This notebook covers the modeling performed with the NYC Open Data dataset to predict annual electricity usage for buildings. The model uses building size, primary use and energy efficiency information to make a prediction on the annual electricity use. Identifying the important features in determining electricity use could help to inform energy credit programs that can incentivize energy abusers.

The first simple model uses linear regression which produces an RMSE of 940 MWh. The final ensemble model is a Random Forest Regressor which produces an RMSE of 614 MWh. The average annual electricity usage for a building >50,000 sqft in NYC is 975 MWh +/- 1,700 MWh.

Business Understanding

Buildings account for 76% of all electricity use in the United States. This translates to 40% of total greenhouse gas emissions from baseline power (coal and oil). Over half of a buildings energy draw comes from environment control:

- lighting
- cooling
- heating
- ventilation
- refrigeration



New York State Energy Plan:

"The State Energy Plan is a comprehensive roadmap to build a clean, resilient, and affordable energy system for all New Yorkers. The Plan coordinates every State agency and authority that touches energy to advance the REV agenda, unleashing groundbreaking regulatory reform to integrate clean energy into the core of our power grid, redesigning programs to unlock private capital, and actively deploying innovative energy solutions across the State's own public facilities and operations." (EnergyPlan.NY.gov)

The plan has the following goals for the state:

- 40% reduction in greenhouse gas emissions from 1990 levels
- 50% of energy generation from renewable energy sources
- 600 trillion Btu increase in statewide energy efficiency

Being able to predict the annual electricity usage of a building based on it's size, primary use and energy efficiency rating would allow for better municipal planning and could help with identifying buildings that over-consume. This would be ideal for planning city sustainability projects as energy consumption could be predicted based on a forecasted change in energy efficiency of buildings. Based on required energy reductions predicted by this model the state could tailor sustainability programs to incentivize building managers to increase energy efficiency (reduce energy consumption).

Data Understanding

Target Variable - Annual Electricity Usage:

Energy use is typically tracked in kWh (killowatt hours) or MWh (megawatt hours = 1,000 killowatt hours) using electric meters installed by a utility company. NYC's Local Law 87 requires that buildings over 50,000 sqft undergo energy audits which include accurate, annual reporting of energy usage. Over 30,000 buildings within New York City fall under this law.

Building Energy Efficiency:

A building's EnergyStar Score is a relative benchmark to other buildings of similar qualities. The higher the score the more energy efficient that building is. An EnergyStar Score of 50 is the median energy efficiency. In the state of New York, buildings above an EnergyStar Score of 70 are eligible for rebates through special state run programs.

Primary Use:

The primary use of a building is a general building classification. The top 5 classifications:

1. Multifamily Housing
2. Office
3. K-12 School
4. College/University
5. Hotel

Data Exploration

Below the necessary imports are loaded.

```
In [ ]: import pandas as pd
        import numpy as np
        import itertools

        import matplotlib.pyplot as plt
        import matplotlib.dates as mdates
        from matplotlib.dates import DateFormatter
        import seaborn as sns

        from sklearn.impute import SimpleImputer
        from sklearn.model_selection import train_test_split, cross_val_score, GridSearc
        from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
        from sklearn.linear_model import LinearRegression
        from sklearn.feature_selection import RFE
        from sklearn.metrics import mean_squared_error, mean_squared_log_error
        from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

        from statsmodels.formula.api import ols
        from statsmodels.api import qqplot

        from scipy import stats
```

The data comes from NYC Open Data. The target column is the 'Electricity Use - Grid Purchase (kWh)'. Here, the dataset is read in and the used columns are filtered for.

```
In [ ]: df = pd.read_csv('/Users/evanjays/Desktop/Programming/Flatiron/CAPSTONE/data_big
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/IPython/core/interacti
veshell.py:3145: DtypeWarning: Columns (14,31,32,111,113,127,206,236,237,238) ha
ve mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

```
In [ ]: dfx = df.filter(['Property Id','Primary Property Type - Self Selected','Largest
```

Dropping Duplicate Buildings in dataset based on property id

```
In [ ]: dfx.drop_duplicates('Property Id',inplace=True)
```

```
In [ ]: dfx.drop('Property Id',axis=1,inplace=True)
        print('Starting Buildings Count:',df.shape[0],'|','Starting Columns Count:',df.s
        print('Buildings Count:',dfx.shape[0],'|','Feature Count:',dfx.shape[1]-1)
```

```
Starting Buildings Count: 34686 | Starting Columns Count: 256
Buildings Count: 22681 | Feature Count: 5
```

Data Cleaning

Label encoding the EnergyStar Certification Eligibility column. Reminder - buildings who meet the eligibility have an EnergyStar Score of 70 or greater (in the top 30% of energy performance relative to similar buildings).

```
In [ ]: def energystar(x):
            if x == 'No':
                return 0
            else:
                return 1

        dfx['ENERGY STAR Certification - Eligibility'] = dfx['ENERGY STAR Certification
```

Renaming building primary uses to all be in English.

```
In [ ]: def use_cleanup(x):
            if x == 'Collège/Université':
                return 'College/University'
            elif x == 'Immeuble à logements multiples':
                return 'Multifamily Housing'
            elif x == 'Bureau':
                return 'Office'
            elif x == 'Résidence/dortoir':
                return 'Residence Hall/Dormitory'
            elif x == 'Autre':
                return 'Other'
            elif x == 'Lieu de culte':
                return 'Worship Facility'
            elif x == 'Autre - Éducation':
                return 'Other - Education'
            elif x == 'Hôtel':
                return 'Hotel'
            elif x == 'Résidence pour personnes âgées':
                return 'Senior Care Community'
            elif x == 'Entreposage libre-service':
                return 'Self-Storage Facility'
            elif x == 'Entrepôt non réfrigéré':
                return 'Non-Refrigerated Warehouse'
            elif x == 'Supermarché/épicerie':
                return 'Supermarket/Grocery Store'
            elif x == 'Propriété à usage mixte':
                return 'Mixed Use Property'
            elif x == 'Commerce de détail':
                return 'Retail Store'
            elif x == 'Entrepôt réfrigéré':
                return 'Refrigerated Warehouse'
            elif x == 'Centre des arts de la scène':
                return 'Performing Arts'
            elif x == 'Patinoire/piste de curling':
                return 'Ice/Curling Rink'
            elif x == 'Autre - Divertissement/Rassemblement public':
                return 'Other - Entertainment/Public Assembly'
            elif x == 'Hôpital (soins médicaux et chirurgicaux)':
                return 'Hospital (General Medical & Surgical)'
            elif x == 'Usine de fabrication/industrie':
                return 'Manufacturing/Industrial Plant'
            else:
                return x

        df['Primary Property Type - Self Selected'] = df['Primary Property Type - Self S
        dfx['Primary Property Type - Self Selected'] = dfx['Primary Property Type - Self
```

For this model we only want to look at buildings that have active electricity usage so any building with no usage is removed. Also, outliers more than 3.5 standard deviations from mean for electricity use are also removed.

```
In [ ]: before_drop_count = dfx.shape[0]
        dfx = dfx[dfx['Electricity Use - Grid Purchase (kWh)'] > 0]
        dfx = dfx[(np.abs(stats.zscore(dfx['Electricity Use - Grid Purchase (kWh)'])) <
```

```
In [ ]: print('Building Count Before Drop:',before_drop_count)
```

```
print('Building Count After Drop:',dfx.shape[0])
print('% Drop:',round((1-dfx.shape[0]/before_drop_count)*100,1))
```

```
Building Count Before Drop: 22681
Building Count After Drop: 20610
% Drop: 9.1
```

Train/Test Split

In [ ]:
```
X_train,X_test,y_train,y_test = train_test_split(dfx.drop(['Electricity Use - Gr
```

Imputing EnergyStar Scores

In [ ]:
```
X_train.reset_index(inplace=True,drop=True)
X_test.reset_index(inplace=True,drop=True)
y_train.reset_index(inplace=True,drop=True)
y_test.reset_index(inplace=True,drop=True)

df_train = pd.concat([X_train,y_train],axis=1)
df_test = pd.concat([X_test,y_test],axis=1)

si = SimpleImputer()

df_train['ENERGY STAR Score'] = si.fit_transform(df_train[['ENERGY STAR Score']]
df_train['ENERGY STAR Score'] = si.transform(df_train[['ENERGY STAR Score']])

df_train.dropna(inplace=True)
df_test.dropna(inplace=True)

X_train = df_train.drop('Electricity Use - Grid Purchase (kWh)',axis=1)
X_test = df_test.drop('Electricity Use - Grid Purchase (kWh)',axis=1)
y_train = df_train['Electricity Use - Grid Purchase (kWh)']
y_test = df_test['Electricity Use - Grid Purchase (kWh)']
```

One Hot Encoding the Primary Use Input Variable. There are 86 unique building classifiers used in the training data.

In [ ]:
```
ohe = OneHotEncoder(handle_unknown='ignore')

ohe_train_array = ohe.fit_transform(X_train[['Primary Property Type - Self Selec
ohe_train_df = pd.DataFrame(ohe_train_array.todense(),columns=[name[3:] for name

ohe_test_array = ohe.transform(X_test[['Primary Property Type - Self Selected']]
ohe_test_df = pd.DataFrame(ohe_test_array.todense(),columns=[name[3:] for name i

X_train.reset_index(inplace=True,drop=True)
X_train_ohe = pd.concat([X_train,ohe_train_df],axis=1)
X_train_ohe.drop(['Primary Property Type - Self Selected'],axis=1,inplace=True)

X_test.reset_index(inplace=True,drop=True)
X_test_ohe = pd.concat([X_test,ohe_test_df],axis=1)
X_test_ohe.drop(['Primary Property Type - Self Selected'],axis=1,inplace=True)
```

In [ ]:
```
print('Number of Features Pre OHE:',X_train.shape[1])
print('Number of Features Post OHE:',X_train_ohe.shape[1])
```

```
Number of Features Pre OHE: 5
Number of Features Post OHE: 68
```

Modeling

Simple Model

For the initial model

```
In [ ]: lr = LinearRegression()
        lr.fit(X_train_ohe,y_train)
        lr.score(X_test_ohe,y_test)
```

```
Out[ ]: 0.6682202068525991
```

```
In [ ]: np.sqrt(mean_squared_error(y_test,lr.predict(X_test_ohe)))
```

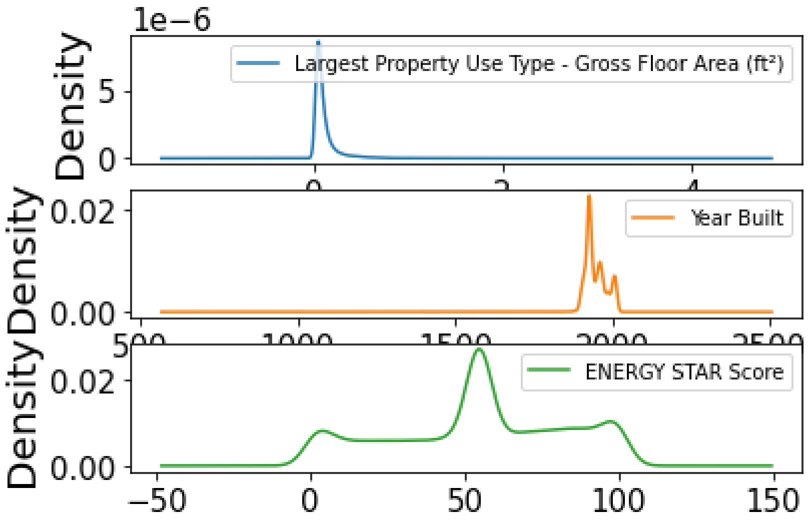
```
Out[ ]: 940211.2359823538
```

Simple Model Evaluation

Checking Normal Distribution of Input Variables

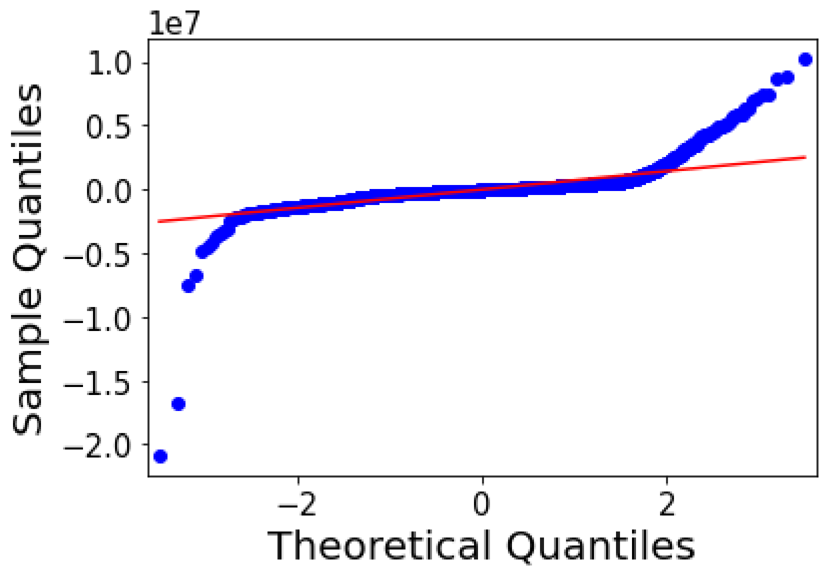
```
In [ ]: X_train_ohe.filter(['Largest Property Use Type - Gross Floor Area (ft²)','Year B
```

```
Out[ ]: array([<AxesSubplot:ylabel='Density'>, <AxesSubplot:ylabel='Density'>,
               <AxesSubplot:ylabel='Density'>], dtype=object)
```



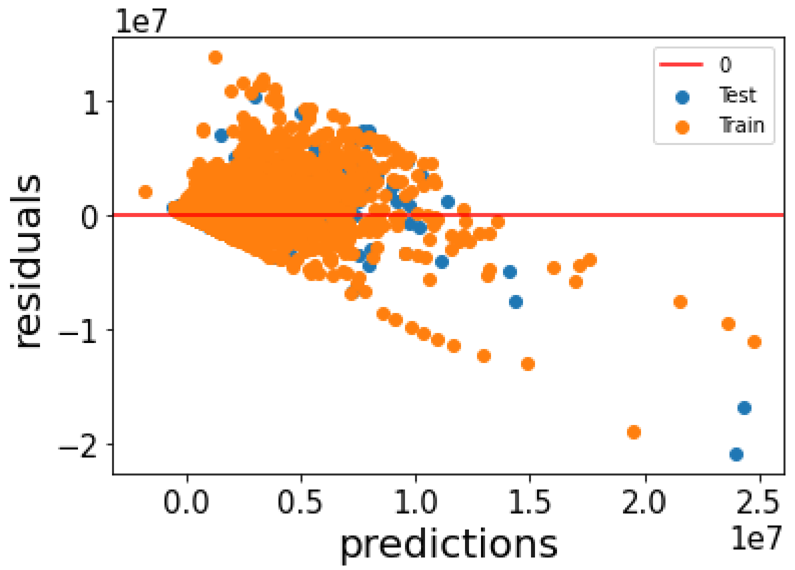
Checking Normal Distribution of Residuals

```
In [ ]: qqplot(y_test-lr.predict(X_test_ohe),line='r')
        plt.show()
```



Checking for Heteroskedasticity of Residuals

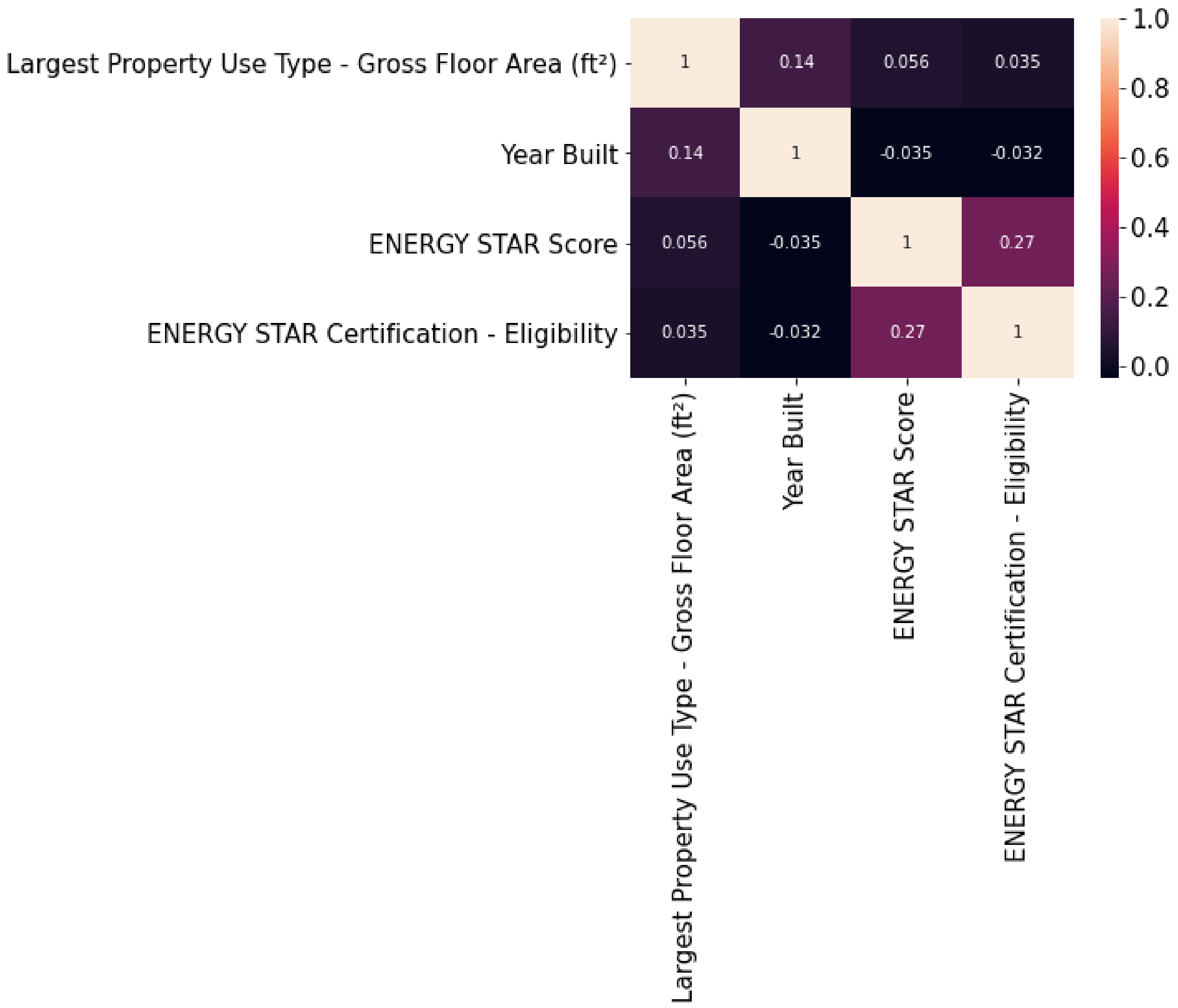
In [ ]:
```
plt.scatter(lr.predict(X_test_ohe), y_test-lr.predict(X_test_ohe), label='Test')
plt.scatter(lr.predict(X_train_ohe), y_train-lr.predict(X_train_ohe), label='Tra

plt.axhline(y=0, color = 'red', label = '0')
plt.xlabel('predictions')
plt.ylabel('residuals')
plt.legend()
plt.show()
```



Checking Multicollinearity of Input Variables

In [ ]:
```
sns.heatmap(X_train.corr(),annot=True)
```

Out[ ]:
```
<AxesSubplot:>
```



Based on the above assumption checks, linear regression does not appear to be the best choice.

Final Model

```
In [ ]: rfr = RandomForestRegressor()
```

```
In [ ]: rfe = RFE(rfr,n_features_to_select=30)
        rfe.fit(X_train_ohe,y_train)
```

```
Out[ ]: RFE(estimator=RandomForestRegressor(), n_features_to_select=30)
```

```
In [ ]: # final_list = ['Largest Property Use Type - Gross Floor Area (ft²)', 'Year Buil
```

```
In [ ]: keep_list = [(k,v) for k,v in zip(X_train_ohe.columns,rfe.support_)]
        keep_string = ''
        for k,v in keep_list:
            if v:
                keep_string += k + '+'
        keep_string = keep_string[:-1]
        final_list = [ x[0] for x in keep_list if x[1] ]
```

```
In [ ]: rfr.fit(X_train_ohe.filter(final_list),y_train)
```

Out[ ]:
```
RandomForestRegressor()
```

In [ ]:
```
params = {'n_estimators':np.linspace(10,400,num=10,dtype=int),
          'max_depth': np.linspace(1,40,num=5,dtype=int),
          'min_samples_split': np.linspace(2,10,num=5,dtype=int)}
gs = GridSearchCV(rfr,params)
gs.fit(X_train_ohe,y_train)
gs.best_params_
```

Out[ ]:
```
{'max_depth': 40, 'min_samples_split': 10, 'n_estimators': 356}
```

In [ ]:
```
best_params = {'max_depth': 36, 'min_samples_split': 10, 'n_estimators': 356}
```

In [ ]:
```
rfr = RandomForestRegressor(max_depth=best_params['max_depth'],min_samples_split
rfr.fit(X_train_ohe.filter(final_list),y_train)
```

Out[ ]:
```
RandomForestRegressor(max_depth=36, min_samples_split=10, n_estimators=356)
```

In [ ]:
```
rfr.score(X_test_ohe.filter(final_list),y_test)
```

Out[ ]:
```
0.8585274156171433
```

Cross Validation to check for overfitting.

In [ ]:
```
cross_val_score(rfr,X_train_ohe.filter(final_list),y_train)
```

Out[ ]:
```
array([0.78969241, 0.80263724, 0.75755329, 0.80460961, 0.78111326])
```

In [ ]:
```
y_hat_train = rfr.predict(X_train_ohe.filter(final_list))
print('RMSE Train Data:',round(np.sqrt(mean_squared_error(y_train,y_hat_train)))
y_hat_test = rfr.predict(X_test_ohe.filter(final_list))
print('RMSE Test Data:',round(np.sqrt(mean_squared_error(y_test,y_hat_test))))
```

```
RMSE Train Data: 482473.0
RMSE Test Data: 613955.0
```

In [ ]:
```
dfx[['Electricity Use - Grid Purchase (kWh)']].describe()
```

Out[ ]:

	Electricity Use - Grid Purchase (kWh)
count	2.061000e+04
mean	9.752697e+05
std	1.704314e+06
min	1.200000e+01
25%	2.164623e+05
50%	3.993497e+05
75%	9.134812e+05
max	1.536148e+07

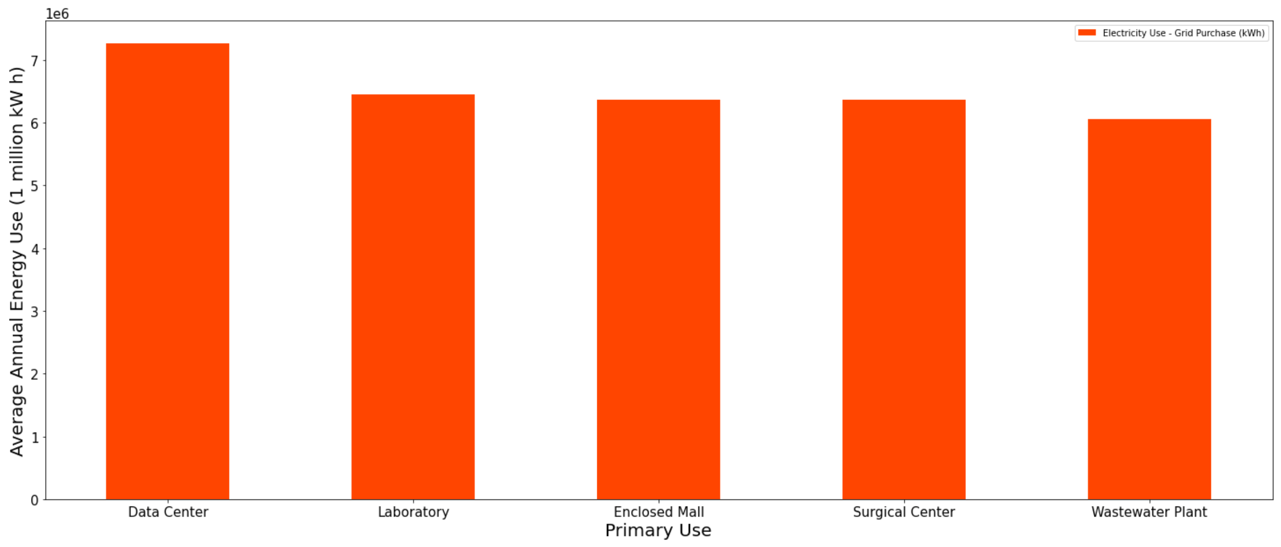
Findings

```
In [ ]: features = [(x,y) for x,y in zip(rfr.feature_importances_,final_list)]
        features.sort(key=lambda x: x[0],reverse=True)
```

The final model is able to predict a building's annual electricity usage from the grid to within 595,614 kW h. The standard deviation of the dataset is 1,630,000 kW h.

Visualizing Top Energy Users

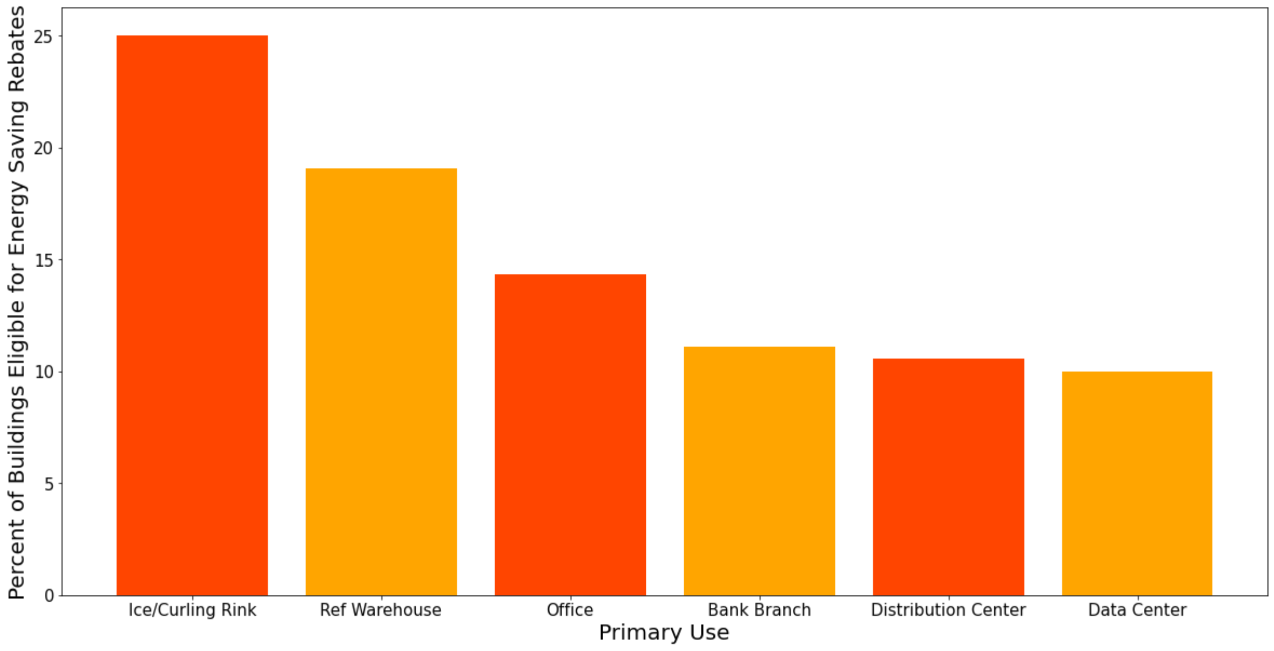
```
In [ ]: plt.rcParams['xtick.labelsize'] = 15
        plt.rcParams['ytick.labelsize'] = 15
        plt.rcParams['axes.labelsize'] = 20

        def change_ent(x):
            if x == 'Hospital (General Medical & Surgical)':
                return 'Hospital'
            elif x == 'Refrigerated Warehouse':
                return 'Ref Warehouse'
            elif x == 'Non-Refrigerated Warehouse':
                return 'Non-Ref Warehouse'
            elif x == 'Wastewater Treatment Plant':
                return 'Wastewater Plant'
            elif x == 'Ambulatory Surgical Center':
                return 'Surgical Center'
            else:
                return x

        dfx['Primary Property Type - Self Selected'] = dfx['Primary Property Type - Self

        dfx.groupby('Primary Property Type - Self Selected').mean()[['Electricity Use -
        plt.show()
```



The top energy users fall into categories that typically have large square footage buildings and contain energy intensive processes.

```
In [ ]: efficiency_list = []
        # for x in dfx.value_counts('Primary Property Type - Self Selected').index:
        #     temp_df = pd.DataFrame()
        #     temp_df = dfx[dfx['Primary Property Type - Self Selected'] == x]
        #     efficiency_list.append((x,temp_df['ENERGY STAR Certification - Eligibility

        df['Primary Property Type - Self Selected'] = df['Primary Property Type - Self S
```

```
for x in df.value_counts('Primary Property Type - Self Selected').index:
    temp_df = pd.DataFrame()
    temp_df = df[df['Primary Property Type - Self Selected'] == x].drop_duplicat
    temp_df['ENERGY STAR Certification - Eligibility'] = temp_df['ENERGY STAR Ce
    efficiency_list.append((x,temp_df['ENERGY STAR Certification - Eligibility']

efficiency_list.sort(key=lambda x: x[1],reverse=True)

final_effic_list = efficiency_list[:6]

# final_effic_list = [('Office', 14.878892733564014), ('Bank Branch', 14.2857142

fig,ax = plt.subplots(figsize=(20,10))

ax.bar(x=[x[0] for x in final_effic_list],height=[y[1] for y in final_effic_list
ax.set(xlabel="Primary Use")
ax.set(ylabel="Percent of Buildings Eligible for Energy Saving Rebates")
plt.xticks(rotation=0)
plt.show()
```



The above graph shows the top performing primary uses by percentage of buildings that qualify for Energy Credits (70th percentile for energy efficiency). Ambulatory Surgical Centers and Wastewater Treatment Plants, while being in the top for energy consumption are not present in the top performing primary uses.

Recommendations

- Focus energy efficiency incentive programs on high electricity use buildings that have the lowest percentage of energy credit qualifying buildings (Prisons, Surgical Centers, Wastewater Treatment Plants).
- Forecast Citywide Annual Electricity Usage by making predictions using a targeted energy efficiency metric. For example predict electricity usage for buildings using a goal energy

efficiency metric (i.e. 5% increase in energy efficiency) to determine potential efficiency savings.

Future Research

- Use sub-metering data in the model to include the exact energy use (HVAC,Lighting,Computers). For example, understanding which energy use by primary use of the building could help with creating more targeted incentive programs and recommended solutions.
- Incorporate weather data to show effects of seasonality on energy use