Project 2

May 24, 2021

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

from sklearn.linear_model import LinearRegression, Lasso, Ridge # loading models
from sklearn.model_selection import GridSearchCV, cross_val_score # loading␣
 ↪gridserach cross validation package
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
 ↪mean_squared_log_error
from sklearn.metrics import median_absolute_error, r2_score # loading␣
 ↪evaluation metrics


from sklearn.tree import DecisionTreeRegressor # decision tree regressor
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor # random forest regressor
from sklearn.ensemble import GradientBoostingRegressor # gradient boosting␣
 ↪regressor

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
apt = pd.read_excel("apartment-1.xlsx")
of =pd.read_excel("office-1.xlsx")
```

[3]:
```
of.head()
```

[3]:
```
   month  day  hour  Drybulb Temperature  Wetbulb Temperature  \
0      1    1     1                 16.0            13.705041
1      1    1     2                 15.6            13.758291
2      1    1     3                 15.1            13.595604
3      1    1     4                 14.8            13.512457
4      1    1     5                 14.4            13.227824

   Relative Humidity  Wind Speed  Wind Direction  Solar Radiation  \
0                 78         0.5             190                0
1                 82         2.1             120                0
```

```
2                 85          2.1             120                  0
3                 87          2.1             140                  0
4                 88          1.0             150                  0

   Sky Clearness  Total Electric Demand  HVAC Electric Demand
0            0.0            33566.91208              8.440094
1            0.0            33566.91208              8.440094
2            0.0            33566.91208              8.440094
3            0.0            33566.91208              8.440094
4            0.0            33566.91208              8.440094
```

1 Question 1:

Assuming Electricity Demand to be average hourly demand Compare the total and HVAC electricity consumption of the two buildings on a monthly and annual basis. (a bar chart is preferred)

[4]:
```
# comparing Total Electric Demand between Office and apartment (monthly basis):
```

[5]:
```
apt_monthly = apt.groupby('month')[['Total Electric Demand', 'HVAC Electric␣
 ↪Demand']].sum()
of_monthly = of.groupby('month')[['Total Electric Demand', 'HVAC Electric␣
 ↪Demand']].sum()
```

[6]:
```
of_monthly.columns = ["Office Total Electric Consumption", "Office HVAC␣
 ↪Electric Consumption"]
apt_monthly.columns =["Apartment Total Electric Consumption", "Apartment HVAC␣
 ↪Electric Consumption"]
```

[7]:
```
compar1 = pd.concat([apt_monthly.iloc[:,0], of_monthly.iloc[:,0]], axis =1)
compar1.reset_index(inplace =True)
compar1.head()
```

[7]:
```
   month  Apartment Total Electric Consumption  \
0      1                          1.682073e+07
1      2                          1.519335e+07
2      3                          1.674008e+07
3      4                          1.595244e+07
4      5                          1.650184e+07

   Office Total Electric Consumption
0                       4.487559e+07
1                       4.068455e+07
2                       4.684226e+07
3                       4.251098e+07
4                       4.508912e+07
```

[8]:
```
compar2 = pd.concat([apt_monthly.iloc[:,1], of_monthly.iloc[:,1]], axis =1)
compar2.reset_index(inplace =True)
compar2.head()
```

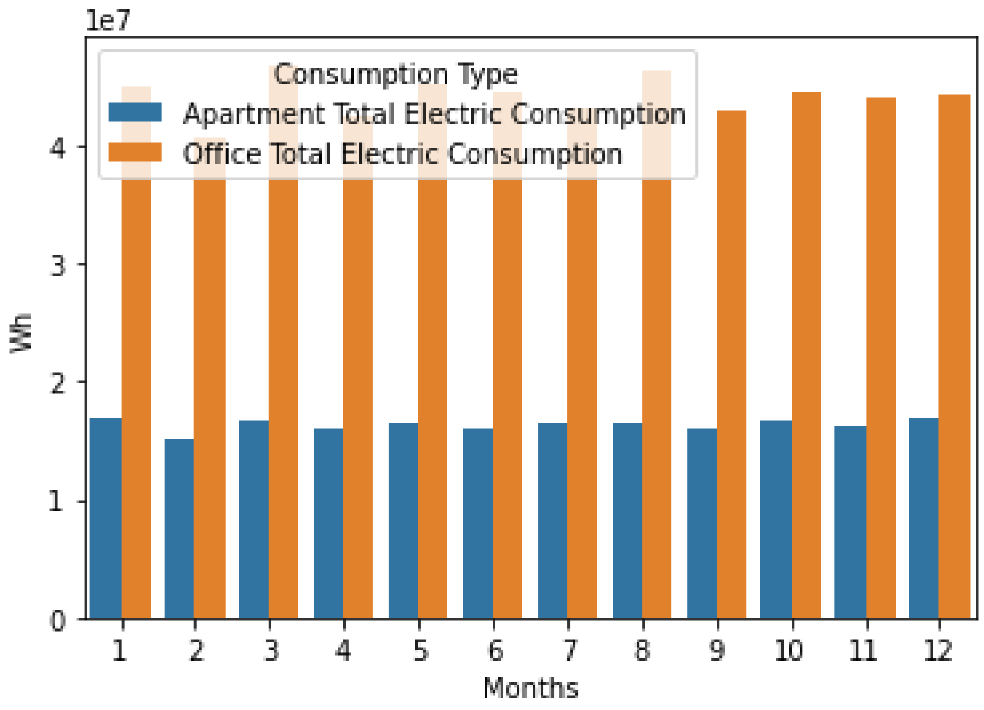
[8]:
```
   month  Apartment HVAC Electric Consumption  \
0      1                         3.332422e+06
1      2                         4.128330e+06
2      3                         6.112963e+06
3      4                         8.679104e+06
4      5                         1.424524e+07

   Office HVAC Electric Consumption
0                      9.684091e+06
1                      1.068902e+07
2                      1.605553e+07
3                      1.948842e+07
4                      2.902400e+07
```

[9]:
```
compar1 = pd.melt(compar1,id_vars = ['month'], value_vars = compar1.columns[1:].
 ↪tolist())
compar1.columns = ['Months', 'Consumption Type', 'Wh']
compar1.head()
```

[9]:
```
   Months                      Consumption Type            Wh
0       1  Apartment Total Electric Consumption  1.682073e+07
1       2  Apartment Total Electric Consumption  1.519335e+07
2       3  Apartment Total Electric Consumption  1.674008e+07
3       4  Apartment Total Electric Consumption  1.595244e+07
4       5  Apartment Total Electric Consumption  1.650184e+07
```

[10]:
```
import seaborn as sns
sns.barplot(x= 'Months', y ='Wh', hue ='Consumption Type' ,data =compar1)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4be22a9370>
```

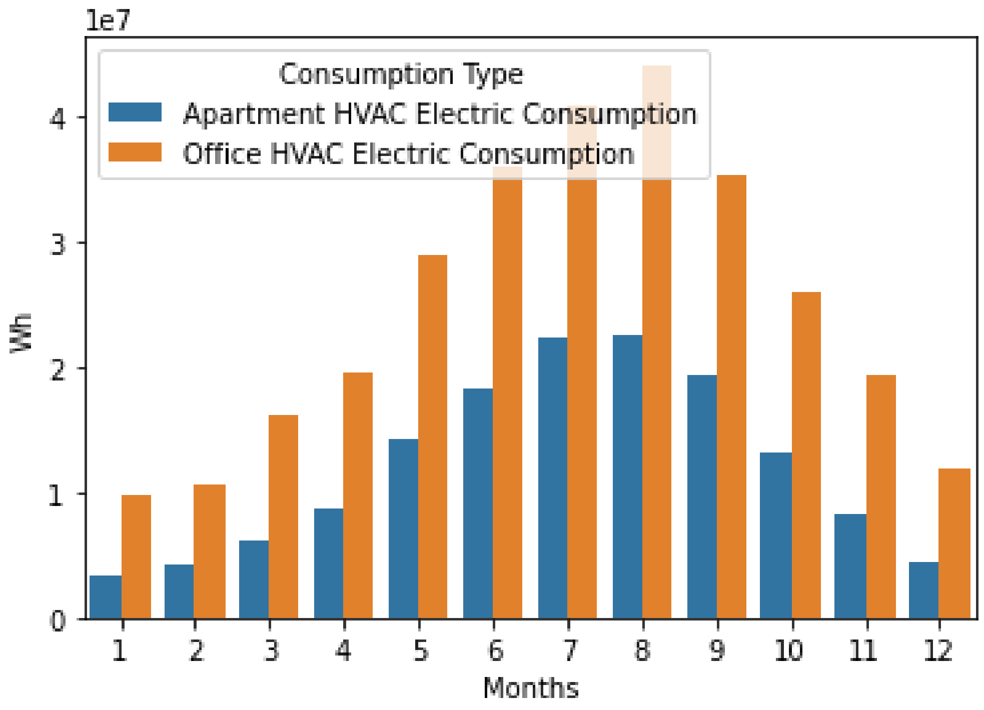


```
[11]: compar2 = pd.melt(compar2,id_vars = ['month'], value_vars = compar2.columns[1:].
       ↪tolist())
      compar2.columns = ['Months', 'Consumption Type', 'Wh']
      compar2.head()
```

```
[11]:    Months                    Consumption Type            Wh
      0       1  Apartment HVAC Electric Consumption  3.332422e+06
      1       2  Apartment HVAC Electric Consumption  4.128330e+06
      2       3  Apartment HVAC Electric Consumption  6.112963e+06
      3       4  Apartment HVAC Electric Consumption  8.679104e+06
      4       5  Apartment HVAC Electric Consumption  1.424524e+07
```

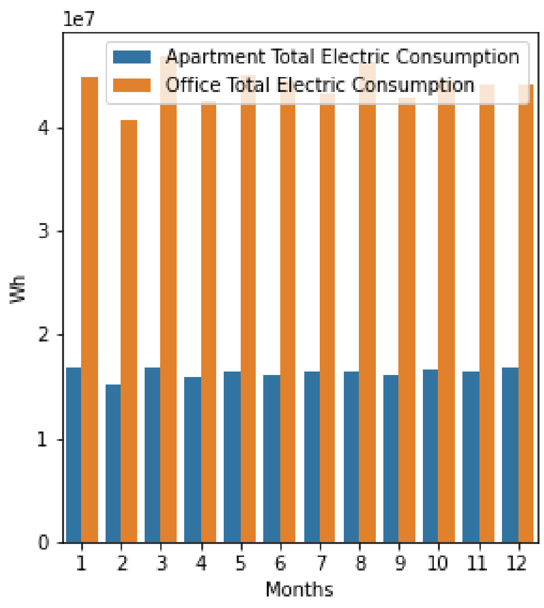
```
[12]: sns.barplot(x= 'Months', y ='Wh', hue ='Consumption Type' ,data =compar2)
```

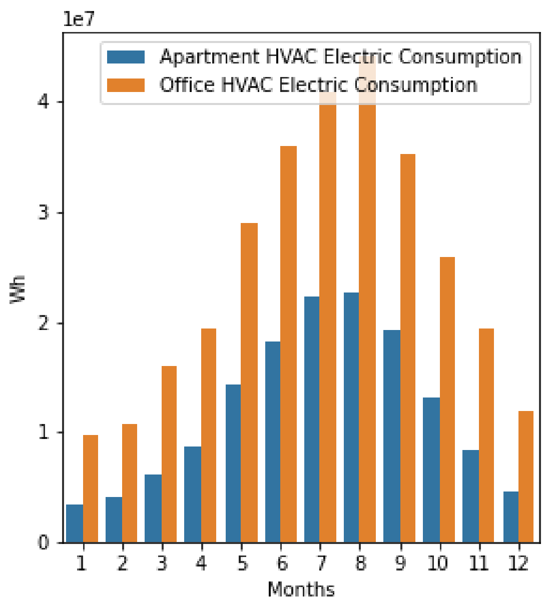
```
[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4be26a44c0>
```



[13]:
```
fig = plt.figure(figsize =(10,5))
plt.subplot(121)
sns.barplot(x= 'Months', y ='Wh', hue ='Consumption Type' ,data =compar1)
plt.legend(loc='best')
plt.subplot(122)
sns.barplot(x= 'Months', y ='Wh', hue ='Consumption Type' ,data =compar2)
plt.legend(loc='best')
```

[13]:
```
<matplotlib.legend.Legend at 0x7f4bdff5cbb0>
```





2

Notwithstanding the steady electric consumption of both builidngs, on monthly basis, we observe that Apartment's Total electric consumption is almost 3 times lower than Office's Total Consumption.

However, the HVAC electricity consumption increases steadily and symmetrically for both types of buildings until it reaches its peak in July for both buildings. Nonethless, Apartment's HVAC electric consumption is always lower than 10MWh or more, relative to the Office building.

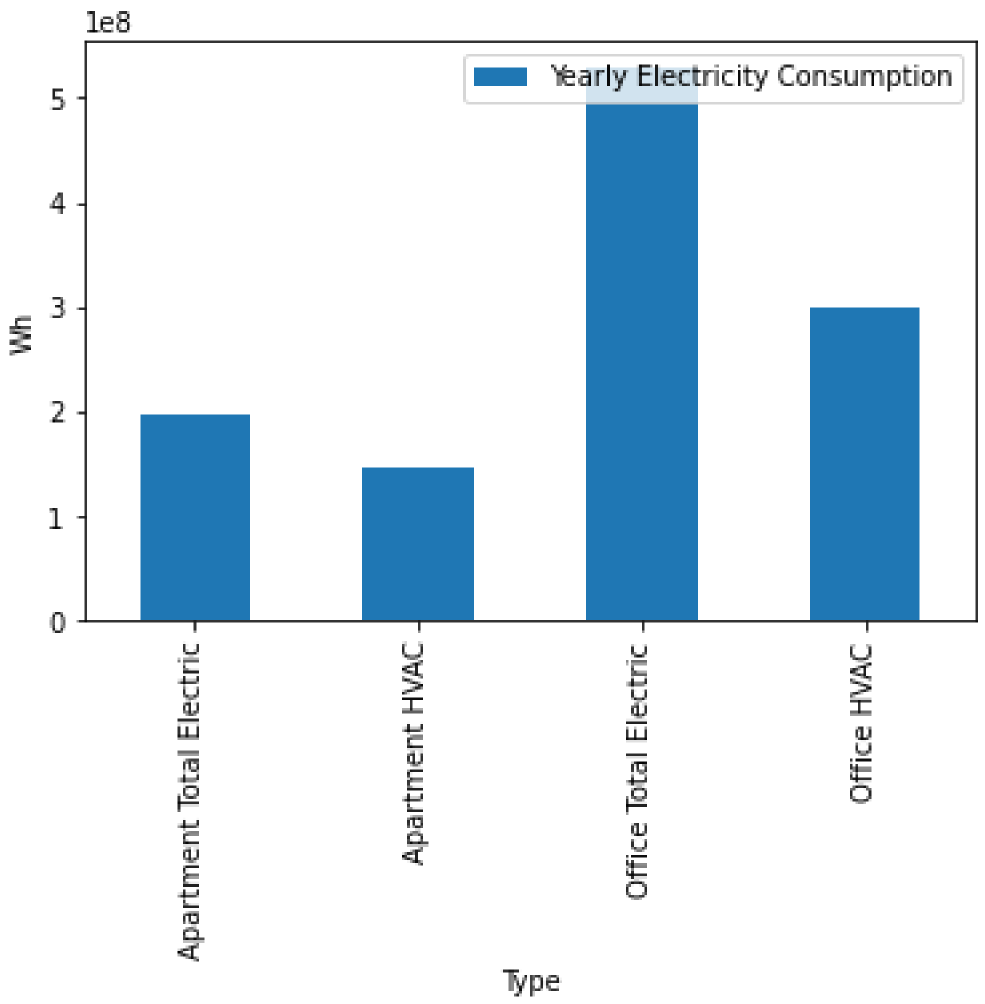
```
[14]: apt_monthly = apt.groupby('month')[['Total Electric Demand', 'HVAC Electric␣
       ↪Demand']].sum().sum()
      of_monthly = of.groupby('month')[['Total Electric Demand', 'HVAC Electric␣
       ↪Demand']].sum().sum()
```

```
[15]: yearly = pd.concat([apt_monthly, of_monthly], axis =0).to_frame()
      yearly.columns= ['Yearly Electricity Consumption']
      yearly.index = ['Apartment Total Electric', 'Apartment HVAC', 'Office Total␣
       ↪Electric', 'Office HVAC']
```

```
[16]: yearly.reset_index(inplace =True)
      yearly.columns = ['Type','Yearly Electricity Consumption']
```

```
[17]: yearly.plot(x = 'Type',y = 'Yearly Electricity Consumption', kind ='bar')
      plt.ylabel("Wh")
```

```
[17]: Text(0, 0.5, 'Wh')
```



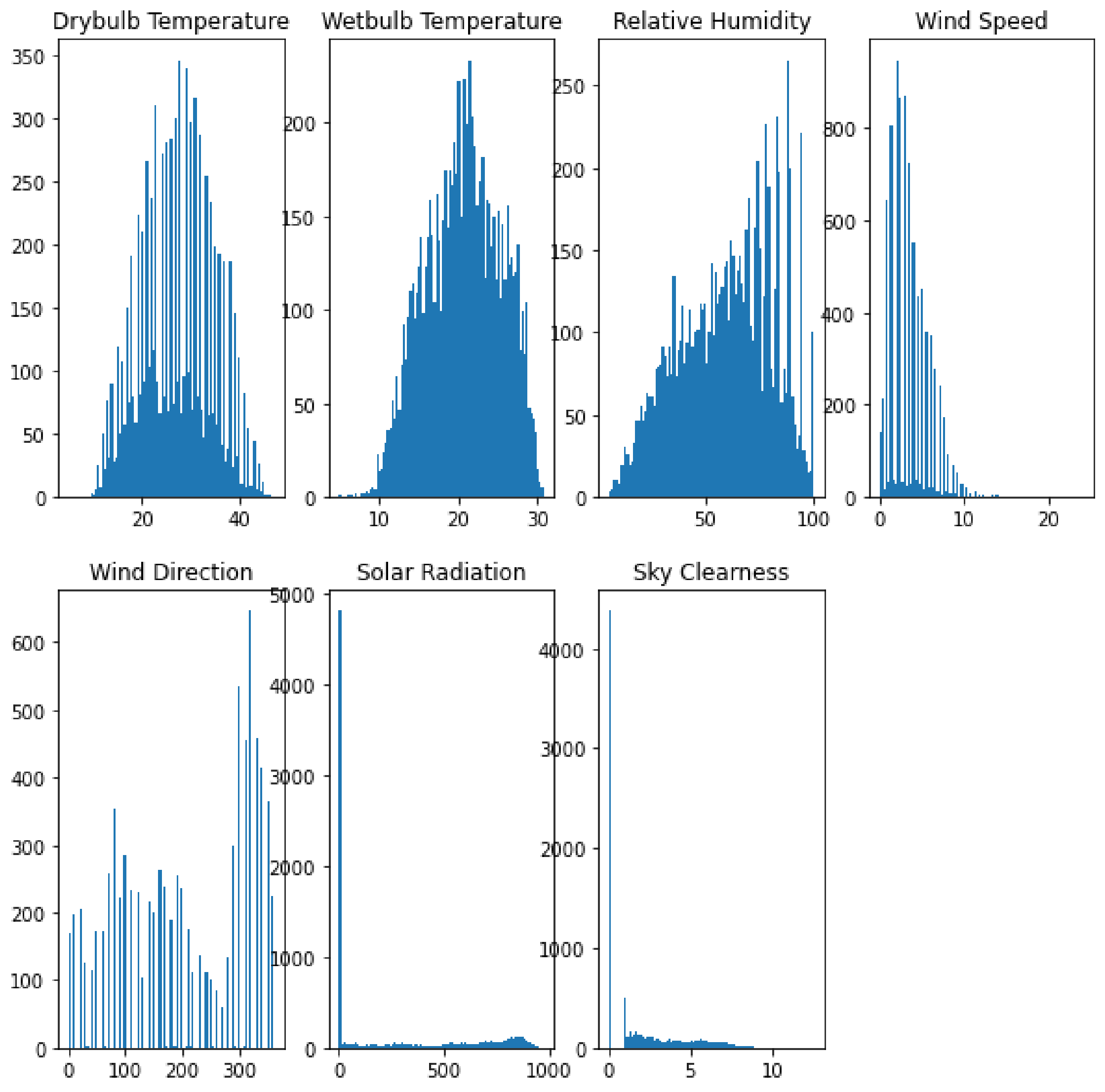
3

We see that Total electric consumption for entire year of Offices is much higher than apartments. The Total HVAC electric consumption is always less than total for both types.

4 Question 2:

Plot and describe the distribution of the weather data.

[18]:
```
figure = plt.figure(figsize =(10,10))
weather_cols = apt.columns[3:-2].tolist()
for i, v in enumerate(weather_cols):
    figure.add_subplot(2,4,i+1)
    plt.hist(apt[v], bins = int(np.sqrt(len(apt))))
    plt.title(v)
```



```
[19]: apt[apt.columns[3:-2]].skew()
```

```
[19]: Drybulb Temperature    0.045843
      Wetbulb Temperature   -0.127939
      Relative Humidity     -0.283523
      Wind Speed             1.062260
      Wind Direction        -0.208958
      Solar Radiation        0.792666
      Sky Clearness          1.171593
      dtype: float64
```

5

-> The Dry bulb temperature, wet bulb temperature, and Wind Direction are fairly symmetrical

-> Relative Humidity is also fairly symmetrical as its skewness is between -0.5 and 0.5

-> Wind Speed and Sky Clearness, however, are highly skewed positively as their skewness is greater than 1

-> Solar Radiation is moderately skewed positively as its skewness is between 0.5 and 1

[20]:
```
# from scipy.stats import norm
# from scipy.stats import t
# mean, var = norm.fit(apt['Wind Speed'])
# plt.scatter(apt['Wind Speed'], norm.pdf(apt['Wind Speed'], mean, var), label␣
 ↪= 'normal distribution') # normal distribution
# plt.scatter(apt['Wind Speed'], t.pdf(apt['Wind Speed'], mean, var), label ='t␣
 ↪distribution') # t distribution
# sns.distplot(apt['Wind Speed'], hist= False, label = 'original distribution')
# plt.legend()
```

6 Question 3:

Report the correlations between weather conditions and HVAC demand for each building.

[21]:
```
apt.corr()['HVAC Electric Demand'][3:-2]
```

[21]:
```
Drybulb Temperature     0.935065
Wetbulb Temperature     0.847445
Relative Humidity      -0.495784
Wind Speed              0.301006
Wind Direction          0.224513
Solar Radiation         0.324191
Sky Clearness           0.300573
Name: HVAC Electric Demand, dtype: float64
```

[22]:
```
of.corr()['HVAC Electric Demand'][3:-2]
```

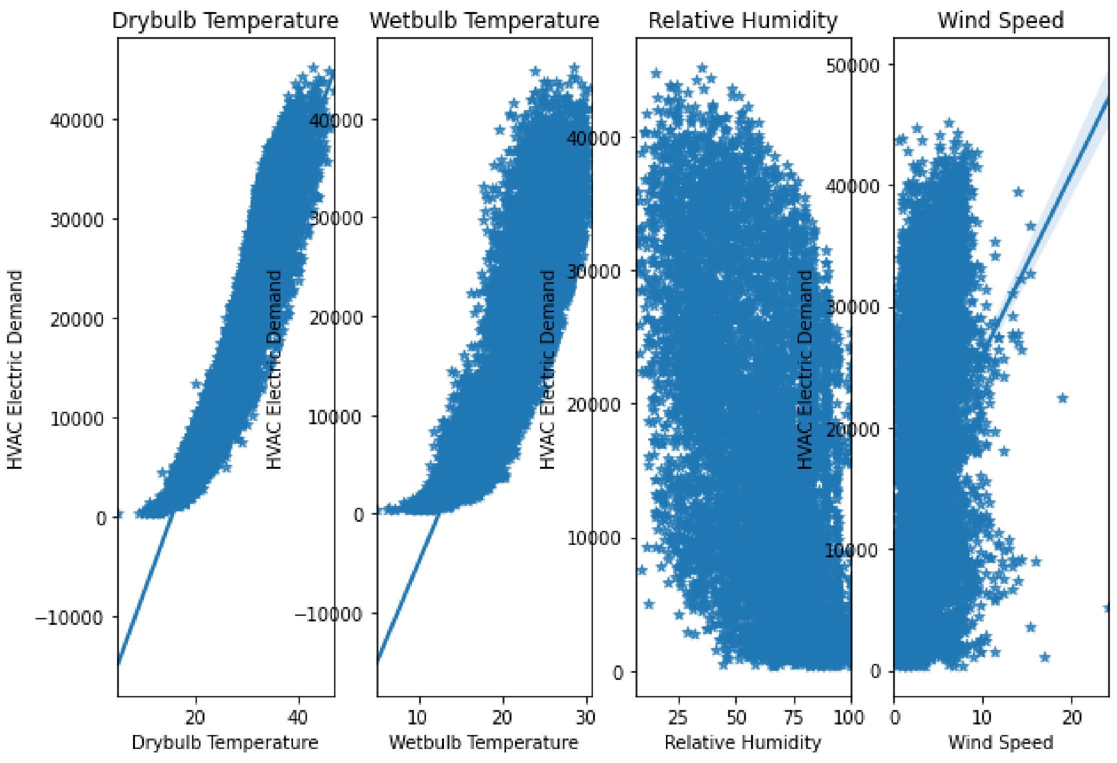
[22]:
```
Drybulb Temperature     0.693113
Wetbulb Temperature     0.432901
Relative Humidity      -0.567162
Wind Speed              0.396214
Wind Direction          0.301195
Solar Radiation         0.583861
```

```
Sky Clearness          0.553459
Name: HVAC Electric Demand, dtype: float64
```

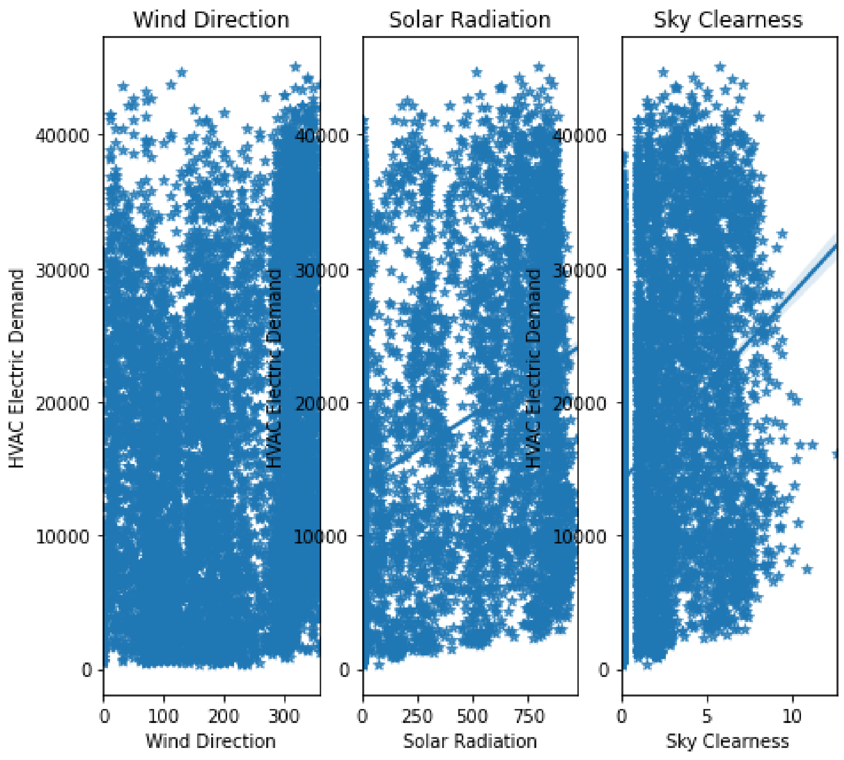
7 Question 4:

Create a scatter plot of the weather conditions vs HVAC demand and explain what you can learn from these associations for each building.

[23]:
```
# For Apartment
figure = plt.figure(figsize =(10,15))
weather_cols = apt.columns[3:-2].tolist()
for i, v in enumerate(weather_cols):
    figure.add_subplot(2,4,i+1)
    sns.regplot(x =v, y = "HVAC Electric Demand", data= apt, marker ='*')
    plt.title(v)
```



Drybulb Temperature
Wetbulb Temperature
Relative Humidity
Wind Speed
HVAC Electric Demand
Drybulb Temperature
Wetbulb Temperature
Relative Humidity
Wind Speed



Wind Direction
Solar Radiation
Sky Clearness
HVAC Electric Demand
Wind Direction
Solar Radiation
Sky Clearness

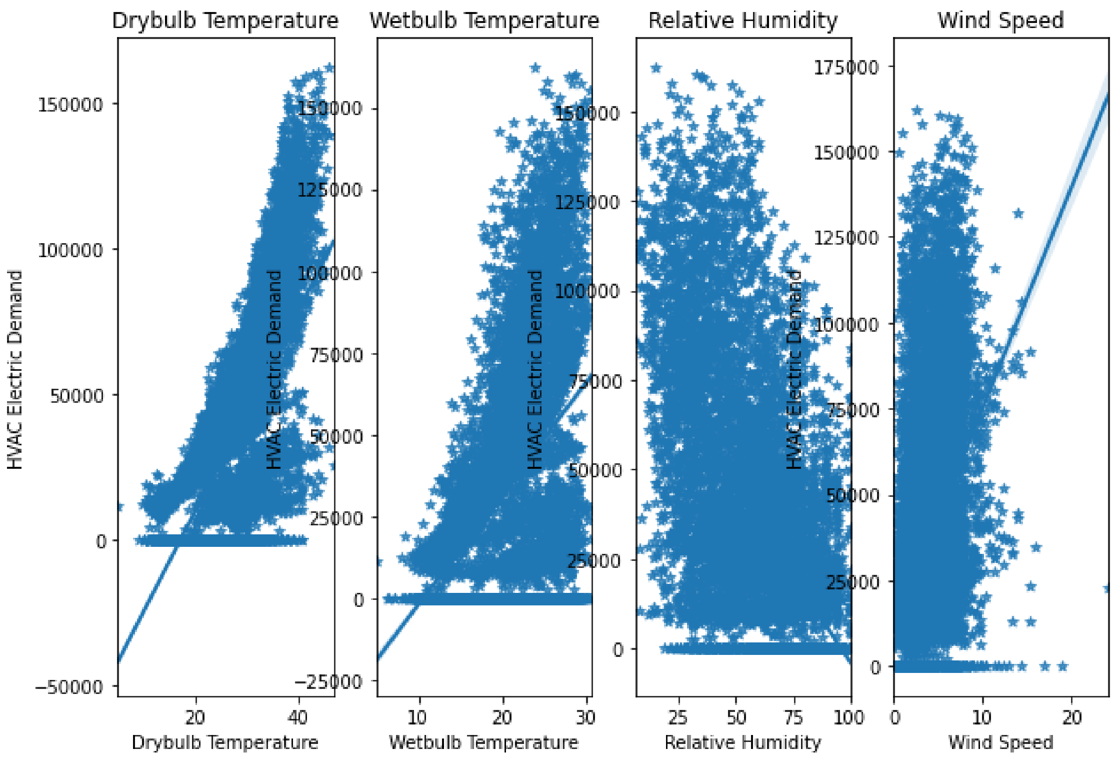
8

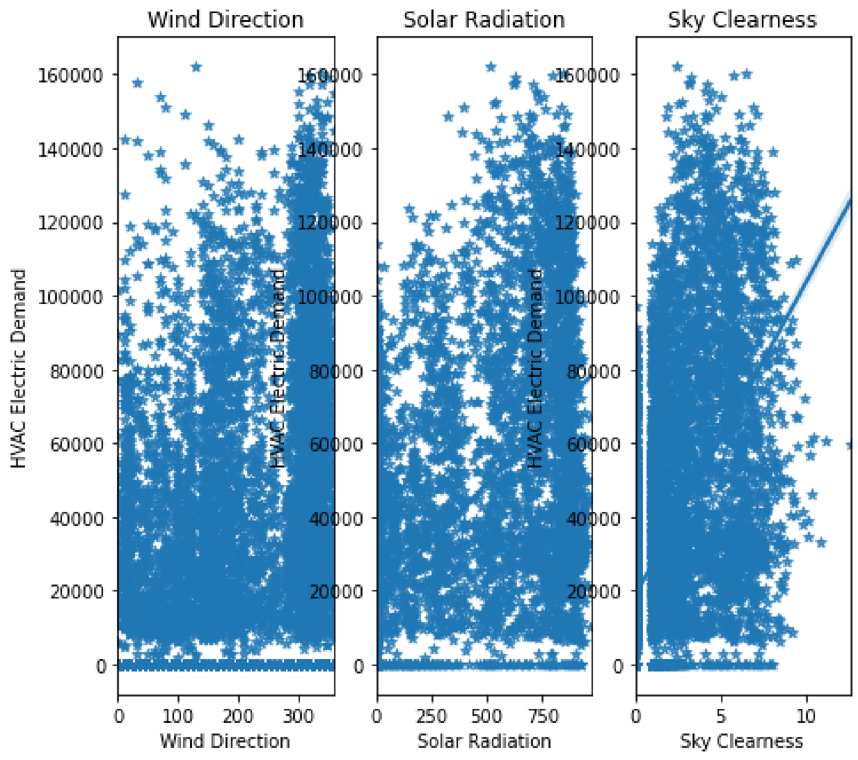
-> There is a high and positive correlation of Dry bulb temperature and wet bulb temperature with the HVAC Electric consumption in apartments

-> Other variables have low correlation; in fact, relative humidity has negative correlation

-> We, therefore, infer that with an increase in temperature the HVAC consumption seems to increase in the apartment and this accounts for more than 80 percent of variation in HVAC demand in apartments

[24]:
```
# For Office
figure = plt.figure(figsize =(10,15))
weather_cols = of.columns[3:-2].tolist()
for i, v in enumerate(weather_cols):
    figure.add_subplot(2,4,i+1)
    sns.regplot(x =v, y = "HVAC Electric Demand", data= of, marker ='*')
    plt.title(v)
```


Drybulb Temperature
Wetbulb Temperature
Relative Humidity
Wind Speed
HVAC Electric Demand
150000
100000
50000
0
−50000
125000
75000
25000
−25000
175000
20
40
10
30
25
50
75
100
Drybulb Temperature
Wetbulb Temperature
Relative Humidity
Wind Speed


Wind Direction
Solar Radiation
Sky Clearness
HVAC Electric Demand
160000
140000
120000
100000
80000
60000
40000
20000
0
100
200
300
250
500
750
5
10
Wind Direction
Solar Radiation
Sky Clearness

9

-> In office building we observe that again the HVAC consumption seems to increase with an increase in dry and wet bulb temperature. However, this accounts for lesser percentage of variation in HVAC consumption of the office building compared to percentage variation inside apartment building because of an increase in temperature.

-> We observe that there are other variables as well that account for variation in HVAC demand of office. They are Solar radiation and Sky clearness. The greater is the radiation on a given day of sun, and the greater the degree of clearness of sky, we find that HVAC demand increases in office

10 Question 5:

Split the data into training and test with a ratio of 0.2 as the test data.

[25]:
```
X_apt = apt[apt.columns[3:-2]]
y_apt =  apt["HVAC Electric Demand"]
```

[26]:
```
from sklearn.model_selection import train_test_split


X_train_apt, X_test_apt, y_train_apt, y_test_apt = train_test_split(X_apt,␣
 ↪y_apt, test_size=0.2)
```

[27]:
```
X_of = of[of.columns[3:-2]]
y_of =  of["HVAC Electric Demand"]
X_train_of, X_test_of, y_train_of, y_test_of = train_test_split(X_of,y_of,␣
 ↪test_size=0.2)
```

11 Question 6:

Create a linear regression model and train it based on the training data using weather conditions as the feature set and HVAC demand as the label for each building.

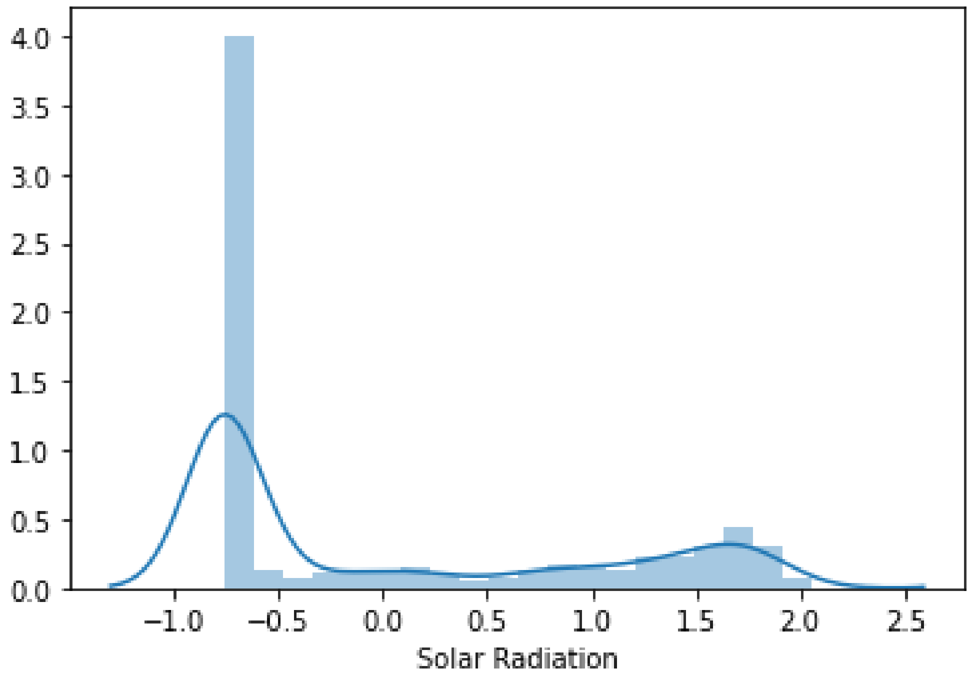
[28]:
```
# Standardizing the apartment and office datasets
```

[ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_apt[apt.columns[3:-2]] = scaler.fit_transform(X_train_apt)
X_test_apt[apt.columns[3:-2]] = scaler.transform(X_test_apt)
```

[ ]:
```
scaler = StandardScaler()
X_train_of[of.columns[3:-2]] = scaler.fit_transform(X_train_of)
X_test_of[of.columns[3:-2]] = scaler.transform(X_test_of)
```

[31]:
```
sns.distplot(X_train_apt["Solar Radiation"], bins = 20)
```

```
[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4bdfe862b0>
```



```
[32]: from sklearn.linear_model import LinearRegression
      from sklearn.metrics import mean_squared_error
      lr_apt = LinearRegression()
      lr_apt.fit(X_train_apt, y_train_apt)
      yhat_train = lr_apt.predict(X_train_apt)
      yhat_test = lr_apt.predict(X_test_apt)
      r2_train = lr_apt.score(X_train_apt, y_train_apt)
      mse_train =mean_squared_error(y_train_apt, yhat_train)
      r2_test = lr_apt.score(X_test_apt, y_test_apt)
      mse_test =mean_squared_error(y_test_apt, yhat_test)
```

```
[33]: print(f'For apartment building, r square value for training data is:␣
       ↪{round(r2_train,2)} with a mean squared error of: {int(mse_train)}')
      print(f'For apartment building, r square value for testing data is:␣
       ↪{round(r2_test,2)} with a mean squared error of: {int(mse_test)}')
```

```
For apartment building, r square value for training data is: 0.92 with a mean
squared error of: 9653654
For apartment building, r square value for testing data is: 0.92 with a mean
squared error of: 10238793
```

[34]:
```
lr_of = LinearRegression()
lr_of.fit(X_train_of, y_train_of)
yhat_train = lr_of.predict(X_train_of)
yhat_test = lr_of.predict(X_test_of)
r2_train = lr_of.score(X_train_of, y_train_of)
mse_train =mean_squared_error(y_train_of, yhat_train)
r2_test = lr_of.score(X_test_of, y_test_of)
mse_test =mean_squared_error(y_test_of, yhat_test)
```

[35]:
```
print(f'For Office building, r square value for training data is:␣
 ↪{round(r2_train,2)} with a mean squared error of: {int(mse_train)}')
print(f'For Office building, r square value for testing data is:␣
 ↪{round(r2_test,2)} with a mean squared error of: {int(mse_test)}')
```

```
For Office building, r square value for training data is: 0.6 with a mean
squared error of: 543690546
For Office building, r square value for testing data is: 0.6 with a mean squared
error of: 536418796
```

12 Question 7:

Incorporate the role of season and time of day into your regression model by introducing two sets of categorical variables:

First, explain how to add categorical variables into a regression model through OneHotEncoder in sklearn and what OneHotEncoder is (we did not cover this in our lecture and this is defined as an assignment for you.) Second, use OneHotEncoder object and transform 'month' column and concatenate it to your weather conditions input. Third, use pandas map method and convert the 'hour' column values as follows: {0,1,2,3,4,5}–>value=0 {6,7,8,9}–>value=1 {10,11,12}–>value=2 {13,14,15,16}–>value=3 {17,18,19}–>value=4 {20,21,22,23}–>value=5 Fourth, apply OneHotEncoder on this new column and concatenate it to your input.

13

13.0.1 Ans7:

To add a categorical variable in a regression model, we must convert that variable into dummy variables with one less classification to avoid the dummy trap. One Hot Encoder converts a categorical variable into a dataframe with columns equal to number of classes; each class shows a 1 if it is true for that class and 0 of that class is false. To use this variable in regression, we will remove one class of that variable which will become the base variable and the value of coefficient that will be assigned to one of the classes of a categorical variable, partaking in regression, will be compared with the base variable

[36]:
```
# Converting months to dummy for apartment and office
# Creating Month of July as the base variable
```

```
[37]: months = pd.get_dummies(apt['month'])
      months.drop(7, axis =1, inplace =True)
      months
```

```
[37]:       1   2   3   4   5   6   8   9   10  11  12
      0      1   0   0   0   0   0   0   0   0   0   0
      1      1   0   0   0   0   0   0   0   0   0   0
      2      1   0   0   0   0   0   0   0   0   0   0
      3      1   0   0   0   0   0   0   0   0   0   0
      4      1   0   0   0   0   0   0   0   0   0   0
      …     ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
      8755   0   0   0   0   0   0   0   0   0   0   1
      8756   0   0   0   0   0   0   0   0   0   0   1
      8757   0   0   0   0   0   0   0   0   0   0   1
      8758   0   0   0   0   0   0   0   0   0   0   1
      8759   0   0   0   0   0   0   0   0   0   0   1

      [8760 rows x 11 columns]
```

```
[38]: months.columns = ["Jan", "Feb", "March", "April", "May", "June", "Aug", "Sep",␣
       ↪"Oct", "Nov", "Dec"]
```

```
[39]: months
```

```
[39]:       Jan  Feb  March  April  May  June  Aug  Sep  Oct  Nov  Dec
      0       1    0      0      0    0     0    0    0    0    0    0
      1       1    0      0      0    0     0    0    0    0    0    0
      2       1    0      0      0    0     0    0    0    0    0    0
      3       1    0      0      0    0     0    0    0    0    0    0
      4       1    0      0      0    0     0    0    0    0    0    0
      …     …   …    …      …  …    …  …   …   …   …   …
      8755    0    0      0      0    0     0    0    0    0    0    1
      8756    0    0      0      0    0     0    0    0    0    0    1
      8757    0    0      0      0    0     0    0    0    0    0    1
      8758    0    0      0      0    0     0    0    0    0    0    1
      8759    0    0      0      0    0     0    0    0    0    0    1

      [8760 rows x 11 columns]
```

```
[40]: weather_variables = apt.columns[3:-2].tolist()
      weather_variables
```

```
[40]: ['Drybulb Temperature',
       'Wetbulb Temperature',
       'Relative Humidity',
       'Wind Speed',
       'Wind Direction',
       'Solar Radiation',
       'Sky Clearness']
```

[42]:
```
month_list = months.columns.tolist()
month_list
```

[42]:
```
['Jan',
 'Feb',
 'March',
 'April',
 'May',
 'June',
 'Aug',
 'Sep',
 'Oct',
 'Nov',
 'Dec']
```

[43]:
```
# changing hours to other category:
mapp = {0: 0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:1,7:1,8:1,9:1,10:2,11:2,12:2,13:3,14:
 ↪3,15:3,16:3, 17:4,18:4,19:4,20:5,21:5,22:5,23:5}
```

[44]:
```
apt['hour'] = apt['hour'].map(mapp)
of['hour'] = of['hour'].map(mapp)
```

[45]:
```
hours = pd.get_dummies(apt['hour'], drop_first =True) # Dropping 0 category
hours
```

[45]:
```
      1  2  3  4  5
0     0  0  0  0  0
1     0  0  0  0  0
2     0  0  0  0  0
3     0  0  0  0  0
4     0  0  0  0  0
…    .. .. .. .. ..
8755  0  0  0  0  1
8756  0  0  0  0  1
8757  0  0  0  0  1
8758  0  0  0  0  1
8759  0  0  0  0  0

[8760 rows x 5 columns]
```

[46]:
```
apt_wmh = pd.concat([apt, months, hours], axis=1)
apt_wmh.head()
```

[46]:
```
   month  day  hour  Drybulb Temperature  Wetbulb Temperature  \
0      1    1     0                 16.0            13.705041
1      1    1     0                 15.6            13.758291
2      1    1     0                 15.1            13.595604
3      1    1     0                 14.8            13.512457
4      1    1     0                 14.4            13.227824

   Relative Humidity  Wind Speed  Wind Direction  Solar Radiation  \
0                 78         0.5             190                0
1                 82         2.1             120                0
2                 85         2.1             120                0
3                 87         2.1             140                0
4                 88         1.0             150                0

   Sky Clearness  …  Aug  Sep  Oct  Nov  Dec  1  2  3  4  5
0            0.0  …    0    0    0    0    0  0  0  0  0  0
1            0.0  …    0    0    0    0    0  0  0  0  0  0
2            0.0  …    0    0    0    0    0  0  0  0  0  0
3            0.0  …    0    0    0    0    0  0  0  0  0  0
4            0.0  …    0    0    0    0    0  0  0  0  0  0

[5 rows x 28 columns]
```

[47]:
```
hours_list = hours.columns.tolist()
```

[48]:
```
X_apt_wm = apt_wmh[weather_variables + month_list + hours_list]
y_apt_wm=  apt_wmh["HVAC Electric Demand"]
X_train_apt, X_test_apt, y_train_apt, y_test_apt =␣
 ↪train_test_split(X_apt_wm,y_apt_wm, test_size=0.2)
```

[49]:
```
# Doing the same procedure and splitting for office building data set
```

[50]:
```
months = pd.get_dummies(of['month'])
months.drop(7, axis =1, inplace =True)
months
```

[50]:
```
      1   2   3   4   5   6   8   9   10  11  12
0      1   0   0   0   0   0   0   0   0   0   0
1      1   0   0   0   0   0   0   0   0   0   0
2      1   0   0   0   0   0   0   0   0   0   0
3      1   0   0   0   0   0   0   0   0   0   0
4      1   0   0   0   0   0   0   0   0   0   0
…    ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
8755  0   0   0   0   0   0   0   0   0   0   1
8756  0   0   0   0   0   0   0   0   0   0   1
8757  0   0   0   0   0   0   0   0   0   0   1
8758  0   0   0   0   0   0   0   0   0   0   1
```

```
8759   0   0   0   0   0   0   0   0   0   0   1

[8760 rows x 11 columns]
```

[51]:
```
months.columns = month_list
```

[52]:
```
hours = pd.get_dummies(of['hour'], drop_first =True) # Dropping 0 category
hours
```

[52]:
```
      1  2  3  4  5
0     0  0  0  0  0
1     0  0  0  0  0
2     0  0  0  0  0
3     0  0  0  0  0
4     0  0  0  0  0
…    .. .. .. .. ..
8755  0  0  0  0  1
8756  0  0  0  0  1
8757  0  0  0  0  1
8758  0  0  0  0  1
8759  0  0  0  0  0

[8760 rows x 5 columns]
```

[53]:
```
of_wmh = pd.concat([of, months, hours], axis=1)
of_wmh.head()
```

[53]:
```
   month  day  hour  Drybulb Temperature  Wetbulb Temperature  \
0      1    1     0                 16.0            13.705041
1      1    1     0                 15.6            13.758291
2      1    1     0                 15.1            13.595604
3      1    1     0                 14.8            13.512457
4      1    1     0                 14.4            13.227824

   Relative Humidity  Wind Speed  Wind Direction  Solar Radiation  \
0                 78         0.5             190                0
1                 82         2.1             120                0
2                 85         2.1             120                0
3                 87         2.1             140                0
4                 88         1.0             150                0

   Sky Clearness  …  Aug  Sep  Oct  Nov  Dec  1  2  3  4  5
0            0.0  …    0    0    0    0    0  0  0  0  0  0
1            0.0  …    0    0    0    0    0  0  0  0  0  0
2            0.0  …    0    0    0    0    0  0  0  0  0  0
3            0.0  …    0    0    0    0    0  0  0  0  0  0
4            0.0  …    0    0    0    0    0  0  0  0  0  0
```

```
[5 rows x 28 columns]
```

[54]:
```
X_of_wm = of_wmh[weather_variables + month_list + hours_list]
y_of_wm=  of_wmh["HVAC Electric Demand"]
X_train_of, X_test_of, y_train_of, y_test_of =␣
 ↪train_test_split(X_of_wm,y_of_wm, test_size=0.2)
```

14 Question 8:

Repeat question 6 with the new dataset for both buildings and report any improvement you see in training and test MSE values.

[55]:
```
# Standardizing all variables of both concatenated datasets except dummy␣
 ↪variables
```

[56]:
```
X_train_apt.columns[:-16]
```

[56]:
```
Index(['Drybulb Temperature', 'Wetbulb Temperature', 'Relative Humidity',
       'Wind Speed', 'Wind Direction', 'Solar Radiation', 'Sky Clearness'],
      dtype='object')
```

[ ]:
```
scaler = StandardScaler()
X_train_apt[X_train_apt.columns[:-16]] = scaler.
 ↪fit_transform(X_train_apt[X_train_apt.columns[:-16]])
X_test_apt[X_test_apt.columns[:-16]] = scaler.transform(X_test_apt[X_test_apt.
 ↪columns[:-16]])
```

[ ]:
```
scaler = StandardScaler()
X_train_of[X_train_of.columns[:-16]] = scaler.
 ↪fit_transform(X_train_of[X_train_of.columns[:-16]])
X_test_of[X_test_of.columns[:-16]] = scaler.transform(X_test_of[X_test_of.
 ↪columns[:-16]])
```

Regression on Apartment's Concatenated Data

[59]:
```
lr_apt_concat = LinearRegression()
lr_apt_concat.fit(X_train_apt, y_train_apt)
yhat_train = lr_apt_concat.predict(X_train_apt)
yhat_test = lr_apt_concat.predict(X_test_apt)
r2_train = lr_apt_concat.score(X_train_apt, y_train_apt)
mse_train =mean_squared_error(y_train_apt, yhat_train)
r2_test = lr_apt_concat.score(X_test_apt, y_test_apt)
mse_test =mean_squared_error(y_test_apt, yhat_test)
print(f'For apartment building, r square value for training data with weather␣
 ↪and time variables is: {round(r2_train,2)} with a mean squared error of:␣
 ↪{int(mse_train)}')
```

```
print(f'For apartment building, r square value for testing data with weather␣
 ↪and time variables is: {round(r2_test,2)} with a mean squared error of:␣
 ↪{int(mse_test)}')
```

```
For apartment building, r square value for training data with weather and time
variables is: 0.97 with a mean squared error of: 4227240
For apartment building, r square value for testing data with weather and time
variables is: 0.97 with a mean squared error of: 3982953
```

Regression on Office's Concatenated Data

[60]:
```
lr_of_concat = LinearRegression()
lr_of_concat.fit(X_train_of, y_train_of)
yhat_train = lr_of_concat.predict(X_train_of)
yhat_test = lr_of_concat.predict(X_test_of)
r2_train = lr_of_concat.score(X_train_of, y_train_of)
mse_train =mean_squared_error(y_train_of, yhat_train)
r2_test = lr_of_concat.score(X_test_of, y_test_of)
mse_test =mean_squared_error(y_test_of, yhat_test)
print(f'For Office building, r square value for training data with weather and␣
 ↪time variables is: {round(r2_train,2)} with a mean squared error of:␣
 ↪{int(mse_train)}')
print(f'For Office building, r square value for testing data with weather and␣
 ↪time variables is: {round(r2_test,2)} with a mean squared error of:␣
 ↪{int(mse_test)}')
```

```
For Office building, r square value for training data with weather and time
variables is: 0.67 with a mean squared error of: 450501802
For Office building, r square value for testing data with weather and time
variables is: 0.67 with a mean squared error of: 432940468
```

For the apartment building, adding months and hours has increased the r2 value of training and testing data from 0.92 to 0.97 and has significantly decreased the MSE value of both sets from approximately 9813801 to 4100685

For the office building, adding months and hours has increased the r2 value of training and testing data from 0.6 to 0.66 (training) and 0.59 to 0.68 (testing) and has significantly decreased the MSE value of both sets from approximately 539165142 to 450700177

15 Question 8:

Explain what regularization is in supervised learning and repeat step 8 using sklearn Ridge and Lasso classes based on the below instruction:

Ans -> Regularization is the addition of bias in the model to prevent the model from overfitting due to the noise in the data by reducing the variance of the estimates as per the degree of regularization parameter alpha specified

[61]:
```
# Ridge Regression on both buildings
```

[62]:
```
from sklearn.linear_model import Ridge
RigeModel_apt_concat=Ridge()
RigeModel_apt_concat.fit(X_train_apt, y_train_apt)
yhat_train = RigeModel_apt_concat.predict(X_train_apt)
yhat_test = RigeModel_apt_concat.predict(X_test_apt)
r2_train = RigeModel_apt_concat.score(X_train_apt, y_train_apt)
mse_train =mean_squared_error(y_train_apt, yhat_train)
r2_test = RigeModel_apt_concat.score(X_test_apt, y_test_apt)
mse_test =mean_squared_error(y_test_apt, yhat_test)
print(f'For apartment building, r square value using Ridge Regression using␣
 ↪default value of alpha, for training data with weather and time variables is:
 ↪ {round(r2_train,2)} with a mean squared error of: {int(mse_train)}')
print(f'For apartment building, r square value, using Ridge Regression using␣
 ↪default value of alpha, for testing data with weather and time variables is:␣
 ↪{round(r2_test,2)} with a mean squared error of: {int(mse_test)}')
```

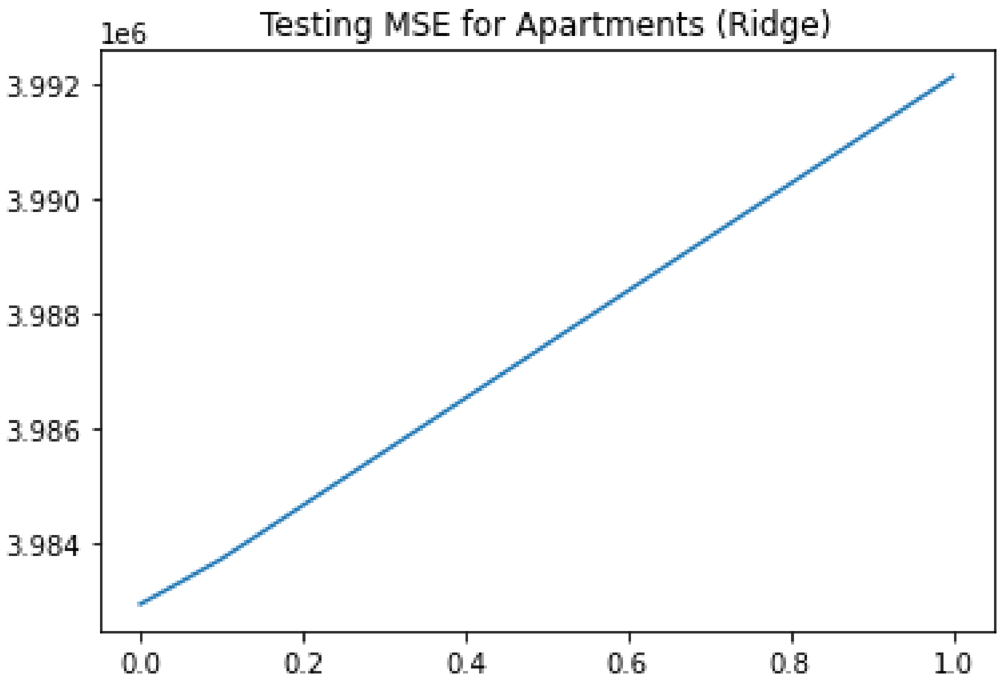
```
For apartment building, r square value using Ridge Regression using default
value of alpha, for training data with weather and time variables is: 0.97 with
a mean squared error of: 4229056
For apartment building, r square value, using Ridge Regression using default
value of alpha, for testing data with weather and time variables is: 0.97 with a
mean squared error of: 3992116
```

[63]:
```
RigeModel_of_concat=Ridge()
RigeModel_of_concat.fit(X_train_of, y_train_of)
yhat_train = RigeModel_of_concat.predict(X_train_of)
yhat_test = RigeModel_of_concat.predict(X_test_of)
r2_train = RigeModel_of_concat.score(X_train_of, y_train_of)
mse_train =mean_squared_error(y_train_of, yhat_train)
r2_test = RigeModel_of_concat.score(X_test_of, y_test_of)
mse_test =mean_squared_error(y_test_of, yhat_test)
print(f'For Office building, r square value using Ridge Regression using␣
 ↪default value of alpha, for training data with weather and time variables is:
 ↪ {round(r2_train,2)} with a mean squared error of: {int(mse_train)}')
print(f'For Office building, r square value, using Ridge Regression using␣
 ↪default value of alpha, for testing data with weather and time variables is:␣
 ↪{round(r2_test,2)} with a mean squared error of: {int(mse_test)}')
```

```
For Office building, r square value using Ridge Regression using default value
of alpha, for training data with weather and time variables is: 0.67 with a mean
squared error of: 450512538
For Office building, r square value, using Ridge Regression using default value
of alpha, for testing data with weather and time variables is: 0.67 with a mean
squared error of: 433014600
```

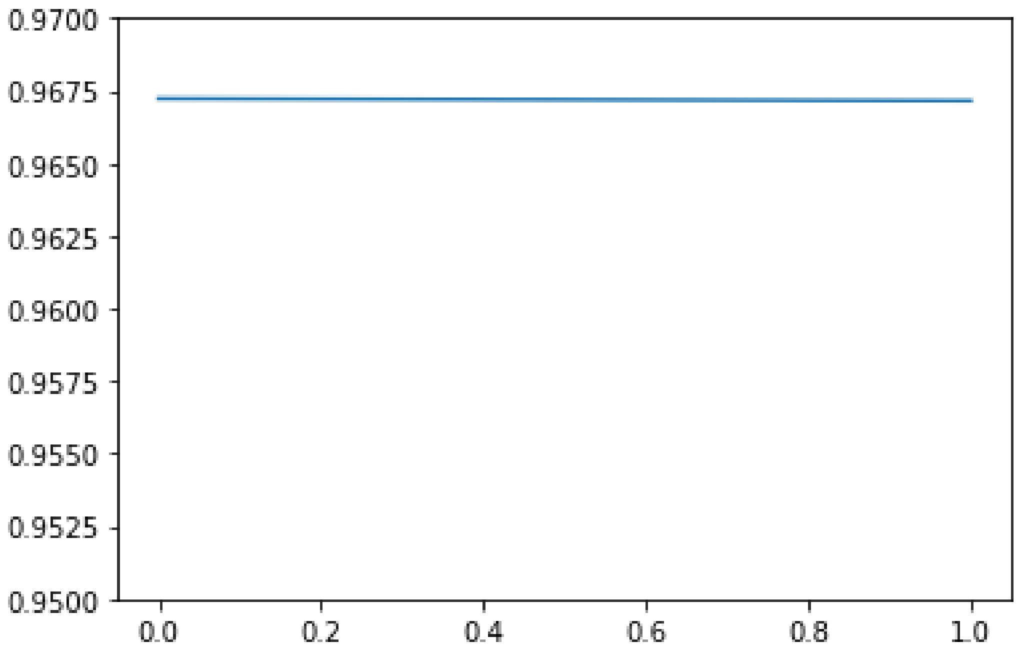
[64]:
```
# Plotting mse graph for Apartments
test_mse_apt = []
test_r2_apt = []
ALFA = [0,0.005,0.05,0.1,1]
for alfa in ALFA:
    RigeModel_apt_concat = Ridge(alpha=alfa)
    RigeModel_apt_concat.fit(X_train_apt, y_train_apt)
    yhat_test = RigeModel_apt_concat.predict(X_test_apt)
    r2_test = RigeModel_apt_concat.score(X_test_apt, y_test_apt)
    test_r2_apt.append(r2_test)
    mse_test =mean_squared_error(y_test_apt, yhat_test)
    test_mse_apt.append(mse_test)
```

[65]:
```
plt.plot(ALFA, test_mse_apt)
plt.title("Testing MSE for Apartments (Ridge)");
```



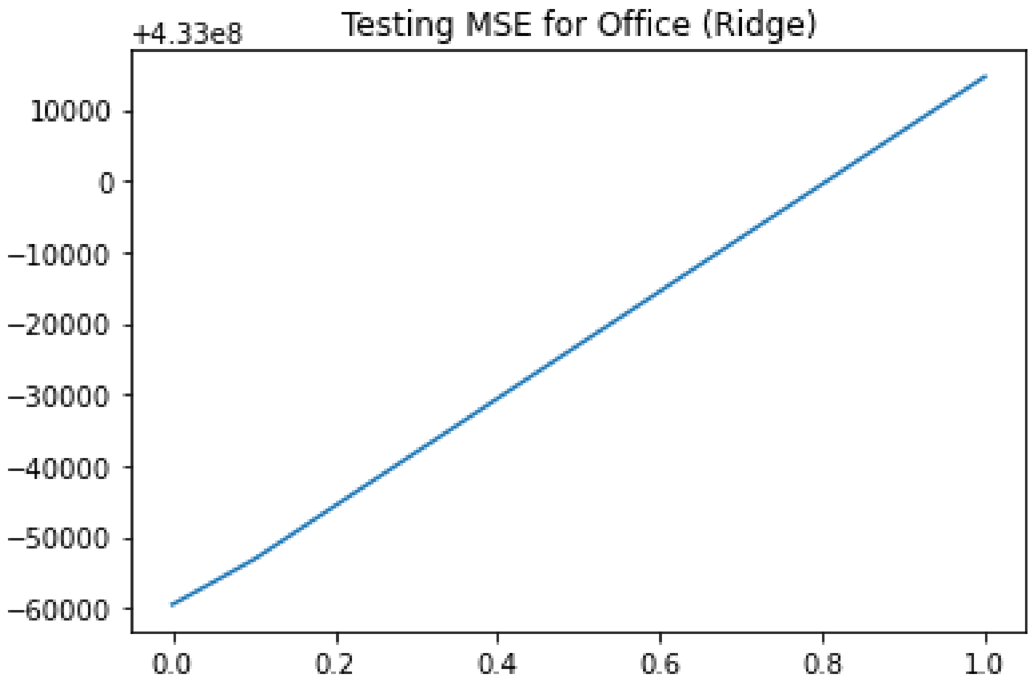
[66]:
```
plt.plot(ALFA,test_r2_apt)
plt.ylim(0.95,0.97)
#list(zip(ALFA,test_r2_apt))
```

[66]:
```
(0.95, 0.97)
```



[67]:
```
# Plotting mse graph for Office
test_mse_of = []
ALFA = [0,0.005,0.05,0.1,1]
for alfa in ALFA:
    RigeModel_of_concat=Ridge(alpha =alfa)
    RigeModel_of_concat.fit(X_train_of, y_train_of)
    yhat_test = RigeModel_of_concat.predict(X_test_of)
    mse_test =mean_squared_error(y_test_of, yhat_test)
    test_mse_of.append(mse_test)
```

[68]:
```
plt.plot(ALFA, test_mse_of)
plt.title("Testing MSE for Office (Ridge)");
```



Increasing the value of alpha for both buildings with Ridge regression increases the Mean Squared Error of the model which is the price we must pay in order to introduce bias in the model

[69]:
```
# Lasso Regression on both buildings
```

[70]:
```
LassoModel_apt_concat=Lasso()
LassoModel_apt_concat.fit(X_train_apt, y_train_apt)
yhat_train = LassoModel_apt_concat.predict(X_train_apt)
yhat_test = LassoModel_apt_concat.predict(X_test_apt)
r2_train = LassoModel_apt_concat.score(X_train_apt, y_train_apt)
mse_train =mean_squared_error(y_train_apt, yhat_train)
r2_test = LassoModel_apt_concat.score(X_test_apt, y_test_apt)
mse_test =mean_squared_error(y_test_apt, yhat_test)
print(f'For apartment building, r square value using Lasso Regression using␣
 ↪default value of alpha, for training data with weather and time variables is:
 ↪ {round(r2_train,2)} with a mean squared error of: {int(mse_train)}')
print(f'For apartment building, r square value, using Lasso  Regression using␣
 ↪default value of alpha, for testing data with weather and time variables is:␣
 ↪{round(r2_test,2)} with a mean squared error of: {int(mse_test)}')
```

```
For apartment building, r square value using Lasso Regression using default
value of alpha, for training data with weather and time variables is: 0.97 with
a mean squared error of: 4230428
For apartment building, r square value, using Lasso  Regression using default
```

```
value of alpha, for testing data with weather and time variables is: 0.97 with a
mean squared error of: 3995146
```

[71]:
```
LassoModel_of_concat=Lasso()
LassoModel_of_concat.fit(X_train_of, y_train_of)
yhat_train = LassoModel_of_concat.predict(X_train_of)
yhat_test = LassoModel_of_concat.predict(X_test_of)
r2_train = LassoModel_of_concat.score(X_train_of, y_train_of)
mse_train =mean_squared_error(y_train_of, yhat_train)
r2_test = LassoModel_of_concat.score(X_test_of, y_test_of)
mse_test =mean_squared_error(y_test_of, yhat_test)
print(f'For Office building, r square value using Lasso Regression using␣
 ↪default value of alpha, for training data with weather and time variables is:
 ↪ {round(r2_train,2)} with a mean squared error of: {int(mse_train)}')
print(f'For Office building, r square value, using Lasso  Regression using␣
 ↪default value of alpha, for testing data with weather and time variables is:␣
 ↪{round(r2_test,2)} with a mean squared error of: {int(mse_test)}')
```

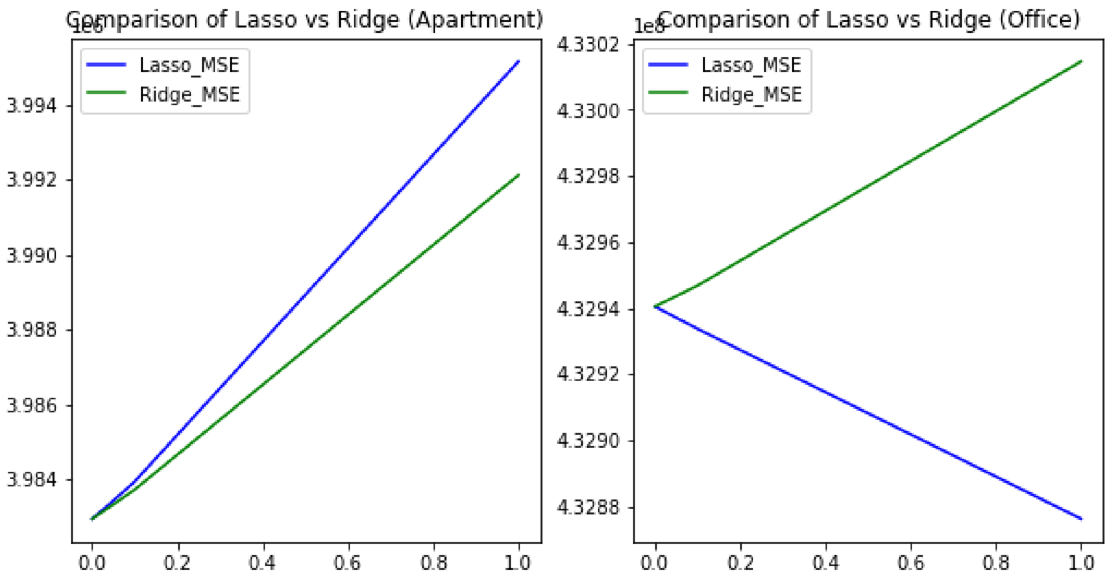
```
For Office building, r square value using Lasso Regression using default value
of alpha, for training data with weather and time variables is: 0.67 with a mean
squared error of: 450505360
For Office building, r square value, using Lasso  Regression using default value
of alpha, for testing data with weather and time variables is: 0.67 with a mean
squared error of: 432876222
```

[ ]:
```
# Plotting mse graph for Apartments
test_mse_apt_lasso = []
test_r2_apt_lasso = []
ALFA = [0,0.005,0.05,0.1,1]
for alfa in ALFA:
    LassoModel_apt_concat = Lasso(alpha=alfa)
    LassoModel_apt_concat.fit(X_train_apt, y_train_apt)
    yhat_test = LassoModel_apt_concat.predict(X_test_apt)
    r2_test = LassoModel_apt_concat.score(X_test_apt, y_test_apt)
    test_r2_apt_lasso.append(r2_test)
    mse_test =mean_squared_error(y_test_apt, yhat_test)
    test_mse_apt_lasso.append(mse_test)
plt.plot(ALFA, test_mse_apt_lasso, color ='b')
plt.title("Testing MSE for Apartments (Lasso)")
```

[ ]:
```
# Plotting mse graph for Office
test_mse_of_lasso = []
ALFA = [0,0.005,0.05,0.1,1]
for alfa in ALFA:
    LassoModel_of_concat=Lasso(alpha =alfa)
    LassoModel_of_concat.fit(X_train_of, y_train_of)
    yhat_test = LassoModel_of_concat.predict(X_test_of)
    mse_test =mean_squared_error(y_test_of, yhat_test)
    test_mse_of_lasso.append(mse_test)

plt.plot(ALFA, test_mse_of_lasso)
plt.title("Testing MSE for Office (Lasso)");
```

[74]:
```
# Comparing Lasso with Ridge
```

[75]:
```
figure =plt.figure(figsize= (10,5))
figure.add_subplot(1,2,1)
plt.plot(ALFA, test_mse_apt_lasso, color ='b', label = "Lasso_MSE")
plt.plot(ALFA, test_mse_apt, color ='g', label ="Ridge_MSE")
plt.title("Comparison of Lasso vs Ridge (Apartment)")
plt.legend();
figure.add_subplot(1,2,2)
plt.plot(ALFA, test_mse_of_lasso, color ='b', label = "Lasso_MSE")
plt.plot(ALFA, test_mse_of, color ='g', label ="Ridge_MSE")
plt.title("Comparison of Lasso vs Ridge (Office)")
plt.legend();
```



There is not much difference in Lasso and Ridge for apartment building but Lasso clearly performs better than Ridge for office building!

16 Question 8:

Use the following sklearn regressors and compare the training and test MSE values and report the model with the best generalization (do not change the default values for these objects):

AdaBoostRegresso; BaggingRegressor; SVR; RandomForestRegressor

[76]:
```
# Comparing train and test MSE for Apartments
```

[77]:
```
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
regressor_names = ["SVR", "BaggingRegressor", "DecisionTreeRegressor",␣
 ↪"AdaBoostRegressor"]
```

[78]:
```
mse_train_list = []
mse_test_list = []
for name in regressor_names:
    regressor_model =eval(name)()
    regressor_model.fit(X_train_apt, y_train_apt)
    yhat_train = regressor_model.predict(X_train_apt)
    yhat_test = regressor_model.predict(X_test_apt)
    mse_train =mean_squared_error(y_train_apt, yhat_train)
    mse_test =mean_squared_error(y_test_apt, yhat_test)
    mse_train_list.append(mse_train)
    mse_test_list.append(mse_test)
```

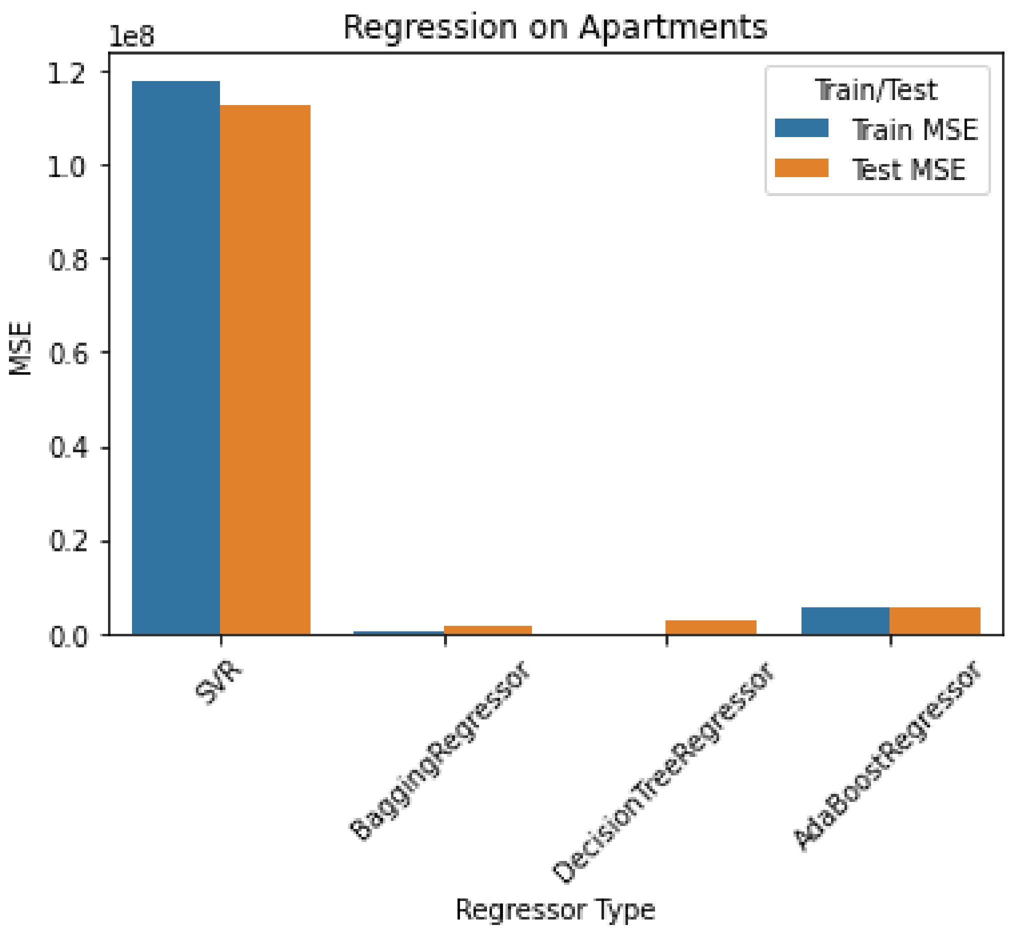
[79]:
```
Regressors = pd.DataFrame({"Train MSE":mse_train_list, "Test MSE":
 ↪mse_test_list, "Regressor":regressor_names})
Regressors
```

[79]:
```
      Train MSE      Test MSE              Regressor
0  1.177923e+08  1.123306e+08                    SVR
1  2.843997e+05  1.465651e+06       BaggingRegressor
2  6.346153e+02  2.748232e+06  DecisionTreeRegressor
3  5.573272e+06  5.570869e+06      AdaBoostRegressor
```

[80]:
```
regressor = pd.melt(Regressors, id_vars = ['Regressor'] ,value_vars = ["Train␣
 ↪MSE", "Test MSE"])
regressor.columns = ['Regressor Type', "Train/Test", "MSE"]
regressor
```

[80]:
```
          Regressor Type Train/Test           MSE
0                    SVR  Train MSE  1.177923e+08
1       BaggingRegressor  Train MSE  2.843997e+05
2  DecisionTreeRegressor  Train MSE  6.346153e+02
3      AdaBoostRegressor  Train MSE  5.573272e+06
4                    SVR   Test MSE  1.123306e+08
5       BaggingRegressor   Test MSE  1.465651e+06
6  DecisionTreeRegressor   Test MSE  2.748232e+06
7      AdaBoostRegressor   Test MSE  5.570869e+06
```

[87]:
```
sns.barplot(x= "Regressor Type", y="MSE", hue = "Train/Test", data =regressor)
plt.title("Regression on Apartments")
plt.xticks(rotation = 45);
```

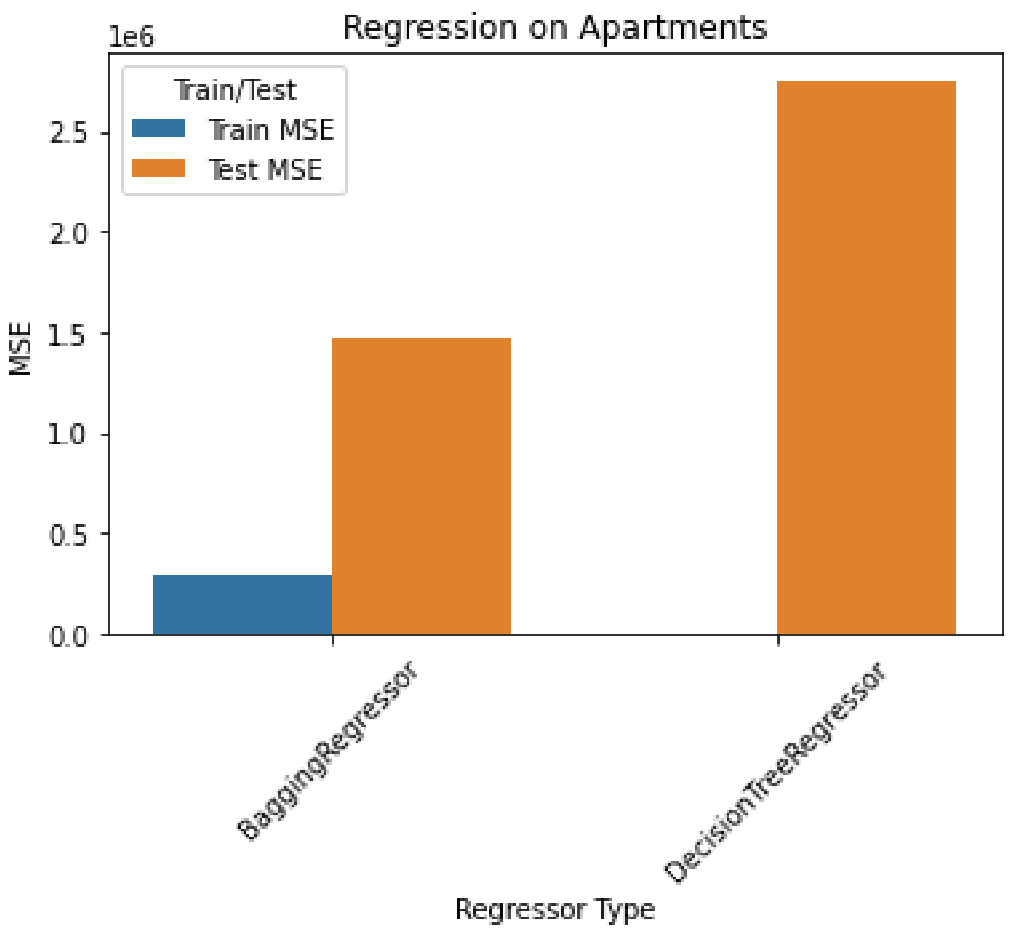


[82]:
```
# checking which one is better (Bagging or DecisionTreeRegressor)
```

[83]:
```
new = regressor[(regressor["Regressor Type"] != "SVR") & (regressor["Regressor␣
↪Type"] != "AdaBoostRegressor")]
new
```

[83]:
```
           Regressor Type Train/Test           MSE
1        BaggingRegressor  Train MSE  2.843997e+05
2   DecisionTreeRegressor  Train MSE  6.346153e+02
5        BaggingRegressor   Test MSE  1.465651e+06
6   DecisionTreeRegressor   Test MSE  2.748232e+06
```

[86]:
```
sns.barplot(x= "Regressor Type", y="MSE", hue = "Train/Test", data =new)
plt.title("Regression on Apartments")
plt.xticks(rotation = 45);
```

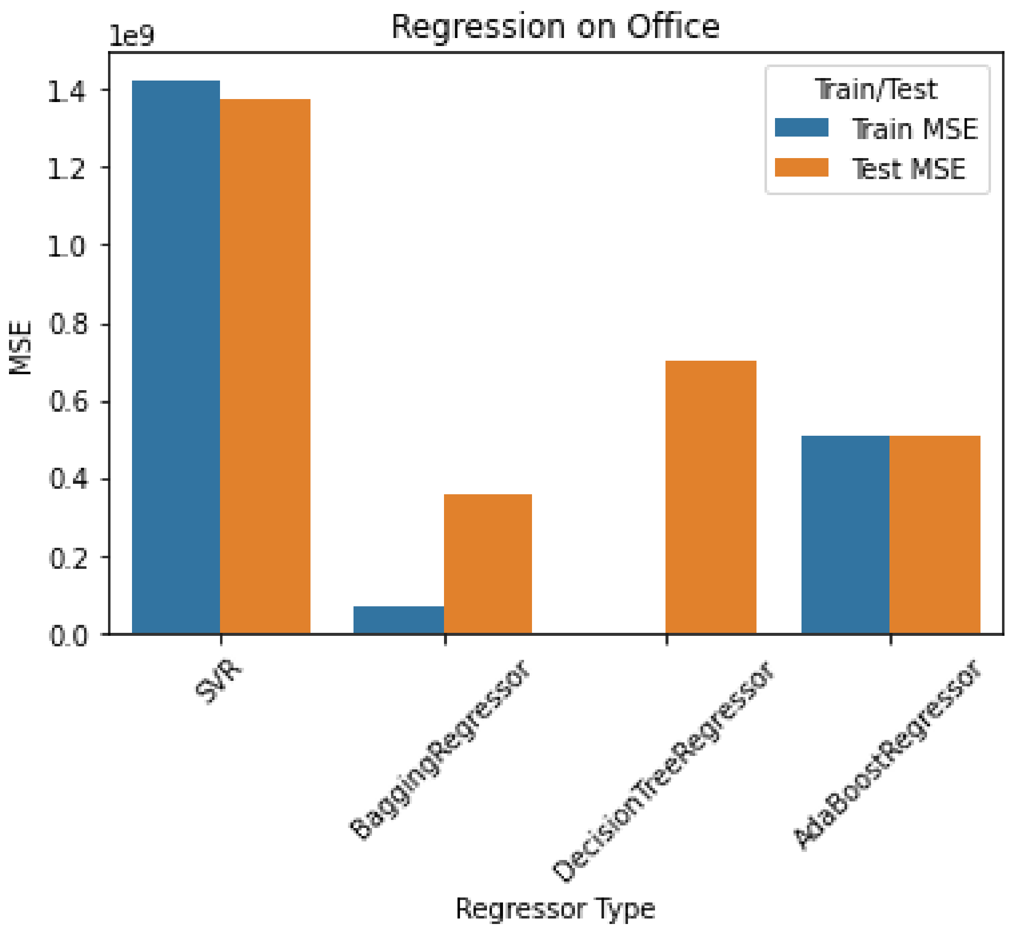


Although the training data has 0 MSE for Decision Tree Regressor, but Bagging Regressor has the best performace because its MSE on testing data is much lower than that for DecisionTreeRegressor on testing data

[ ]:
```
# Comparing train and test MSE for Office
```

[85]:
```
mse_train_list = []
mse_test_list = []
for name in regressor_names:
    regressor_model =eval(name)()
    regressor_model.fit(X_train_of, y_train_of)
    yhat_train = regressor_model.predict(X_train_of)
    yhat_test = regressor_model.predict(X_test_of)
    mse_train =mean_squared_error(y_train_of, yhat_train)
    mse_test =mean_squared_error(y_test_of, yhat_test)
    mse_train_list.append(mse_train)
    mse_test_list.append(mse_test)
```

[89]:
```
Regressors = pd.DataFrame({"Train MSE":mse_train_list, "Test MSE":
 ↪mse_test_list, "Regressor":regressor_names})
regressor = pd.melt(Regressors, id_vars = ['Regressor'] ,value_vars = ["Train␣
 ↪MSE", "Test MSE"])
regressor.columns = ['Regressor Type', "Train/Test", "MSE"]
sns.barplot(x= "Regressor Type", y="MSE", hue = "Train/Test", data =regressor)
plt.xticks(rotation = 45)
plt.title("Regression on Office");
```



For the office building, MSE is quite high for all types but Bagging Regressor has outperformed all, just like in the case of apartment

[ ]: