Energy Usage Prediction - Hyeondeok Cho

In [1]:
```
import pandas as pd
import numpy as np
import datetime
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import root_mean_squared_error
```

In [2]:
```
#Task 1
#Examine the data, parse the time fields wherever necessary. Take the sum of the
#(Use [kW]) to get per day usage and merge it with weather data

#Dataframe for weather and energy
weather_data = pd.read_csv('weather_data.csv')
energy_data = pd.read_csv('energy_data.csv')
```

In [3]:
```
weather_data.head()
```

Out[3]:

	temperature	icon	humidity	visibility	summary	pressure	windSpeed	cloudCover	
0	34.98	partly-cloudy-night	0.64	10.00	Partly Cloudy	1017.69	7.75	0.29	138853
1	16.49	clear-night	0.62	10.00	Clear	1022.76	2.71	0.06	138853
2	14.63	clear-night	0.68	10.00	Clear	1022.32	4.84	0.03	138854
3	13.31	clear-night	0.71	10.00	Clear	1021.64	4.00	0.14	138854
4	13.57	clear-night	0.71	9.93	Clear	1020.73	3.67	0.04	138854

In [4]:
```
#In the "time" column from weather data, each row should be unique.
#check if there is duplicated value of time.
weather_data[weather_data.duplicated(subset=['time'])]
```

Out[4]:

temperature	icon	humidity	visibility	summary	pressure	windSpeed	cloudCover	time	windE

There is no duplicated time in weather_data.

In [5]:
```
#check if there is any null value in weather_data
weather_data.isnull().any()
```

Out[5]:
```
temperature          False
icon                 False
humidity             False
visibility           False
summary              False
pressure             False
windSpeed            False
cloudCover            True
time                 False
windBearing          False
precipIntensity      False
dewPoint             False
precipProbability    False
dtype: bool
```

In [6]:
```
weather_data['cloudCover'].isnull().sum()
```

Out[6]:
```
1470
```

There are 1470 rows that have null values in the "cloudCover" column from weather_data, but I don't think they will affect significantly to my prediction of the electricity usage based on weather condition, so I would just leave them as null values.

In [7]:
```
energy_data.head()
```

Out[7]:

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utilit Base Bath
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.00
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.0C
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.00
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.00
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.00

In [8]:
```
#In the "Date & Time" column from energy_data, each row should be unique.
#check if there is duplicated value of "Date & Time"
energy_data[energy_data.duplicated(subset=['Date & Time'])]
```

Out[8]:

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]
14641	2014-11-02 01:00:00	0.483964	0.0	0.483964	0.000019	0.009070	0.010133	0.000276	0.003584
14643	2014-11-02 01:30:00	0.474490	0.0	0.474490	0.000034	0.009326	0.018928	0.000294	0.003708

After checking duplicated value, it shows that there are two duplicated rows, but even though they are same date, time is different. So we can safely assume that both of them are unique in terms of Date & Time.

In [9]:

```
#check if there is any null value in energy_data
energy_data.isnull().any()
```

Out[9]:

```
Date & Time                            False
use [kW]                               False
gen [kW]                               False
Grid [kW]                              False
AC [kW]                                False
Furnace [kW]                           False
Cellar Lights [kW]                     False
Washer [kW]                            False
First Floor lights [kW]                False
Utility Rm + Basement Bath [kW]        False
Garage outlets [kW]                    False
MBed + KBed outlets [kW]               False
Dryer + egauge [kW]                    False
Panel GFI (central vac) [kW]           False
Home Office (R) [kW]                   False
Dining room (R) [kW]                   False
Microwave (R) [kW]                     False
Fridge (R) [kW]                        False
dtype: bool
```

There is no null values in energy_data

In [10]:

```
#Since the "Date & Time" column is object type, we can convert it to datetime ty
energy_data['Date & Time'] = pd.to_datetime(energy_data['Date & Time'])
```

In [11]:

```
energy_data.head()
```

Out[11]:

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utilit Base Bath
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.00

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utilit Base Bath
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.0C
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.00
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.00
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.00

In [12]:
```
#create new column called "Date" in order to to take sum the sum of the energy u
energy_data['Date'] = pd.to_datetime(energy_data['Date & Time']).dt.date
```

In [13]:
```
energy_data.head()
```

Out[13]:

	Date & Time	use [kW]	gen [kW]	Grid [kW]	AC [kW]	Furnace [kW]	Cellar Lights [kW]	Washer [kW]	First Floor lights [kW]	Utilit Base Bath
0	2014-01-01 00:00:00	0.304439	0.0	0.304439	0.000058	0.009531	0.005336	0.000126	0.011175	0.00
1	2014-01-01 00:30:00	0.656771	0.0	0.656771	0.001534	0.364338	0.005522	0.000043	0.003514	0.0C
2	2014-01-01 01:00:00	0.612895	0.0	0.612895	0.001847	0.417989	0.005504	0.000044	0.003528	0.00
3	2014-01-01 01:30:00	0.683979	0.0	0.683979	0.001744	0.410653	0.005556	0.000059	0.003499	0.00
4	2014-01-01 02:00:00	0.197809	0.0	0.197809	0.000030	0.017152	0.005302	0.000119	0.003694	0.00

In [14]:
```
#usekw_data is data frame of the sum of the energy usage(Use [kW]) per day (grou
usekw_data = energy_data.groupby('Date').sum()
usekw_data = usekw_data[['use [kW]']]
usekw_data.head()
```

Out[14]:

	use [kW]
Date	
2014-01-01	65.013592
2014-01-02	32.305336
2014-01-03	31.164468
2014-01-04	45.287782
2014-01-05	36.316643

In [15]:

```
#In weather_data, the column 'time' is represented as second, so we should conve
#I added new column "Date" and put converted value into it.

weather_data['Date'] = pd.to_datetime(weather_data['time'], unit='s').dt.date

#grouby 'Date' and take average of rows of other columns

new_weather_data = weather_data.groupby('Date').mean()
weather_data = new_weather_data.drop(columns=['time'])
```

In [16]:

```
weather_data.head()
```

Out[16]:

	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windBearing	prec
Date								
2014-01-01	20.110833	0.556667	9.970000	1025.395000	6.820417	0.031304	252.291667	
2014-01-02	16.382500	0.784583	3.834583	1023.465833	7.433750	0.354444	53.458333	
2014-01-03	6.256667	0.680833	4.509167	1014.428750	12.828333	0.186364	207.333333	
2014-01-04	2.711667	0.617083	9.822917	1030.096250	5.248333	0.001667	240.166667	
2014-01-05	17.654167	0.682083	9.134583	1025.275000	3.417083	0.010952	208.958333	

In [17]:

```
#merge two dataframe on 'Date'
data = pd.merge(usekw_data,weather_data, on = 'Date', how = "left")
```

In [18]:

```
data.head()
```

Out[18]:

Date	use [kW]	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windB
2014-01-01	65.013592	20.110833	0.556667	9.970000	1025.395000	6.820417	0.031304	252.:
2014-01-02	32.305336	16.382500	0.784583	3.834583	1023.465833	7.433750	0.354444	53.4
2014-01-03	31.164468	6.256667	0.680833	4.509167	1014.428750	12.828333	0.186364	207.3
2014-01-04	45.287782	2.711667	0.617083	9.822917	1030.096250	5.248333	0.001667	240.1
2014-01-05	36.316643	17.654167	0.682083	9.134583	1025.275000	3.417083	0.010952	208.9

In [19]:

```
#Task 2
#Split the data obtained from step 1, into training and testing sets. The aim is
#for each day in the month of December using the weather data, so split accordin
#per devices should be dropped, only the "use [kW]" column is to be used for pre

#we are going to consider training set as data from January to November, and tes

#training_set
training_set = data[data.index < datetime.date(2014,12,1)]
training_set
```

Out[19]:

Date	use [kW]	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windB
2014-01-01	65.013592	20.110833	0.556667	9.970000	1025.395000	6.820417	0.031304	252.:
2014-01-02	32.305336	16.382500	0.784583	3.834583	1023.465833	7.433750	0.354444	53.4
2014-01-03	31.164468	6.256667	0.680833	4.509167	1014.428750	12.828333	0.186364	207.3
2014-01-04	45.287782	2.711667	0.617083	9.822917	1030.096250	5.248333	0.001667	240.1
2014-01-05	36.316643	17.654167	0.682083	9.134583	1025.275000	3.417083	0.010952	208.9
...	...	...	...	...	...	...	...	
2014-11-26	27.712850	36.385000	0.778333	6.551667	1019.266250	6.445833	0.171333	185.3
2014-11-27	30.114004	31.992500	0.847083	7.394583	1012.272917	7.599167	0.420769	316.8

Date	use [kW]	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windB
2014-11-28	26.348404	29.126250	0.763750	8.919167	1018.359583	6.599167	0.268947	316.4
2014-11-29	20.241298	22.344583	0.706667	9.793750	1025.543750	4.299167	0.049167	230.3
2014-11-30	32.239043	36.430000	0.730000	9.826250	1021.495000	5.782917	0.202667	185.7

334 rows × 11 columns

In [20]:

```
#testing set
testing_set = data[data.index>= datetime.date(2014,12,1)]
testing_set
```

Out[20]:

Date	use [kW]	temperature	humidity	visibility	pressure	windSpeed	cloudCover	wind
2014-12-01	30.550010	45.276250	0.722083	9.656667	1018.805417	6.397083	0.263333	226.
2014-12-02	31.748857	34.177917	0.582917	9.839583	1034.805833	7.527083	0.121818	166.
2014-12-03	28.773233	36.345833	0.911250	4.939167	1022.247500	5.691250	0.862000	119.
2014-12-04	39.484491	36.216250	0.584167	9.976667	1024.064583	9.129583	0.130000	286
2014-12-05	33.342503	27.463750	0.698750	9.847083	1035.654167	3.421667	0.069130	63.
2014-12-06	36.470153	34.868750	0.909167	4.692500	1026.207500	3.397083	0.862000	117
2014-12-07	26.486585	33.502917	0.641667	9.490417	1029.725000	12.755417	0.170952	50.
2014-12-08	23.013980	19.519583	0.562917	9.980833	1039.599583	8.700000	0.062105	15.
2014-12-09	27.954351	30.960417	0.857500	6.005417	1023.523333	10.067500	1.000000	20.
2014-12-10	37.422625	36.709583	0.911250	3.816250	1001.643750	9.912083	1.000000	293.
2014-12-11	35.182712	31.840833	0.839583	6.198333	1001.585833	6.097917	0.743333	263
2014-12-12	24.209088	30.110833	0.752500	9.763333	1011.360833	7.061667	0.288333	285.
2014-12-13	20.455440	32.049583	0.711250	9.932500	1012.691250	8.107083	0.199524	313.
2014-12-14	19.821203	32.943750	0.768750	9.749583	1011.030833	6.575417	0.089167	319.

Date	use [kW]	temperature	humidity	visibility	pressure	windSpeed	cloudCover	wind
2014-12-15	41.912526	35.535833	0.724167	9.970417	1016.019167	5.793333	0.144167	321.
2014-12-16	20.712163	30.293333	0.867083	8.840833	1019.601250	2.194167	0.182500	87.
2014-12-17	21.802123	39.820833	0.876250	7.020000	1010.482083	5.465833	0.527273	237.
2014-12-18	19.836075	37.259583	0.695417	9.510417	1010.080417	11.019167	0.348500	289.
2014-12-19	32.802819	32.919583	0.648750	10.000000	1014.558750	9.645833	0.462222	308.
2014-12-20	34.296287	27.155833	0.699167	9.997083	1023.671667	5.012917	0.114167	231
2014-12-21	21.058376	30.712500	0.827917	7.354583	1026.219583	5.300417	0.794000	38.
2014-12-22	27.362027	34.197500	0.842083	9.162917	1027.610833	4.317917	0.660000	24.
2014-12-23	19.387136	38.825417	0.895417	6.885000	1023.628750	5.342500	1.000000	31.
2014-12-24	27.682246	40.647083	0.921667	4.729167	1016.663750	5.624583	1.000000	21.
2014-12-25	40.268132	47.635833	0.766250	6.784583	1002.865000	8.596250	0.330714	226.
2014-12-26	44.563400	42.025000	0.577083	9.990417	1019.035000	9.256250	0.092083	282.
2014-12-27	35.046127	35.487083	0.756250	9.246250	1022.081667	3.677083	0.030417	243
2014-12-28	37.695824	41.892917	0.763750	9.332917	1013.549167	6.587917	0.245909	224.
2014-12-29	28.675929	34.728333	0.592083	9.997083	1018.870833	8.129583	0.119167	281.
2014-12-30	31.514313	24.846667	0.488750	9.998333	1026.102083	7.566667	0.031250	312
2014-12-31	28.674498	19.522917	0.552917	9.986250	1025.940833	5.943750	0.117917	260.

In [21]:

```
#Task 3
#Linear Regression - Predicting Energy Usage:
#Set up a simple linear regression model to train, and then predict energy usage
#month of December using features from weather data (Note that you need to drop
#column in the test set first). How well/badly does the model work? (Evaluate th
#your predictions based on the original “use [kW]” column). Calculate the Root m
#of your model.
#Finally generate a csv dump of the predicted values. Format of csv: Two columns
#the date and second should be the predicted value

#first of all, drop the "use [kW] in the test set"
```

```
testingset_usekw = testing_set[['use [kW]']]

testing_set = testing_set.drop(columns=['use [kW]'])
testing_set.head()
```

Out[21]:

Date	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windBearing	prec
2014-12-01	45.276250	0.722083	9.656667	1018.805417	6.397083	0.263333	226.958333	
2014-12-02	34.177917	0.582917	9.839583	1034.805833	7.527083	0.121818	166.625000	
2014-12-03	36.345833	0.911250	4.939167	1022.247500	5.691250	0.862000	119.333333	
2014-12-04	36.216250	0.584167	9.976667	1024.064583	9.129583	0.130000	286.125000	
2014-12-05	27.463750	0.698750	9.847083	1035.654167	3.421667	0.069130	63.833333	

In [22]:

```
#testingset_usekw is the testing set that only contains the "use [kW]" column
testingset_usekw.head()
```

Out[22]:

Date	use [kW]
2014-12-01	30.550010
2014-12-02	31.748857
2014-12-03	28.773233
2014-12-04	39.484491
2014-12-05	33.342503

In [23]:

```
#In order to create Linear regression model, we also have to split up the traini
trainingset_usekw = training_set[['use [kW]']]
training_set =  training_set.drop(columns=['use [kW]'])
training_set.head()
```

Out[23]:

Date	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windBearing	prec
2014-01-01	20.110833	0.556667	9.970000	1025.395000	6.820417	0.031304	252.291667	
2014-01-02	16.382500	0.784583	3.834583	1023.465833	7.433750	0.354444	53.458333	
2014-01-03	6.256667	0.680833	4.509167	1014.428750	12.828333	0.186364	207.333333	
2014-01-	2.711667	0.617083	9.822917	1030.096250	5.248333	0.001667	240.166667	

	temperature	humidity	visibility	pressure	windSpeed	cloudCover	windBearing	prec
Date								
04								
2014-01-05	17.654167	0.682083	9.134583	1025.275000	3.417083	0.010952	208.958333	

In [24]:
```
trainingset_usekw.head()
```

Out[24]:

	use [kW]
Date	
2014-01-01	65.013592
2014-01-02	32.305336
2014-01-03	31.164468
2014-01-04	45.287782
2014-01-05	36.316643

In [25]:
```
#creat Linear Regression Model of training set and predicted usage
linearReg = LinearRegression()
linearReg.fit(training_set, trainingset_usekw)
predicted_usage = linearReg.predict(testing_set)
```

In [26]:
```
predicted_df = testingset_usekw.copy()
predicted_df['predicted use [kW]'] = predicted_usage
predicted_df = predicted_df.drop(columns=['use [kW]'])
predicted_df
```

Out[26]:

	predicted use [kW]
Date	
2014-12-01	30.434573
2014-12-02	31.655605
2014-12-03	18.306381
2014-12-04	31.435899
2014-12-05	23.818158
2014-12-06	21.346389
2014-12-07	22.959651
2014-12-08	24.902482
2014-12-09	20.058072
2014-12-10	18.536161
2014-12-11	19.507100

	predicted use [kW]
Date	
2014-12-12	21.959088
2014-12-13	25.625546
2014-12-14	24.562402
2014-12-15	27.906889
2014-12-16	17.042749
2014-12-17	23.646114
2014-12-18	26.085201
2014-12-19	25.600495
2014-12-20	25.383613
2014-12-21	15.022010
2014-12-22	13.784885
2014-12-23	14.203930
2014-12-24	16.896650
2014-12-25	30.401745
2014-12-26	34.002892
2014-12-27	26.726799
2014-12-28	27.750191
2014-12-29	30.477621
2014-12-30	29.754506
2014-12-31	25.711734

In [27]:

```
#calculate root mean squared error
rmse = root_mean_squared_error(testingset_usekw, predicted_df)
rmse
```

Out[27]: 8.740566311137641

The value of root mean squared error is 8.74056631136634. This is never small value if we consider the range of value of usage. We can conclude that this linear regression model works badly.

In [28]:

```
#create csv file of predicted usage
predicted_df.to_csv('linear_regression.csv')
```

In [29]:

```
#. Logistic Regression - Temperature classification:
#Using only weather data we want to classify if the temperature is high or low.
#temperature greater than or equal to 35 is ‘high’ and below 35 is ‘low’. Set up
#model to classify the temperature for each day in the month of December. Calcul
#for the model.
```

```
#Finally generate a csv dump of the classification (1 for high, 0 for low)
#Format: Two columns, first should be the date and second should be the classifi

task4_data = weather_data.copy()
task4_data['temp classification'] = task4_data['temperature'].apply(lambda x: 1
task4_data.drop(columns=['temperature'])
```

Out[29]:

	humidity	visibility	pressure	windSpeed	cloudCover	windBearing	precipIntensity	(
Date								
2014-01-01	0.556667	9.970000	1025.395000	6.820417	0.031304	252.291667	0.000000	
2014-01-02	0.784583	3.834583	1023.465833	7.433750	0.354444	53.458333	0.002004	1
2014-01-03	0.680833	4.509167	1014.428750	12.828333	0.186364	207.333333	0.002029	-
2014-01-04	0.617083	9.822917	1030.096250	5.248333	0.001667	240.166667	0.000000	-
2014-01-05	0.682083	9.134583	1025.275000	3.417083	0.010952	208.958333	0.000033	
...	...	...	...	...	...	...	...	
2014-12-27	0.756250	9.246250	1022.081667	3.677083	0.030417	243.791667	0.000000	2
2014-12-28	0.763750	9.332917	1013.549167	6.587917	0.245909	224.458333	0.003996	3
2014-12-29	0.592083	9.997083	1018.870833	8.129583	0.119167	281.833333	0.000000	2
2014-12-30	0.488750	9.998333	1026.102083	7.566667	0.031250	312.041667	0.000000	
2014-12-31	0.552917	9.986250	1025.940833	5.943750	0.117917	260.083333	0.000000	

365 rows × 10 columns

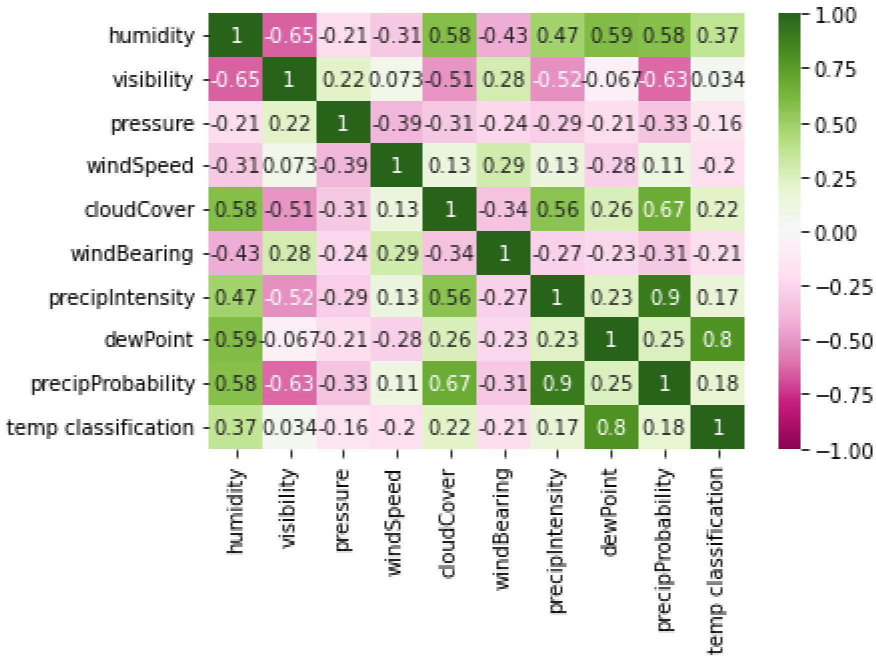
In [30]:

```
#setting up the training set and testing set

task4_training_set = training_set.copy()
task4_testing_set = testing_set.copy()
task4_training_set['temp classification'] = task4_training_set['temperature'].ap
task4_training_set = task4_training_set.drop(columns=['temperature'])
task4_testing_set['temp classification'] = task4_testing_set['temperature'].appl
task4_testing_set = task4_testing_set.drop(columns=['temperature'])
```

In [31]:

```
#Before creating logistic regression model, it is good to choose appropriate fea
#check correlation
heatmap=sns.heatmap(task4_training_set.corr(), vmin=-1, vmax=1, annot=True, cmap
```



It shows that "humidity" and "dew point" are highly correlated with "temp classfication", so we can use them as training and testing features

In [32]:
```
task4_training_features = task4_training_set[['humidity', 'dewPoint']]
task4_training_label = task4_training_set[['temp classification']]

task4_testing_features =task4_testing_set[['humidity','dewPoint']]
task4_testing_label = task4_testing_set[['temp classification']]
```

In [33]:
```
#scaling the training and testing features
scaler = StandardScaler()
task4_training_features = scaler.fit_transform(task4_training_features)
task4_testing_features = scaler.transform(task4_testing_features)
```

In [34]:
```
#build logistic regression model
logisticReg = LogisticRegression()
logisticReg.fit(task4_training_features, task4_training_label.values.ravel())
task4_predicted = logisticReg.predict(task4_testing_features)
```

In [35]:
```
#calculate the F1 score of the logistic regression model
f1_score(task4_testing_label,task4_predicted)
```

Out[35]:
```
0.8125
```

In [36]:
```
#create csv dump of the classification
task4_predicted_df = task4_testing_label.copy()
task4_predicted_df['temp classification'] = task4_predicted
task4_predicted_df.to_csv('logistic_regression.csv')
```

In [37]:
```
#Task 5
# Energy usage data Analysis:
```

```
#We want to analyze how different devices are being used in different times of t
#- Is the washer being used only during the day?
#- During what time of the day is AC used most?
#There are a number of questions that can be asked.
#For simplicity, let’s divide a day in two parts:
#- Day: 6AM - 7PM
#- Night: 7PM - 6AM
#Analyze the usage of any two devices of your choice during the ‘day’ and ‘night


#I added the column "Time" and "Day or Night" in order to classify Day and Night
#I set Day as 1 and and Night as 0
Task5_data = energy_data.copy()
Task5_data['Time'] = pd.to_datetime(Task5_data['Date & Time']).dt.time
Task5_data['Day or Night'] = Task5_data['Time'].apply(lambda x: 1 if x>=datetime
```

In [38]:

```
#First question = During what time of the day is AC used Most?

AC_day = Task5_data[Task5_data['Day or Night'] == 1]
AC_day = AC_day[['AC [kW]']]
AC_usage_during_day = AC_day['AC [kW]'].sum()

AC_night = Task5_data[Task5_data['Day or Night']==0]
AC_night = AC_night[['AC [kW]']]
AC_usage_during_night = AC_night['AC [kW]'].sum()
```

In [39]:

```
AC_usage_during_day
```

Out[39]:

```
382.094051674
```

In [40]:

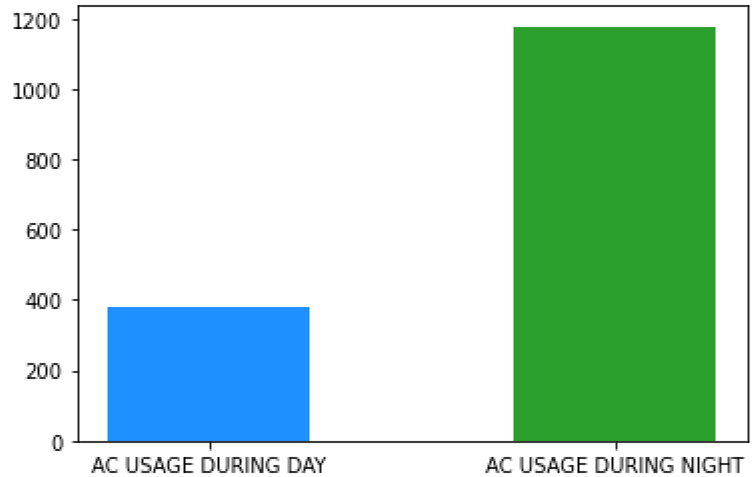
```
AC_usage_during_night
```

Out[40]:

```
1177.164341666
```

In [41]:

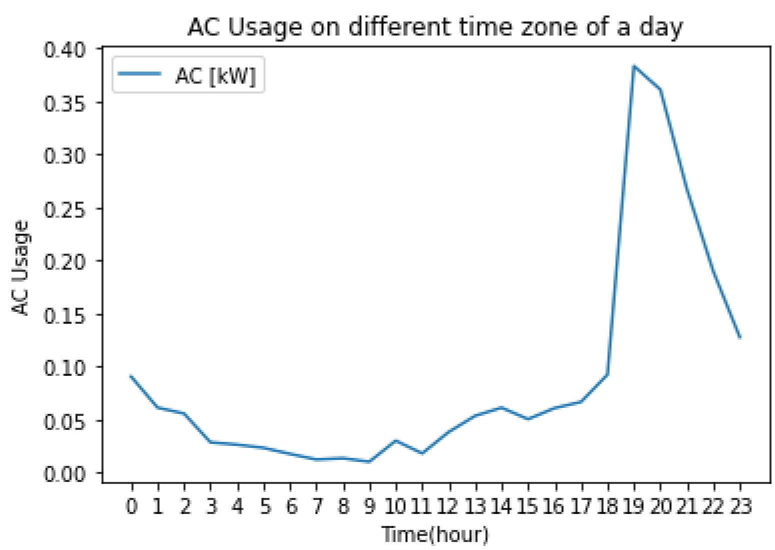
```
x = np.arange(2)
name = ['AC USAGE DURING DAY','AC USAGE DURING NIGHT']
values = [AC_usage_during_day, AC_usage_during_night]
plt.bar(x, values,width=0.5,color=['dodgerblue','C2'])
plt.xticks(x, name)
plt.show()
```



The bar graph shows that AC usage during night is much higher than AC usage during night. I thought people would use AC more during day because temperature usually go to the highest between 12 and 2pm. To investigate why my assumption is wrong, I will create another graph that represents AC usage on different time zone of a day.

In [42]:
```
AC_timezone = Task5_data.copy()
AC_timezone['Hour'] = pd.to_datetime(AC_timezone['Date & Time']).dt.hour
AC_timezone = AC_timezone.groupby('Hour').mean()
AC_timezone = AC_timezone[['AC [kW]']]

AC_timezone.plot()
plt.title("AC Usage on different time zone of a day")
plt.xlabel("Time(hour)")
plt.ylabel("AC Usage")
plt.xticks(range(0,24))
plt.show()
```



The graph is showing that AC usage is rapidly increasing from 6 pm until 8 pm. I think the energy data is based on a regular household because people aren't usually at home durign day for work, school, and etc., and they are less likely to use AC that time. Also, people usually come back from work or school after 5 pm, and they start to turn on AC during evening time. This is why AC usage during day time is much higher than AC usage during night time.

In [43]:
```
#Second question = Is the washer being used only during the day?
washer_day = Task5_data[Task5_data['Day or Night'] == 1]
washer_day = washer_day[['Washer [kW]']]
washer_usage_during_day = washer_day['Washer [kW]'].sum()

washer_night = Task5_data[Task5_data['Day or Night']==0]
washer_night = washer_night[['Washer [kW]']]
washer_usage_during_night = washer_night['Washer [kW]'].sum()
```

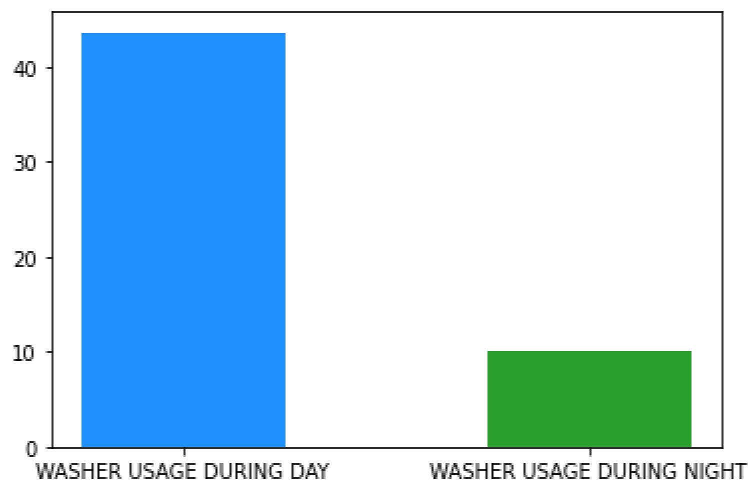
In [44]:
```
washer_usage_during_day
```

Out[44]: 43.634362779

In [45]:
```
washer_usage_during_night
```

Out[45]: 10.101531092999998

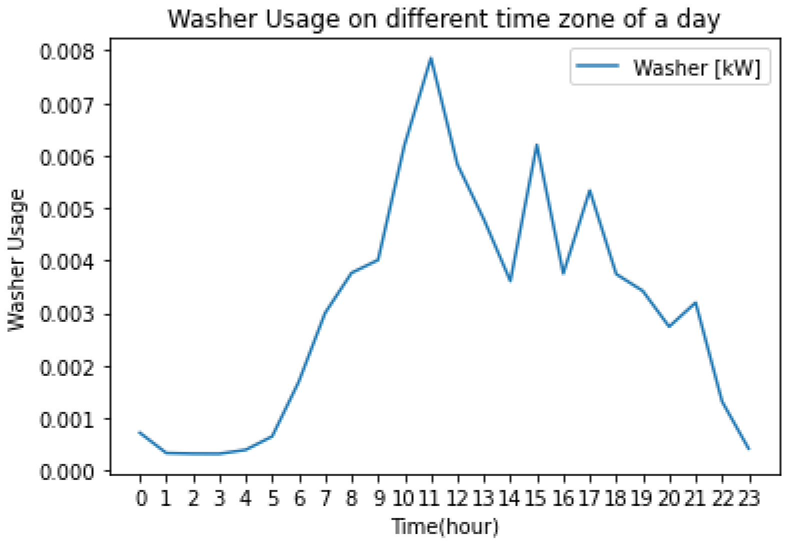
In [46]:
```
x = np.arange(2)
name = ['WASHER USAGE DURING DAY','WASHER USAGE DURING NIGHT']
values = [washer_usage_during_day, washer_usage_during_night]
plt.bar(x, values,width=0.5,color=['dodgerblue','C2'])
plt.xticks(x, name)
plt.show()
```



The bar graph shows that washer is mostly being used during a day. This is reasonable becasue some washers cause a huge noise and may interrupt people who are trying to sleep. Even though washer is mostly being used during a day, we still can see some usage during night, and now we are going to look at the graph that represents washer usage on different time zone of a day.

In [47]:
```
washer_timezone= Task5_data.copy()
washer_timezone['Hour'] = pd.to_datetime(washer_timezone['Date & Time']).dt.hour
washer_timezone = washer_timezone.groupby('Hour').mean()
washer_timezone = washer_timezone[['Washer [kW]']]
```

In [48]:
```
washer_timezone.plot()
plt.title("Washer Usage on different time zone of a day")
plt.xlabel("Time(hour)")
plt.ylabel("Washer Usage")
plt.xticks(range(0,24))
plt.show()
```



As shown above, we can see some washer usuage between 8 and 9 pm. I think it is because people who came back from work or school need to do laundry to prepare clean clothes for the next day.