TOPIC : Forecasting and Load Management

We are using energy dataset for forecasting energy consumption on household appliances.

60009210029:- Purvi Parmar

60009210030:- Miloni Shah

Here We have choose linear regression for modeling because Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.Linear regression is commonly used for predictive analysis and modeling.

Firstly we are going to process and clean our data,these two steps are very important for to be able to ready our data for machine learning algorithm.

```
import pandas as pd
```

Reading Data

```
df= pd.read_csv('/content/energydata_complete.csv', parse_dates=['date'])
```

```
df.head()
```

	date	Appliances	lights	T1	RH_1	T2	RH_2	T3	RH_3	T4	.
0	2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19.000000	
1	2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19.000000	
2	2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18.926667	
3	2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18.890000	
4	2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18.890000	

5 rows × 29 columns

Here we have set date as index because later on in our processing and when we apply algorithm we are going to do forecasting of the use of energy and in which case dates are very important, so we mentioned we are going to indentify the kind of interval that could reason the impact of the noise and converted all columns name to lower case

```
df.columns=[x.lower() for x in df.columns]
```

```
df=df.set_index('date')
```

```
df.head()
```

date	appliances	lights	t1	rh_1	t2	rh_2	t3	rh_3	
2016-01-11 17:00:00	60	30	19.89	47.596667	19.2	44.790000	19.79	44.730000	19
2016-01-11 17:10:00	60	30	19.89	46.693333	19.2	44.722500	19.79	44.790000	19
2016-01-11 17:20:00	50	30	19.89	46.300000	19.2	44.626667	19.79	44.933333	18
2016-01-11 17:30:00	50	40	19.89	46.066667	19.2	44.590000	19.79	45.000000	18
2016-01-11 17:40:00	60	40	19.89	46.333333	19.2	44.530000	19.79	45.000000	18

5 rows × 28 columns

To know how many columns and rows are there in the dataset. We use The Python numpy module has a shape function, which helps us to find the shape or size of an array or matrix.There are 19735 rows and 29 columns in the dataset.

```
df.shape
```

```
(19735, 28)
```

Having an overview structure of the data.In this step we are identifying whether data has null value or not . isnull() is used to know the total number of null values in the dataset.

```
df.isnull().sum()
```

```
appliances     0
lights         0
t1             0
```

```
rh_1           0
t2             0
rh_2           0
t3             0
rh_3           0
t4             0
rh_4           0
t5             0
rh_5           0
t6             0
rh_6           0
t7             0
rh_7           0
t8             0
rh_8           0
t9             0
rh_9           0
t_out          0
press_mm_hg    0
rh_out         0
windspeed      0
visibility     0
tdewpoint      0
rv1            0
rv2            0
dtype: int64
```

There is no null value in our dataset

To know the spread of our data ,descriptive percentage . The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame.

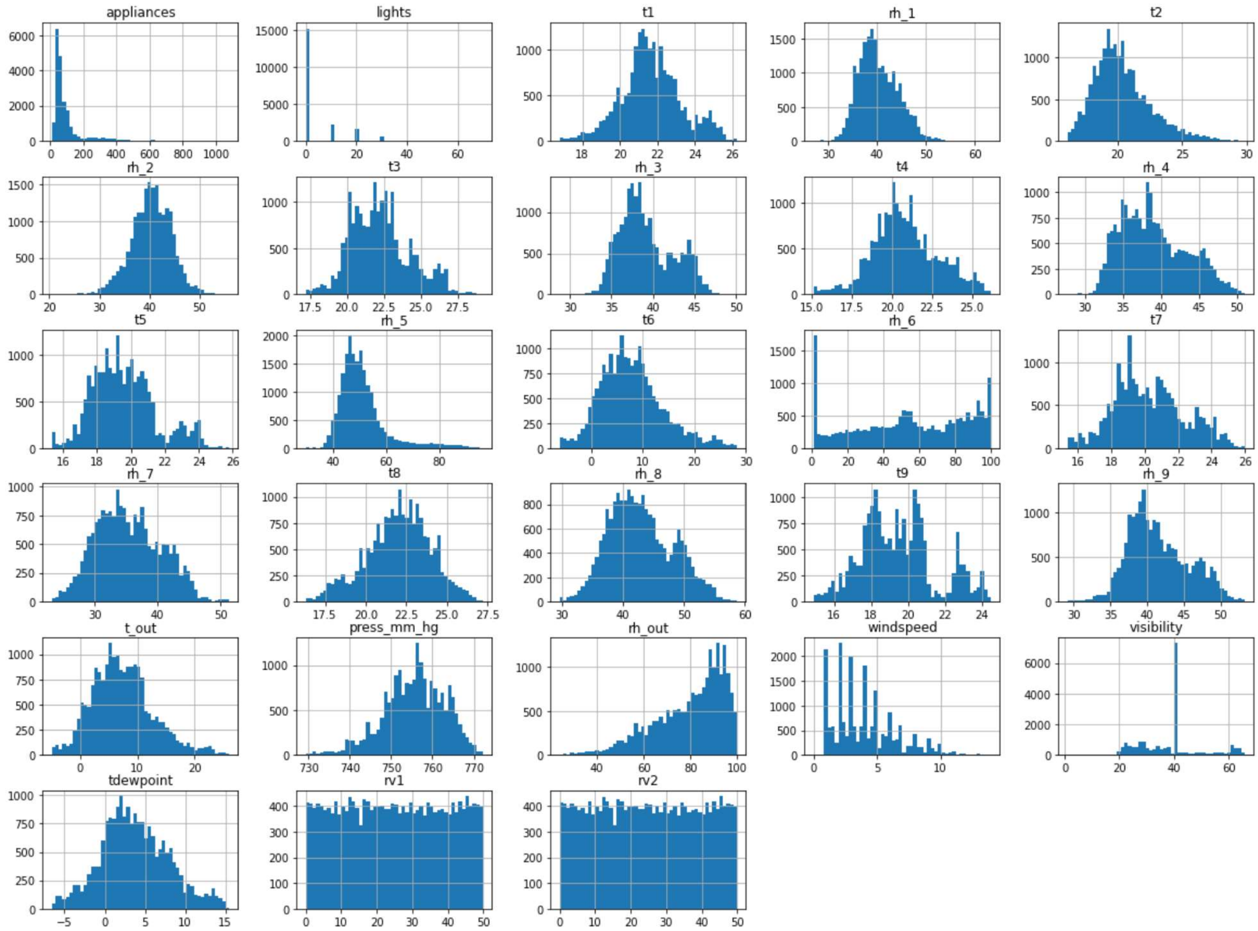
```
df.describe()
```

	appliances	lights	t1	rh_1	t2	
count	19735.000000	19735.000000	19735.000000	19735.000000	19735.000000	19735.0
mean	97.694958	3.801875	21.686571	40.259739	20.341219	40.4
std	102.524891	7.935988	1.606066	3.979299	2.192974	4.0
min	10.000000	0.000000	16.790000	27.023333	16.100000	20.4
25%	50.000000	0.000000	20.760000	37.333333	18.790000	37.9
50%	60.000000	0.000000	21.600000	39.656667	20.000000	40.5
75%	100.000000	0.000000	22.600000	43.066667	21.500000	43.2
max	1080.000000	70.000000	26.260000	63.360000	29.856667	56.0

8 rows × 28 columns

Here we have plotted the numerical distribution of each of our variable and here we could see that each variable has its own distribution so for appliacnces it is skewed to the left. What is the significance of having skewedness? The significance of having similarities of the skewedness or skewness of the different variable is that we can just select among them which one that can properly be used for our modeling.

```
import matplotlib.pyplot as plt
df.hist(bins=50, figsize=(20,15))
plt.savefig("Attribute Histogram Plots")
plt.show()
```



Here we could see the different correlations of the variables with each other so as you could see here in the diagonal its one its because its the correlation of the variable with itself of course that will always give us one so appliances with respect to lights we could see 0.198201.

```
df.corr()
```

	appliances	lights	t1	rh_1	t2	rh_2	t
appliances	1.000000	0.198201	0.058996	0.087890	0.122590	-0.058680	0.08821
lights	0.198201	1.000000	-0.022727	0.107266	-0.004990	0.051428	-0.09639
t1	0.058996	-0.022727	1.000000	0.163976	0.836827	-0.002565	0.89242
rh_1	0.087890	0.107266	0.163976	1.000000	0.269801	0.797675	0.25318
t2	0.122590	-0.004990	0.836827	0.269801	1.000000	-0.165586	0.73519
...	...	...	...	...	...	...	.
19.0	0.163705	0.068885	0.061668	0.077313	0.056358	-0.019359	0.01979
20.0	0.060674	0.151187	0.081955	0.038137	0.058432	-0.002817	0.02181
21.0	0.036430	0.143529	0.089179	-0.003082	0.053618	-0.002587	0.01614
22.0	-0.028430	0.080350	0.085175	-0.019701	0.039029	0.001208	0.01283
23.0	-0.082900	0.053546	0.069758	-0.021405	0.013577	0.013619	0.00362

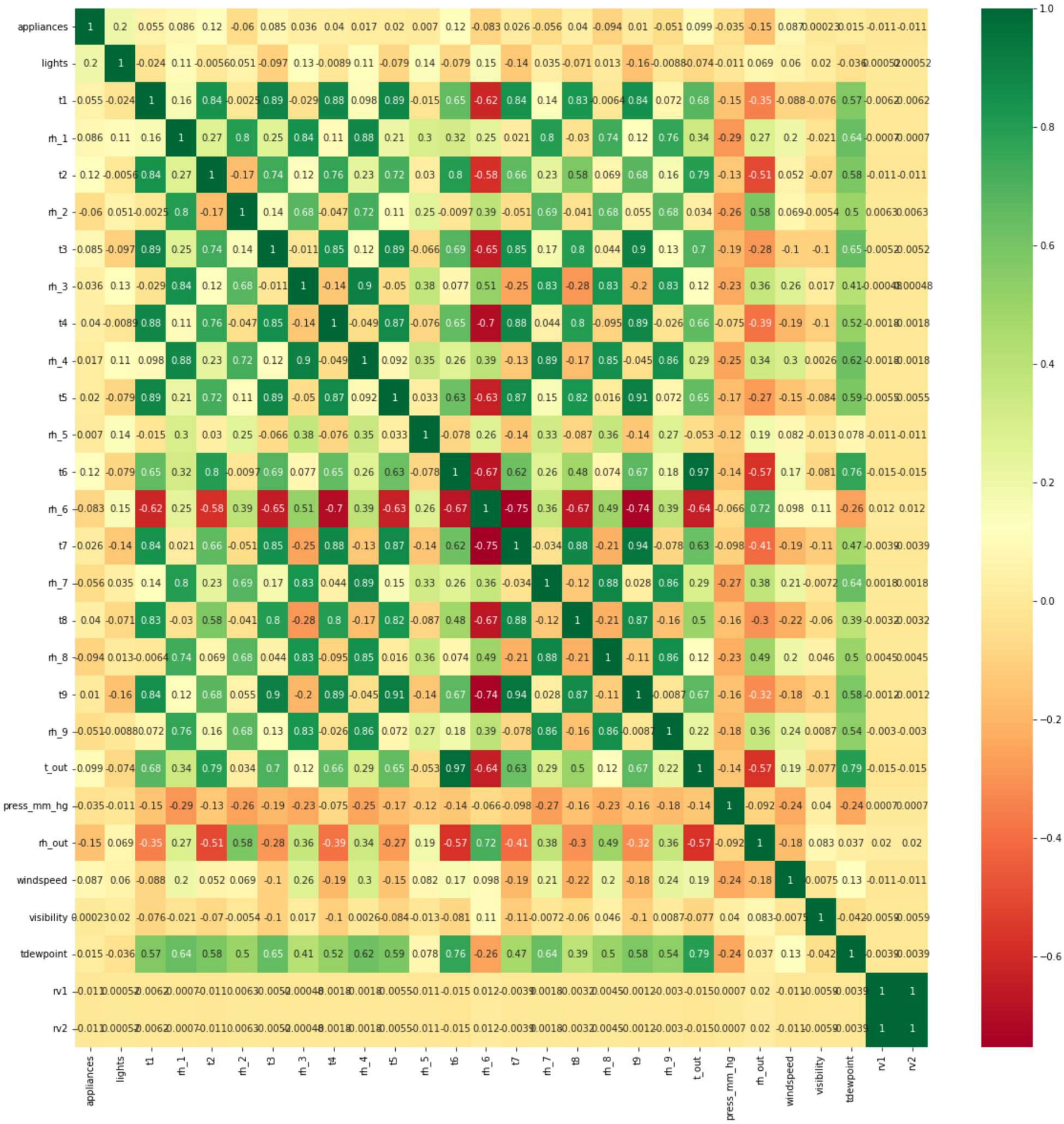
78 rows × 78 columns

```
import seaborn as sns
```

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

Heatmap of the correlations darkest green here means the number one which is positively correlated and darkest red here means that there is a perfect negative correlation

```
corrmap = df.corr()
top_corr_features = corrmap.index
plt.figure(figsize=(20,20)) #size of the heatmap
df_heatmap = sns.heatmap(df[top_corr_features].corr(), annot=True,cmap="RdYlGn")
```



Sorting the value of energy consumption of appliances in descending order.The highest is 1080 W

```
sorted_appliances=df.sort_values('appliances', ascending=False)
sorted_appliances.head()
```

	appliances	lights	t1	rh_1	t2	rh_2	t3
date							
2016-01-16 18:50:00	1080	30	21.930000	42.766667	21.040000	38.080000	20.700000
2016-01-21 18:50:00	1070	30	19.600000	34.300000	18.426667	33.963333	18.390000
2016-01-14 17:00:00	910	0	21.463333	41.693333	20.856667	38.363333	21.666667
2016-04-04 15:40:00	900	0	23.000000	43.166667	22.200000	40.426667	26.100000
2016-01-21 19:00:00	890	20	19.730000	37.863333	18.566667	34.090000	18.390000

5 rows × 28 columns

Only 1% of values we are going to considered as outliers.How we are going to identify the 1% of value ,the code is given below.

```
len(sorted_appliances.head(len(sorted_appliances)//1000))
```

```
19
```

So when we do the execution of the code it will give 19,means that 1% of the values out of 19000 plus values of the engery consumed is 19.

Here we are going to see What is the value of 19th place in our data,because the 19th place would be our baseline for us to be able to identify whether or not we are going to go for more than or less than but in this case we are going to go for more than because we are actually looking for higher value of certain boundary.

```
sorted_appliances.appliances[19]
```

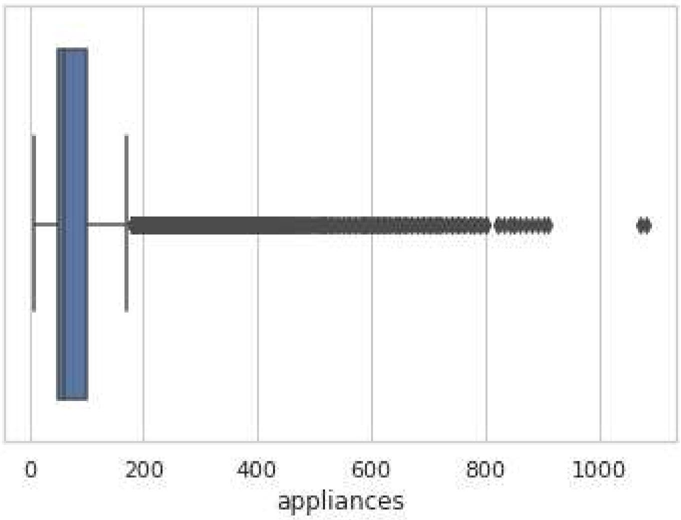
```
790
```

So here we have 790 as the base line of our value so that means to say that above 790 would be considered outliers.

To identify and present these outliers we are going to show them using boxplot for appliances

```
sorted_appliances=df.sort_values('appliances',ascending=False)
print("The number of the 0, 1% top values of appliances' load is",len(sorted_appliances.head(len(sor
#boxplot appliances
sns.set(style="whitegrid")
ax= sns.boxplot(sorted_appliances.appliances)
```

```
The number of the 0, 1% top values of appliances' load is 19 and they have power l
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: P
  FutureWarning
```



Droping outliers having value greater than 790 and values less 0.

```
df=df.dropna()
df=df.drop(df[(df.appliances>790)| (df.appliances<0)].index)
```

```
import warnings
warnings.filterwarnings("ignore")
```

Indexing different elements.Created new feature for our dataset.

```
df['hour']=df.index.hour
df['week']=df.index.week
df['weekday']=df.index.weekday
df['month']=df.index.month
```

```
import numpy as np
df['log_appliances']=np.log(df.appliances)
```

Taking Averages of house temperature and house humidity.We have not consider t6 and rh6 because they are the values outside the building and here we are not going to consider the values outside the house so we only have 8 feature to consider

```
df['house_temp']=(df.t1+df.t2+df.t3+df.t4+df.t5+df.t7+df.t8+df.t9)/8
df['house_hum']=(df.rh_1+df.rh_2+df.rh_3+df.rh_4+df.rh_5+df.rh_7+df.rh_8+df.rh_9)/8
```

```
df['house_temp'].head()
```

```
date
2016-01-11 17:00:00    18.435000
2016-01-11 17:10:00    18.439167
2016-01-11 17:20:00    18.421667
2016-01-11 17:30:00    18.396250
2016-01-11 17:40:00    18.408750
Name: house_temp, dtype: float64
```

```
df['house_hum'].head()
```

```
date
2016-01-11 17:00:00    46.742500
2016-01-11 17:10:00    46.672708
2016-01-11 17:20:00    46.562917
2016-01-11 17:30:00    46.468750
2016-01-11 17:40:00    46.462917
Name: house_hum, dtype: float64
```

Remove Additive assumptions .

```
#Remove Additive assumptions
df['hour*lights']=df.hour*df.lights
df['t3rh3']=df.t3 * df.rh_3
df['t2rh2']=df.t3 * df.rh_2
df['t1rh1']=df.t3 * df.rh_1
df['t4rh4']=df.t3 * df.rh_4
df['t5rh5']=df.t3 * df.rh_5
df['t6rh6']=df.t3 * df.rh_6
df['t7rh7']=df.t3 * df.rh_7
df['t8rh8']=df.t3 * df.rh_8
df['t9rh9']=df.t3 * df.rh_9
```

Calculating avearge energy load per weekly and per hour

```
def code_mean(data,cat_feature, real_feature):
    return dict(data.groupby(cat_feature)[real_feature].mean())
```

```
df['weekday_avg']=list(map(
    code_mean(df[:], 'weekday', "appliances").get, df.weekday))
df['hour_avg']=list(map(
    code_mean(df[:], 'hour', "appliances").get, df.hour))
```

```
df['weekday_avg'].head()
```

```
date
2016-01-11 17:00:00    110.896974
2016-01-11 17:10:00    110.896974
2016-01-11 17:20:00    110.896974
2016-01-11 17:30:00    110.896974
2016-01-11 17:40:00    110.896974
Name: weekday_avg, dtype: float64
```

```
df['hour_avg'].head()
```

```
date
2016-01-11 17:00:00    158.812121
2016-01-11 17:10:00    158.812121
2016-01-11 17:20:00    158.812121
2016-01-11 17:30:00    158.812121
2016-01-11 17:40:00    158.812121
Name: hour_avg, dtype: float64
```

The intervals of our values is 10 mins so as you could see here we have 0, 10 ,20 ,so we are not going to use these kind of interval but instead we are going to use different interval the reason for this is that as much as possible what we are going to do is that we are going to lesson the impact of the noise for that we are going to consider 30 min and 1 hr interval.

```
df_hour=df.resample('1H').mean()
df_30min=df.resample('30min').mean()
```

```
df_hour.head()
```

date	appliances	lights	t1	rh_1	t2	rh_2	t
2016-01-11 17:00:00	55.000000	35.000000	19.890000	46.502778	19.200000	44.626528	19.7900(
2016-01-11 18:00:00	176.666667	51.666667	19.897778	45.879028	19.268889	44.438889	19.7700(
2016-01-11 19:00:00	173.333333	25.000000	20.495556	52.805556	19.925556	46.061667	20.0522
2016-01-11 20:00:00	125.000000	35.000000	20.961111	48.453333	20.251111	45.632639	20.2138
2016-01-11 21:00:00	103.333333	23.333333	21.311667	45.768333	20.587778	44.961111	20.3733

5 rows × 47 columns

```
df_30min.head()
```

date	appliances	lights	t1	rh_1	t2	rh_2	t
2016-01-11 17:00:00	56.666667	30.000000	19.890000	46.863333	19.200000	44.713056	19.7900(
2016-01-11 17:30:00	53.333333	40.000000	19.890000	46.142222	19.200000	44.540000	19.7900(
2016-01-11 18:00:00	60.000000	46.666667	19.845556	45.641389	19.200000	44.477778	19.7500(
2016-01-11 18:30:00	293.333333	56.666667	19.950000	46.116667	19.337778	44.400000	19.7900(
2016-01-11 19:00:00	260.000000	33.333333	20.273333	52.206667	19.717778	45.111111	19.9377

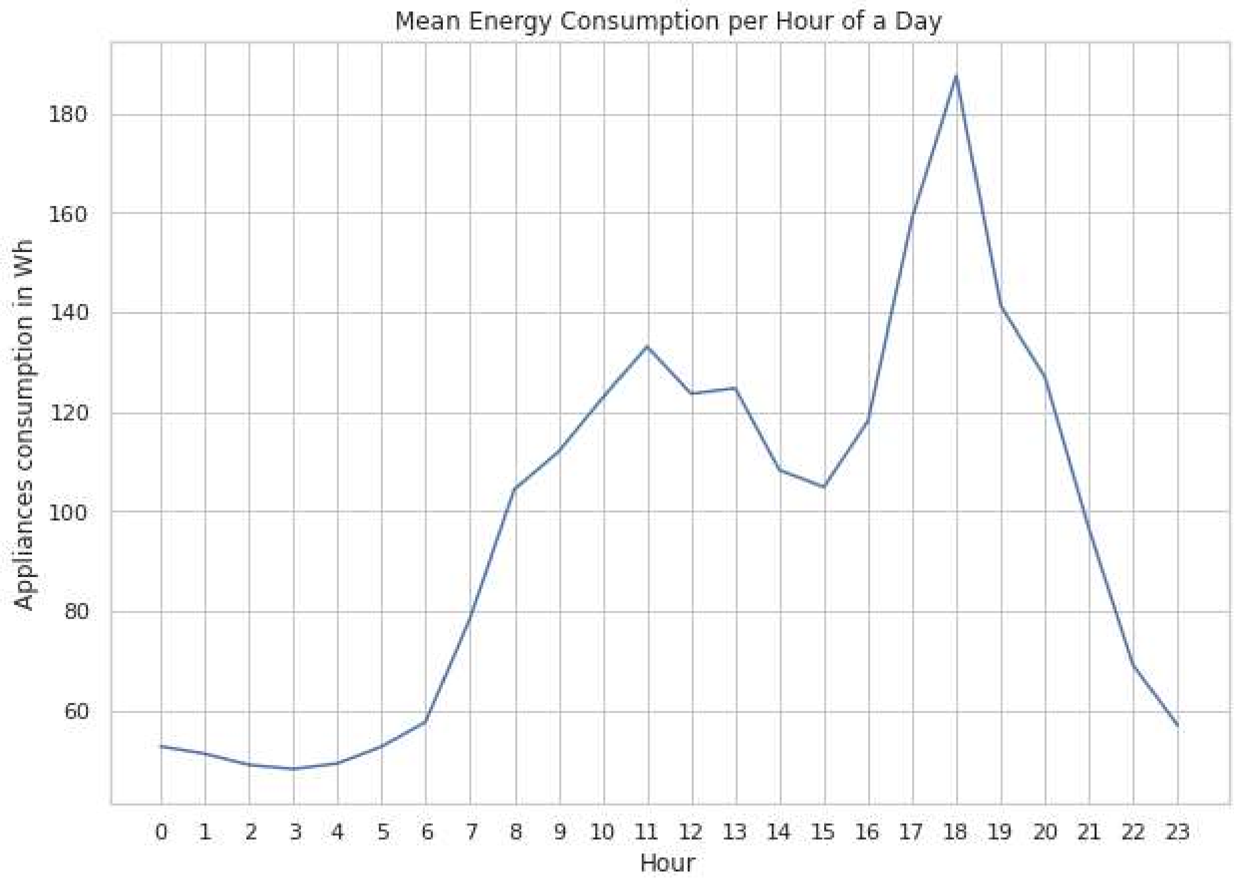
5 rows × 47 columns

Setting the assumptions as to lower or higher ,setting the relationship between consumption and load is very much significance to proceed so ofcourse when the consumption is high the load is higher to ,we are going to do lot of tryouts we are going to indentify which one is going to be our boundary or bases so that we can be able to identify whether or not a certain value at the certain point of date can be considered higher or lower so this would depend on appliances consumption

```
df_hour['low_consum']=(df_hour.appliances+25<(df_hour.hour_avg))*1
df_hour['high_consum']=(df_hour.appliances+25>(df_hour.hour_avg))*1

df_30min['low_consum']=(df_30min.appliances+25<(df_30min.hour_avg))*1
df_30min['high_consum']=(df_30min.appliances+35>(df_30min.hour_avg))*1


def daily(x,df=df):
  return df.groupby('weekday')[x].mean()
def hourly(x,df=df):
  return df.groupby('hour')[x].mean()

def monthly_daily(x,df=df):
  by_day = df.pivot_table(index='weekday',
                          columns=['month'],
                          values=x,
                          aggfunc='mean')
  return round(by_day,ndigits=2)
```

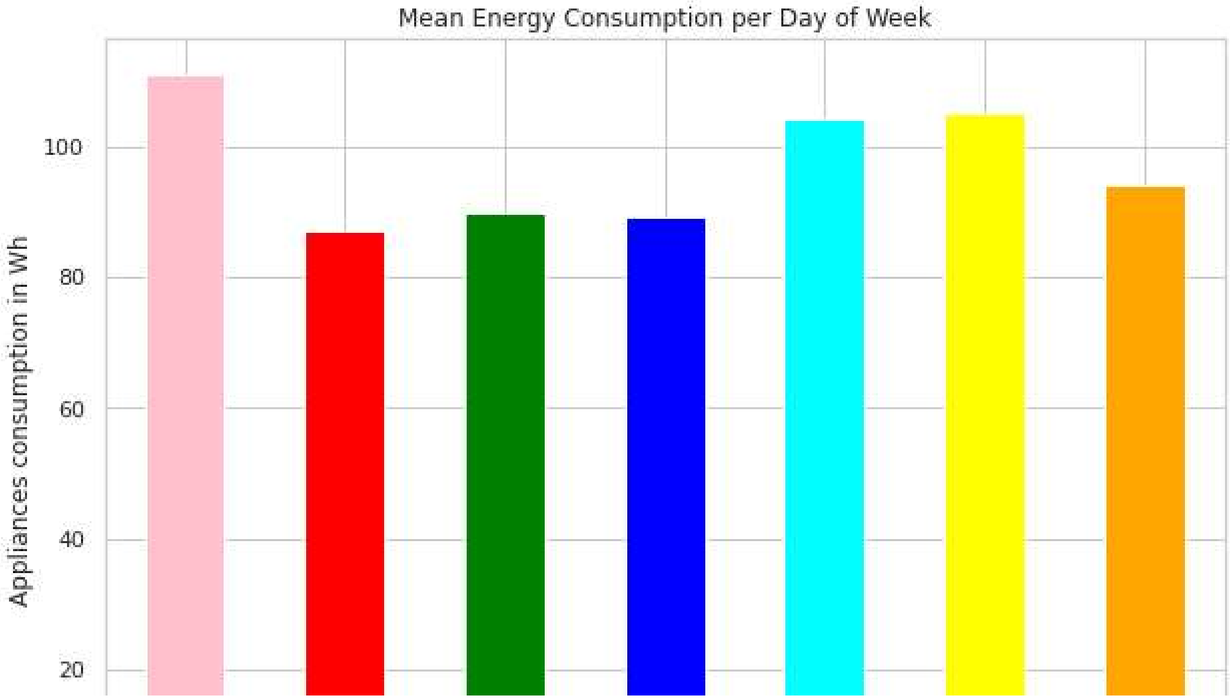
plotting the hourly consumption

```
import matplotlib.pyplot as plt
hourly('appliances').plot(figsize=(10,7))
plt.xlabel('Hour')
plt.ylabel('Appliances consumption in Wh')
ticks=list(range(0,24,1))
plt.title('Mean Energy Consumption per Hour of a Day')
plt.xticks(ticks);
```



Weekly Consumption

```
#Weekly Consumption
daily('appliances').plot(kind='bar', color=['pink', 'red','green','blue','cyan','yellow','orange'],f
ticks=list(range(0,7,1))
labels="Mon Tues Weds Thurs Fri Sat Sun".split()
plt.xlabel('Day')
plt.ylabel('Appliances consumption in Wh')
plt.title('Mean Energy Consumption per Day of Week')
plt.xticks(ticks,labels);
```



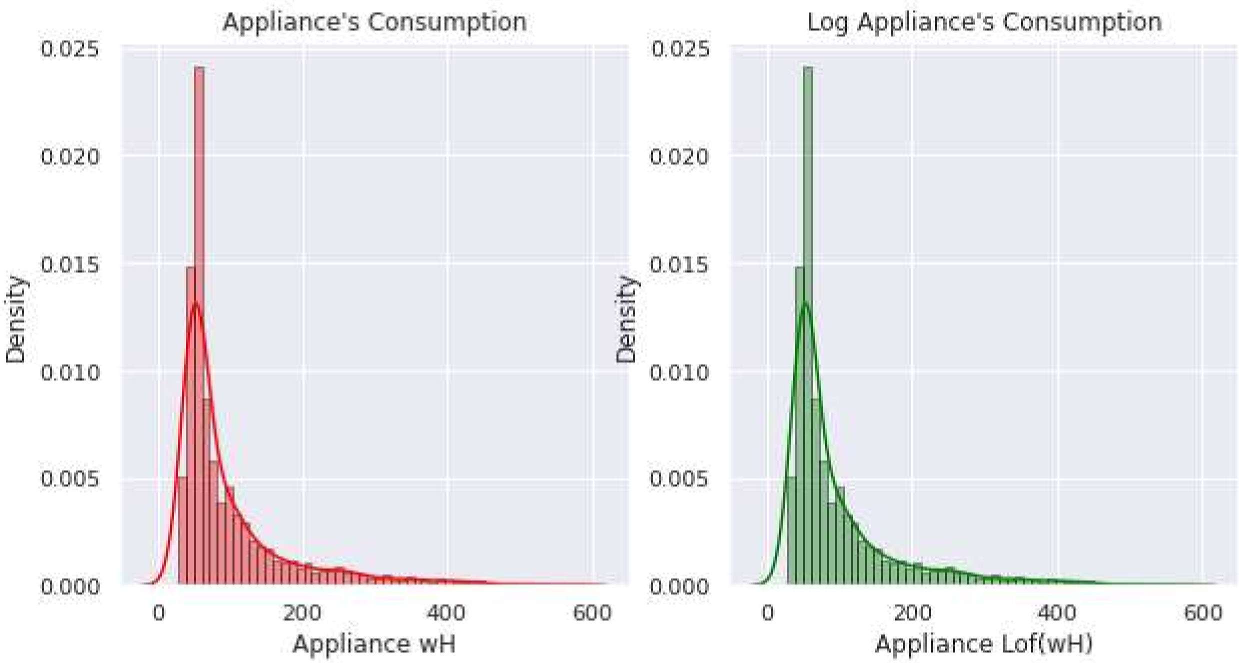
Monthly Consumption


```
#Monthly Consumption
sns.set(rc={'figure.figsize':(10,8)},)
ax=sns.heatmap(monthly_daily('appliances').T,cmap="PiYG",
               xticklabels="Mon Tues Weds Thurs Fri Sat Sun".split(),
               yticklabels="January February March April May".split(),
               annot=True, fmt='g',
               cbar_kws={'label': 'Consumption in wH'}).set_title("<ean appliances Consumption(Wh) p
plt.show()
```

Histogram for raw data and already log transform data

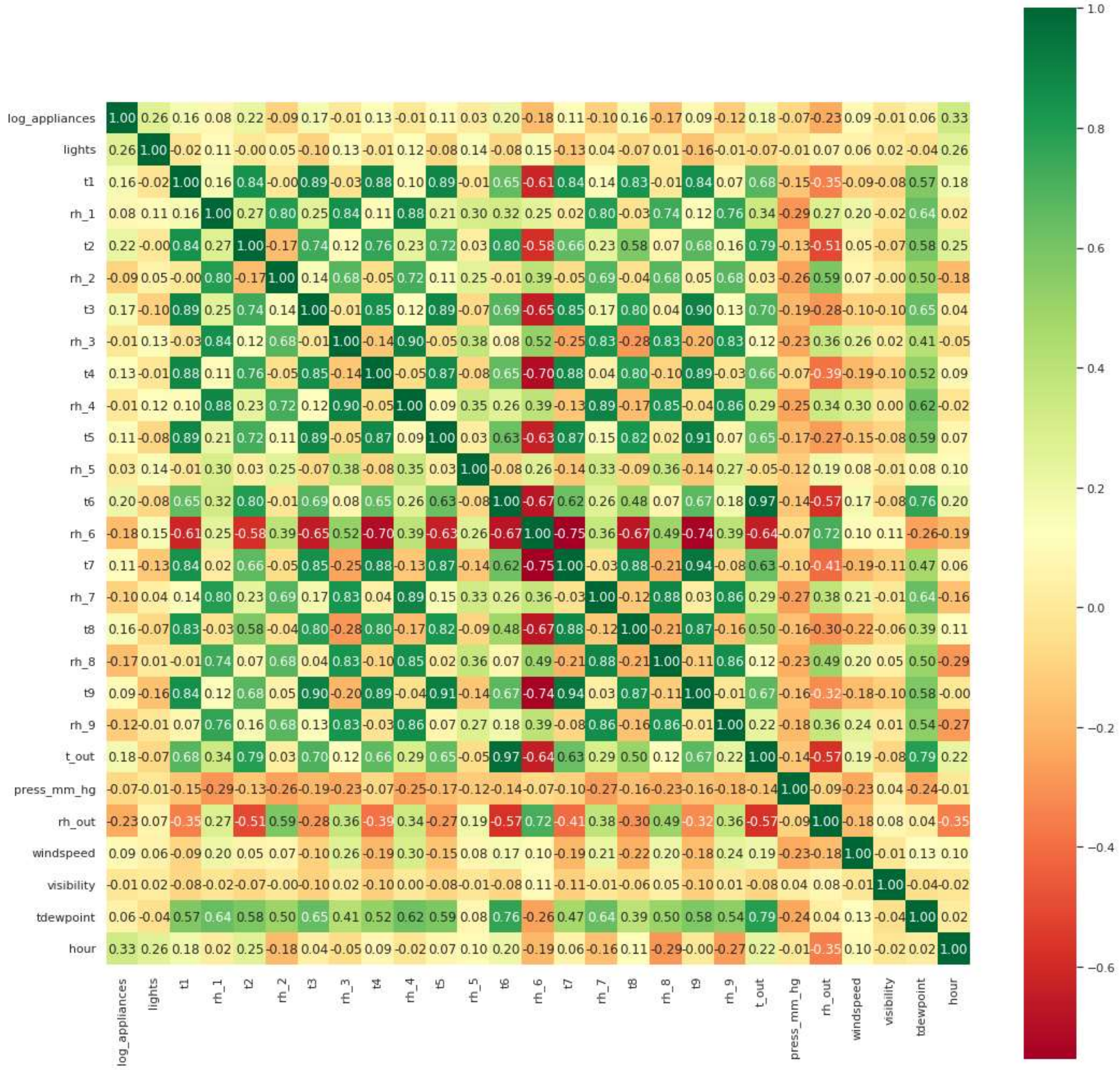
```
f, axes= plt.subplots(1,2,figsize=(10,5))

sns.distplot(df_hour.appliances, hist=True, color='red', hist_kws={'edgecolor':'black'},ax=axes[0])
axes[0].set_title("Appliance's Consumption")
axes[0].set_xlabel('Appliance wH')

sns.distplot(df_hour.appliances, hist=True, color='green', hist_kws={'edgecolor':'black'},ax=axes[1]
axes[1].set_title("Log Appliance's Consumption")
axes[1].set_xlabel('Appliance Lof(wH)')
```

```
Text(0.5, 0, 'Appliance Lof(wH)')
```

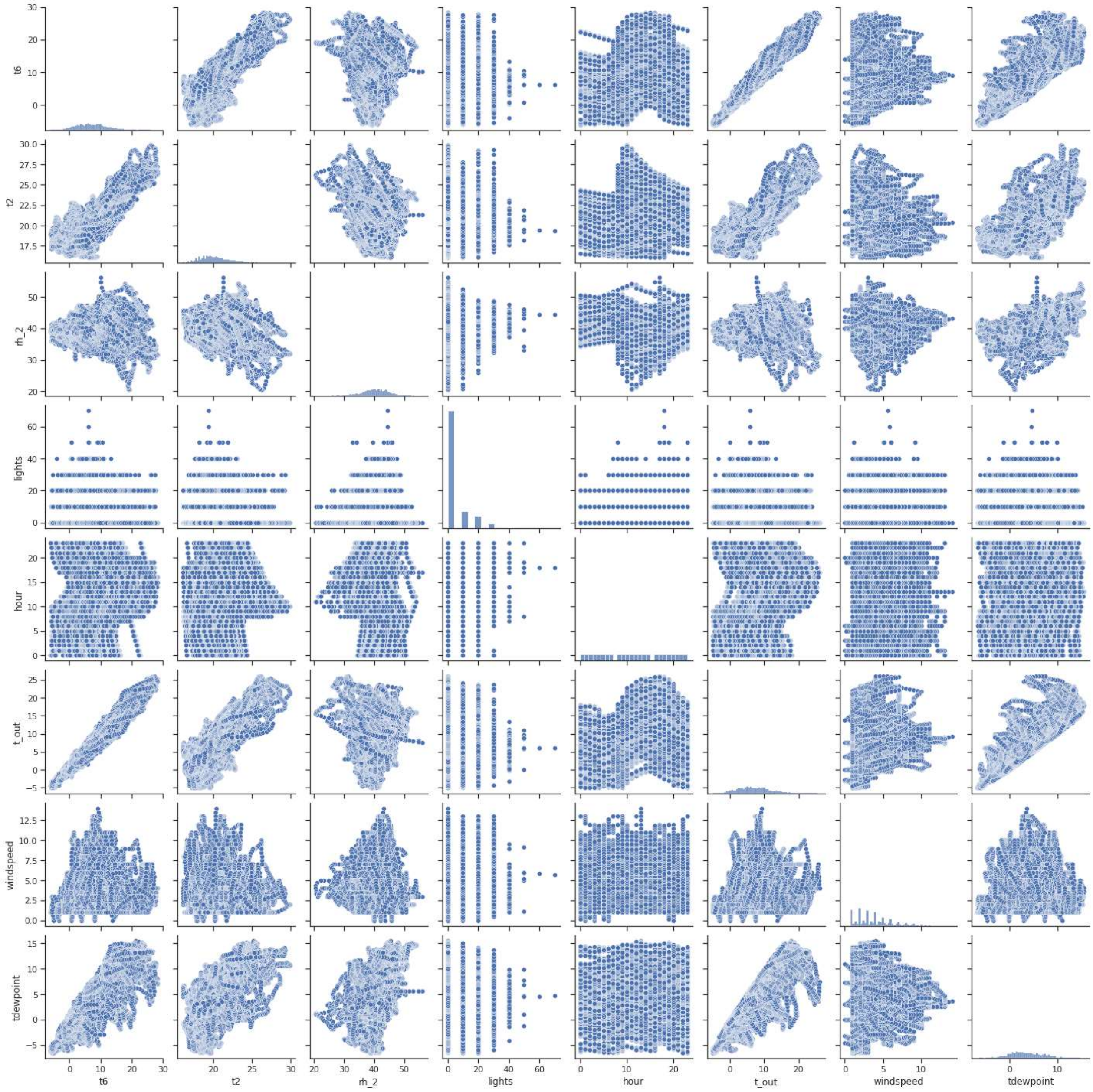


here we have pearsons correlation to identify linearlity among different pairs of features.Here we are going to find which one has stronger correlation with log appliances.

```
col = ['log_appliances','lights','t1','rh_1','t2','rh_2','t3','rh_3','t4',
       'rh_4','t5','rh_5','t6','rh_6','t7','rh_7','t8','rh_8','t9',
       'rh_9','t_out','press_mm_hg','rh_out','windspeed','visibility',
       'tdewpoint','hour']
corr=df[col].corr()
plt.figure(figsize=(18,18))
sns.set(font_scale=1)
sns.heatmap(corr, cbar=True, annot=True, square=True, cmap='RdYlGn', fmt='.2f',xticklabels=col,ytick
plt.show()
```




```
col=['t6','t2','rh_2','lights','hour','t_out','windspeed','tdewpoint']
sns.set(style="ticks",color_codes=True)
sns.pairplot(df[col])
plt.show()
```



t6
t2
rh_2
lights
hour
t_out
windspeed
tdewpoint

Training the model

```
import pandas as pd
for cat_feature in ['weekday','hour']:
    df_hour=pd.concat([df_hour, pd.get_dummies(df_hour[cat_feature])], axis=1)
    df_30min=pd.concat([df_30min, pd.get_dummies(df_30min[cat_feature])], axis=1)
    df=pd.concat([df, pd.get_dummies(df_hour[cat_feature])], axis=1)


lin_model=['low_consum','high_consum','hour','t6','rh_6','lights','hour*lights','windspeed','t6rh6']


df_hour.lights= df_hour.lights.astype(float)
df_hour.log_appliances= df_hour.log_appliances.astype(float)
df_hour.hour= df_hour.hour.astype(float)
df_hour.low_consum= df_hour.low_consum.astype(float)
df_hour.high_consum= df_hour.high_consum.astype(float)
df_hour.t6rh6= df_hour.t6rh6.astype(float)


test_size=.2
test_index= int(len(df_hour.dropna())*(1-test_size))

X1_train, X1_test = df_hour[lin_model].iloc[:test_index,], df_hour[lin_model].iloc[test_index:,]
y1_train = df_hour.log_appliances.iloc[:test_index,]
y_test = df_hour.log_appliances.iloc[test_index:,]


from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X1_train)
X1_train = scaler.transform(X1_train)
X1_test = scaler.transform(X1_test)


from sklearn import linear_model
lin_model = linear_model.LinearRegression()
lin_model.fit(X1_train, y1_train)
```

```
LinearRegression()
```

Model Evaluation and Selection

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
```

```
def evaluate(model, test_features, test_labels):
    prediction = model.predict(test_features)
    errors=abs(prediction - test_labels)
    mape= 100 * np.mean(errors/ test_labels)
    r_score = 100 * r2_score(test_labels, prediction)
    accuracy= 100 - mape
    print(model,'\n')
    print('Average Error: {:0.4f} degrees'.format(np.mean(errors)))
    print('Variance score R^2: {:0.2f}%'.format(r_score))
    print('Accuracy: {:0.2f}%\n'.format(accuracy))
```

```
evaluate(lin_model, X1_test, y_test)
```

```
LinearRegression()

Average Error: 0.3212 degrees
Variance score R^2: 25.35%
Accuracy: 92.62%
```

Validating our model

```
cv = TimeSeriesSplit(n_splits = 10)
print('Linear Model:')
scores =  cross_val_score(lin_model,X1_train, y1_train, cv=cv, scoring='neg_mean_absolute_error')
print("Accuracy : 0%.2f (+/- %0.2f) degrees" % (100+scores.mean(),scores.std()*2))
scores = cross_val_score(lin_model, X1_train , y1_train, cv = cv ,scoring='r2')
print("R^2: %0.2f (+/- %0.2f) degrees " %(scores.mean(),scores.std()*2))
```

```
Linear Model:
Accuracy : 099.64 (+/- 0.07) degrees
R^2: 0.27 (+/- 0.19) degrees
```

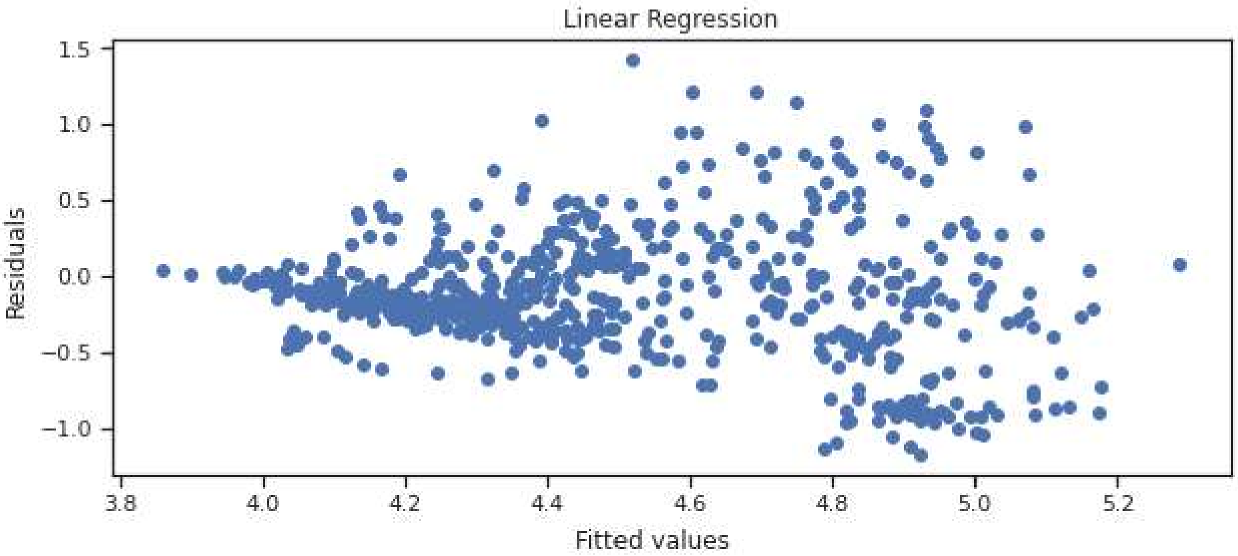
Results of predicted value

```
y1_pred = lin_model.predict(X1_test)
```

```
fig, ax = plt.subplots(figsize=(10,4),sharey=True)
ax.scatter(y1_pred,y_test-y1_pred)
ax.set_title('Linear Regression')
```

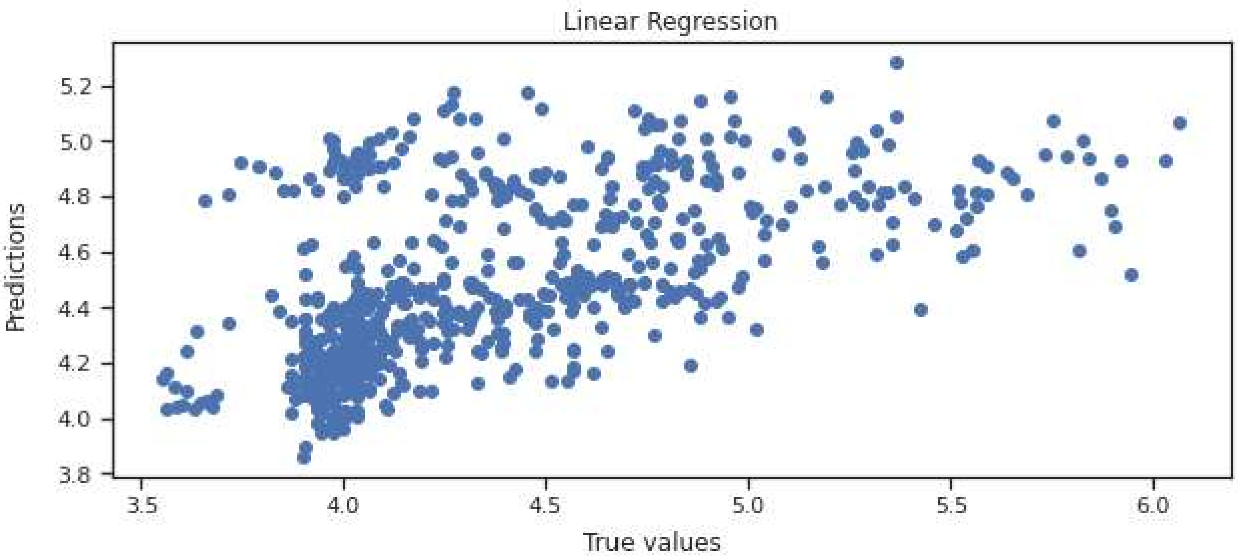
```
fig.text(0.06, 0.5, 'Residuals', ha ='center' , va='center', rotation='vertical')
fig.text(0.5,0.01,'Fitted values',ha ='center' , va='center')
```

```
Text(0.5, 0.01, 'Fitted values')
```




```
fig, ax = plt.subplots(figsize=(10,4),sharey=True)
ax.scatter(y_test,y1_pred)
ax.set_title('Linear Regression')
fig.text(0.06, 0.5, 'Predictions', ha ='center' , va='center', rotation='vertical')
fig.text(0.5,0.01,'True values',ha ='center' , va='center')
```

```
Text(0.5, 0.01, 'True values')
```



Comparing predicted value and actual value to check whether our model is overfitting or underfitting.

```
fig = plt.figure(figsize=(20,8))
plt.plot(y_test.values, label='Target value', color='b')
plt.plot(y1_pred, label='Linear Prediction', linestyle='--', color='y')
plt.legend(loc=1)
```

```
<matplotlib.legend.Legend at 0x7f87c4055710>
```