Individual household electric power consumption Data Set

```
1.Linear Regression,
2.Ridge Regression,
3.Lasso Regression,
4.Elastic Net Regression,
5.Support Vector Regression
```

Dharavath Ramdas

```
Dataset link:
https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
Code link: https://github.com/dharavathramdas101
```

Problem Statement:

Steps:

```
1. Data ingestion
2. EDA
3. Preprocessing
Pickling for the preprocessing object(save the preprocessing model)
After preprocessing you have to store data inside MONGODB You have to load the dat
a from mongo db
4. Model

Regression:linear regression,ridge regression,lasso regression,elastic net, suppor
t vector regression
```

In [ ]:

```

```

Importing Libraries

In [1]:

```
### Pandas and Numpy
import pandas as pd
import numpy as np

### Visualisation libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

### To ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Loding Dataset

In [2]:

```
#df = pd.read_csv(r"C:\Users\DHARAVATH RAMDAS\Downloads\household_power_consumption\househo
```

In [3]:

```
df = pd.read_csv(r"C:\Users\DHARAVATH RAMDAS\Downloads\household_power_consumption\househol
                  parse_dates={'dt' : ['Date', 'Time']}, infer_datetime_format=True,
                 low_memory=False, na_values=['nan','?'],index_col='dt')
```

In [4]:

```
df
```

Out[4]:

dt	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
2006-12-16 17:24:00	4.216	0.418	234.84	18.4	0.0
2006-12-16 17:25:00	5.360	0.436	233.63	23.0	0.0
2006-12-16 17:26:00	5.374	0.498	233.29	23.0	0.0
2006-12-16 17:27:00	5.388	0.502	233.74	23.0	0.0
2006-12-16 17:28:00	3.666	0.528	235.68	15.8	0.0
...	...	...	...	...	...
2010-11-26 20:58:00	0.946	0.000	240.43	4.0	0.0
2010-11-26 20:59:00	0.944	0.000	240.00	4.0	0.0
2010-11-26 21:00:00	0.938	0.000	239.82	3.8	0.0
2010-11-26 21:01:00	0.934	0.000	239.70	3.8	0.0
2010-11-26 21:02:00	0.932	0.000	239.55	3.8	0.0

2075259 rows × 7 columns

Observation:

```
1.data include 'nan' and '?' as a string. i converted both to numpy nan in importi
ng stage(above) and treated both of them the same
2.i merged two columns 'Date' and 'Time' to 'dt'
```

Random Sample data taking 50,000

In [5]:

```
dfs = df.sample(n=50000,replace=False)
dfs.head()
```

Out[5]:

dt	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
2009-06-16 04:34:00	0.234	0.112	241.93	1.2	0.0
2007-08-01 21:36:00	0.212	0.112	235.87	1.0	0.0
2010-10-10 03:04:00	0.490	0.274	240.10	2.2	0.0
2009-06-18 21:24:00	0.542	0.058	241.25	2.2	0.0
2010-01-18 06:48:00	2.032	0.132	242.42	8.4	0.0

In [6]:

```
#dfs=dfs.reset_index()
#dfs.drop(['index'],axis=1,inplace=True)
#dfs.head()
```

In [7]:

```
#dfs.drop('level_0',axis=1,inplace=True)
```

In [8]:

```
## Checking the shape of dataset (no.of rows and no.of columns)
```

In [9]:

```
dfs.shape
```

Out[9]:

```
(50000, 7)
```

Checking top five records of dataframe

In [11]:

```
dfs.head()
```

Out[11]:

dt	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
2009-06-16 04:34:00	0.234	0.112	241.93	1.2	0.0
2007-08-01 21:36:00	0.212	0.112	235.87	1.0	0.0
2010-10-10 03:04:00	0.490	0.274	240.10	2.2	0.0
2009-06-18 21:24:00	0.542	0.058	241.25	2.2	0.0
2010-01-18 06:48:00	2.032	0.132	242.42	8.4	0.0

Checking last five records of dataframe

In [13]:

```
dfs.tail()
```

Out[13]:

dt	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
2008-06-20 14:07:00	0.328	0.254	242.66	1.6	0.0
2009-09-29 20:29:00	4.042	0.068	236.18	17.0	37.0
2010-04-26 18:02:00	0.340	0.114	241.75	1.4	0.0
2007-04-10 19:34:00	0.500	0.108	240.27	2.4	0.0
2007-11-30 23:09:00	0.406	0.220	247.63	1.8	0.0

Checking the data types using info

In [15]:

```
dfs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 50000 entries, 2009-06-16 04:34:00 to 2007-11-30 23:09:00
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Global_active_power    49378 non-null  float64
 1   Global_reactive_power  49378 non-null  float64
 2   Voltage                49378 non-null  float64
 3   Global_intensity       49378 non-null  float64
 4   Sub_metering_1         49378 non-null  float64
 5   Sub_metering_2         49378 non-null  float64
 6   Sub_metering_3         49378 non-null  float64
dtypes: float64(7)
memory usage: 3.1 MB
```

In [ ]:

```

```

Check the columns

In [17]:

```
dfs.columns
```

Out[17]:

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3'],
      dtype='object')
```

In [18]:

```
dfs.isnull().sum()
```

Out[18]:

```
Global_active_power      622
Global_reactive_power    622
Voltage                  622
Global_intensity         622
Sub_metering_1           622
Sub_metering_2           622
Sub_metering_3           622
dtype: int64
```

fill missing values row wise and making the changes permanent in the original dataframe

In [20]:

```
dfs.ffill(axis=0,inplace=True)
```

Cross check wether all missing values are filled

In [22]:

```
dfs.isnull().sum()
```

Out[22]:

```
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
```

EDA (Analysis)

1. Weekly
2. Monthly
3. Quarterly
4. Yearly

Sub Datasets

1. Power Consuption
2. Sub metering
3. Global Recative, Global Active and Global Intensity

Creating Target Variable

In [24]:

```
p1 = (dfs['Global_active_power']*1000/60)
p2 = dfs['Sub_metering_1'] + dfs['Sub_metering_2'] + dfs['Sub_metering_3']
dfs['power_consumption'] = p1-p2
dfs.head()
```

Out[24]:

dt	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
2009-06-16 04:34:00	0.234	0.112	241.93	1.2	0.0
2007-08-01 21:36:00	0.212	0.112	235.87	1.0	0.0
2010-10-10 03:04:00	0.490	0.274	240.10	2.2	0.0
2009-06-18 21:24:00	0.542	0.058	241.25	2.2	0.0
2010-01-18 06:48:00	2.032	0.132	242.42	8.4	0.0

Creating two more columns for index,Date and Time columns seperately

```
with the help of this new column 'Date', it will be easier to do grouping on the d
ata wich will ease the work of visualization for better understanding on Data
```

In [25]:

```
dfs['Date'] = dfs.index.date
dfs['time'] = dfs.index.time
```

In [26]:

```
dfs['Date'] = pd.to_datetime(dfs['Date'])
```

In [27]:

```
dfs.head(2)
```

Out[27]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
dt					
2009-06-16 04:34:00	0.234	0.112	241.93	1.2	0.0
2007-08-01 21:36:00	0.212	0.112	235.87	1.0	0.0

In [28]:

```
dfs = dfs.set_index('Date')
```

In [32]:

```
dfs.head(2)
```

Out[32]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
Date					
2009-06-16	0.234	0.112	241.93	1.2	0.0
2007-08-01	0.212	0.112	235.87	1.0	0.0

In [33]:

```
#dfs = dfs.astype({'Global_active_power':'float', 'Global_reactive_power':'float',
      # 'Voltage':'float', 'Global_intensity':'float', 'Sub_metering_1':'float', 'Sub_meter
       #'Sub_metering_3':'float'})
```

In [34]:

```
dfs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 50000 entries, 2009-06-16 to 2007-11-30
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Global_active_power    50000 non-null  float64
 1   Global_reactive_power  50000 non-null  float64
 2   Voltage                50000 non-null  float64
 3   Global_intensity       50000 non-null  float64
 4   Sub_metering_1         50000 non-null  float64
 5   Sub_metering_2         50000 non-null  float64
 6   Sub_metering_3         50000 non-null  float64
 7   power_consumption      50000 non-null  float64
 8   time                   50000 non-null  object 
dtypes: float64(8), object(1)
memory usage: 3.8+ MB
```

In [44]:

```
dfs.isnull().sum()
```

Out[44]:

```
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
power_consumption        0
time                     0
dtype: int64
```

In [45]:

```
dfs.head(3)
```

Out[45]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	S
Date						
2009-06-16	0.234	0.112	241.93	1.2	0.0	
2007-08-01	0.212	0.112	235.87	1.0	0.0	
2010-10-10	0.490	0.274	240.10	2.2	0.0	

In [47]:

```
### Columns
```

In [48]:

```
dfs.columns
```

Out[48]:

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3', 'power_consumption', 'time'],
      dtype='object')
```

Describe used to see the stats analysis

In [50]:

```
dfs.describe()
```

Out[50]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_meterin
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000
mean	1.089072	0.123812	240.827886	4.616560	1.104
std	1.053175	0.111709	3.245199	4.426557	6.076
min	0.078000	0.000000	224.440000	0.200000	0.000
25%	0.310000	0.048000	238.970000	1.400000	0.000
50%	0.604000	0.102000	240.990000	2.600000	0.000
75%	1.526000	0.194000	242.870000	6.400000	0.000
max	10.348000	0.942000	253.070000	44.600000	81.000

In [51]:

```
dfs.describe().T
```

Out[51]:

	count	mean	std	min	25%	50%	7
Global_active_power	50000.0	1.089072	1.053175	0.078000	0.310000	0.604	1.526
Global_reactive_power	50000.0	0.123812	0.111709	0.000000	0.048000	0.102	0.194
Voltage	50000.0	240.827886	3.245199	224.440000	238.970000	240.990	242.870
Global_intensity	50000.0	4.616560	4.426557	0.200000	1.400000	2.600	6.400
Sub_metering_1	50000.0	1.104080	6.076037	0.000000	0.000000	0.000	0.000
Sub_metering_2	50000.0	1.287000	5.774539	0.000000	0.000000	0.000	1.000
Sub_metering_3	50000.0	6.416420	8.419567	0.000000	0.000000	1.000	17.000
power_consumption	50000.0	9.343692	9.620168	-0.766667	3.833333	5.500	10.466

In [52]:

```
dfs_df = dfs.groupby(['Date']).sum()
```

In [53]:

```
dfs_df.isnull().sum()
```

Out[53]:

```
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
power_consumption        0
dtype: int64
```

In [54]:

```
dfs_df.head()
```

Out[54]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
Date					
2006-12-16	43.670	0.822	3051.65	185.0	0.0
2006-12-17	80.926	5.650	8155.79	346.2	41.0
2006-12-18	69.762	4.608	11355.14	293.4	41.0
2006-12-19	32.018	4.264	9682.64	137.2	0.0
2006-12-20	55.802	2.766	6512.31	233.8	0.0

In [55]:

```
dfs_df.shape
```

Out[55]:

```
(1442, 8)
```

In [56]:

```
df_power_consumption = dfs['power_consumption'].head()
```

In [57]:

```
df_sub_meterings = dfs[['Sub_metering_1','Sub_metering_2','Sub_metering_3']]
```

In [58]:

```
df_sub_meterings.head()
```

Out[58]:

Date	Sub_metering_1	Sub_metering_2	Sub_metering_3
2009-06-16	0.0	0.0	0.0
2007-08-01	0.0	0.0	0.0
2010-10-10	0.0	1.0	1.0
2009-06-18	0.0	0.0	1.0
2010-01-18	0.0	1.0	19.0

In [59]:

```
df_active_reactive = dfs[['Global_active_power','Global_reactive_power','Global_intensity']
```

In [60]:

```
df_active_reactive.head()
```

Out[60]:

Date	Global_active_power	Global_reactive_power	Global_intensity
2009-06-16	0.234	0.112	1.2
2007-08-01	0.212	0.112	1.0
2010-10-10	0.490	0.274	2.2
2009-06-18	0.542	0.058	2.2
2010-01-18	2.032	0.132	8.4

In [61]:

```
df_power_consumption.groupby(df_power_consumption.index.year).describe()
```

Out[61]:

Date	count	mean	std	min	25%	50%	75%	max
2007	1.0	3.533333	NaN	3.533333	3.533333	3.533333	3.533333	3.533333
2009	2.0	5.966667	2.922708	3.900000	4.933333	5.966667	7.000000	8.033333
2010	2.0	10.016667	5.444722	6.166667	8.091667	10.016667	11.941667	13.866667

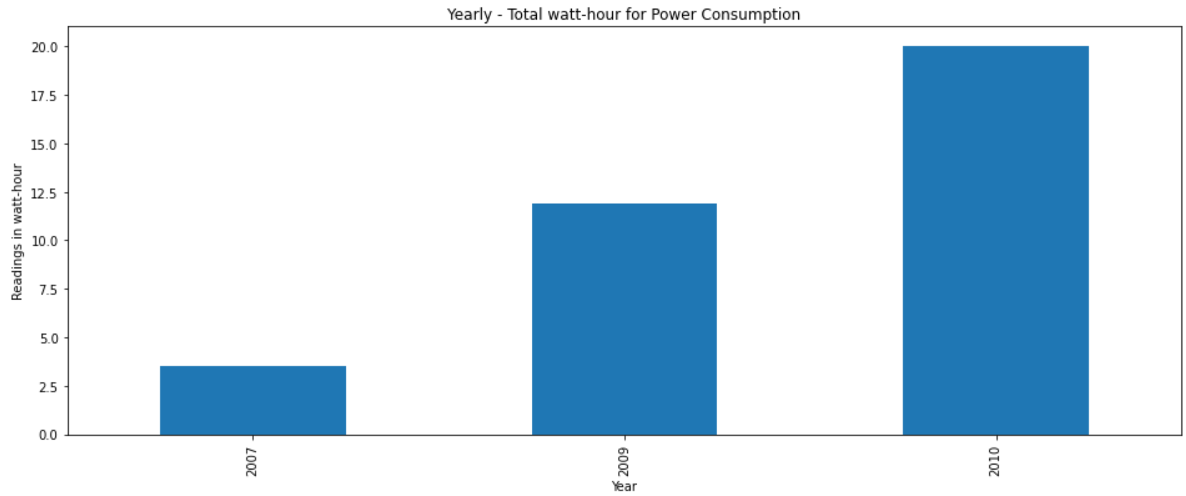
Yearly total watt hour for power consumption

In [62]:

```
.plot(kind="bar",xlabel='Year',ylabel='Readings in watt-hour',title="Yearly - Total watt-hou
```

Out[62]:

```
<AxesSubplot:title={'center':'Yearly - Total watt-hour for Power Consumptio
n'}, xlabel='Year', ylabel='Readings in watt-hour'>
```



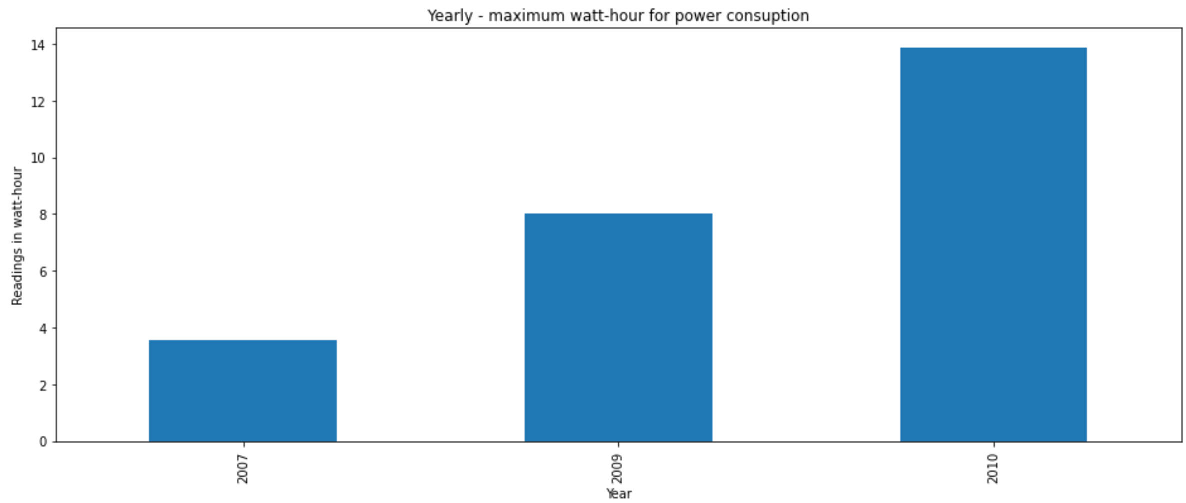
Yearly - Maximum watt-hour for power consumption

In [64]:

```
l='Year',ylabel='Readings in watt-hour',title="Yearly - maximum watt-hour for power consupti
```

Out[64]:

```
<AxesSubplot:title={'center':'Yearly - maximum watt-hour for power consuptio
n'}, xlabel='Year', ylabel='Readings in watt-hour'>
```



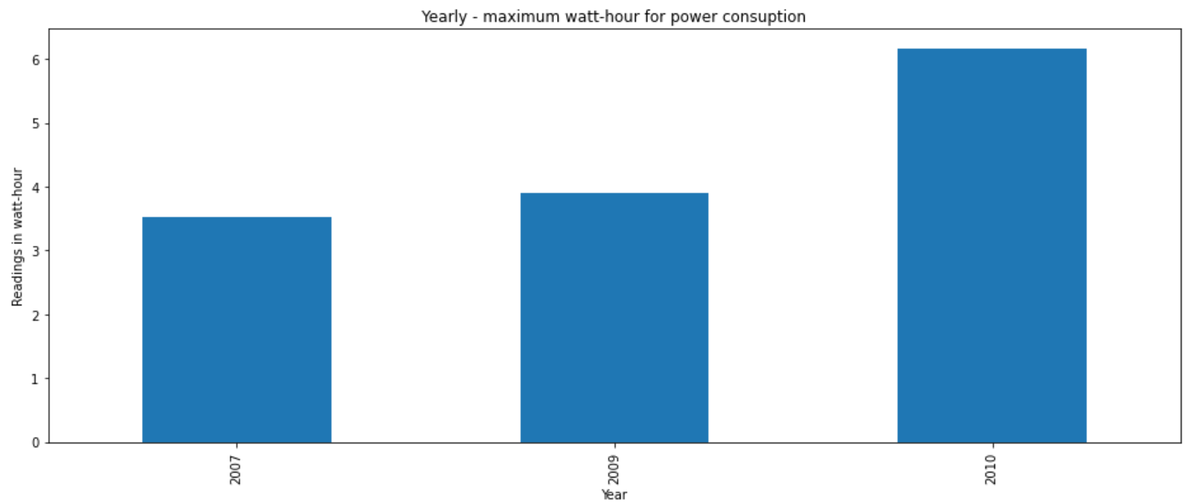
Yearly = minimum watt-hour for power consuption

In [66]:

```
df_power_consumption.groupby(df_power_consumption.index.year).min().plot(kind="bar",xlabel=
```

Out[66]:

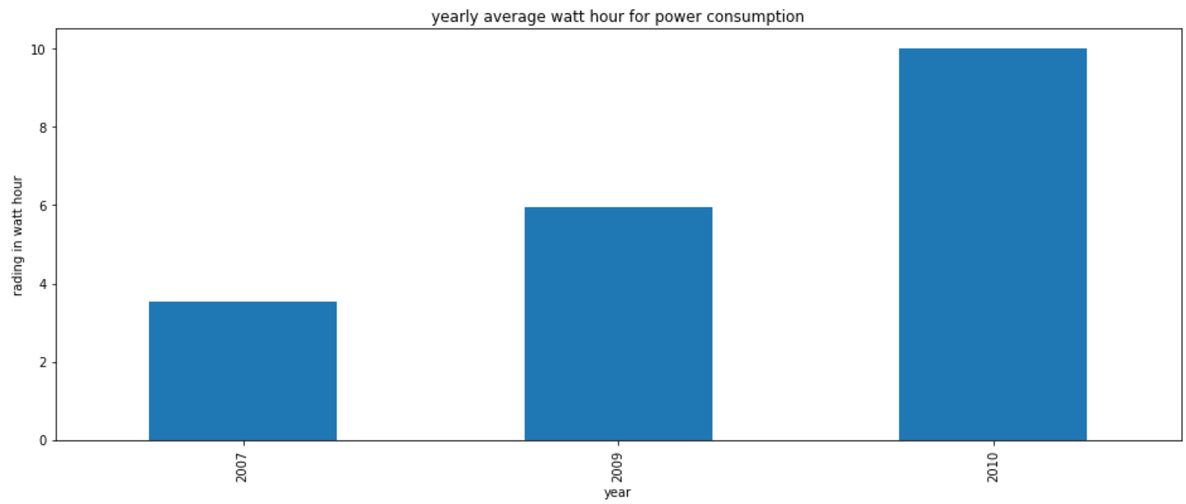
```
<AxesSubplot:title={'center':'Yearly - maximum watt-hour for power consuptio
n'}, xlabel='Year', ylabel='Readings in watt-hour'>
```



Yearly average watt-hour for power consuption

In [68]:

```
bel='year',ylabel="rading in watt hour",title="yearly average watt hour for power consumptio
```



Checking Statistical summary of power consuption monthly

In [69]:

```
df_power_consumption.groupby(df_power_consumption.index.month).describe()
```

Out[69]:

Date	count	mean	std	min	25%	50%	75%	max
1	1.0	13.866667	NaN	13.866667	13.866667	13.866667	13.866667	13.866667
6	2.0	5.966667	2.922708	3.900000	4.933333	5.966667	7.000000	8.033333
8	1.0	3.533333	NaN	3.533333	3.533333	3.533333	3.533333	3.533333
10	1.0	6.166667	NaN	6.166667	6.166667	6.166667	6.166667	6.166667

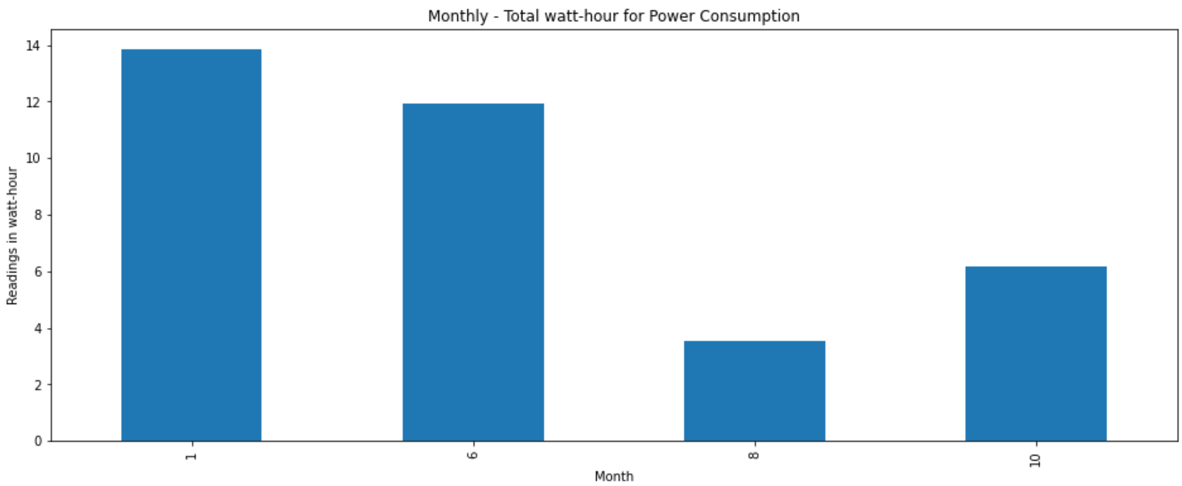
Monthly - Total watt hour for power consumption

In [70]:

```
ndex.month).sum().plot(kind="bar",xlabel='Month',ylabel='Readings in watt-hour',title="Month
```

Out[70]:

```
<AxesSubplot:title={'center':'Monthly - Total watt-hour for Power Consumptio
n'}, xlabel='Month', ylabel='Readings in watt-hour'>
```



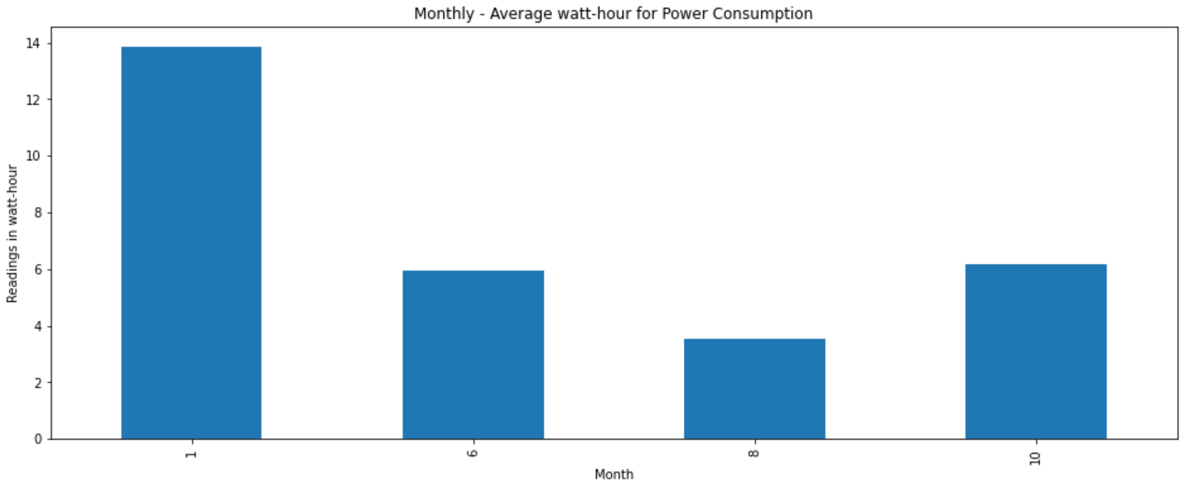
Monthly - Average watt-hour for Power Consumption

In [71]:

```
th',ylabel='Readings in watt-hour',title="Monthly - Average watt-hour for Power Consumption
```

Out[71]:

```
<AxesSubplot:title={'center':'Monthly - Average watt-hour for Power Consumpt
ion'}, xlabel='Month', ylabel='Readings in watt-hour'>
```



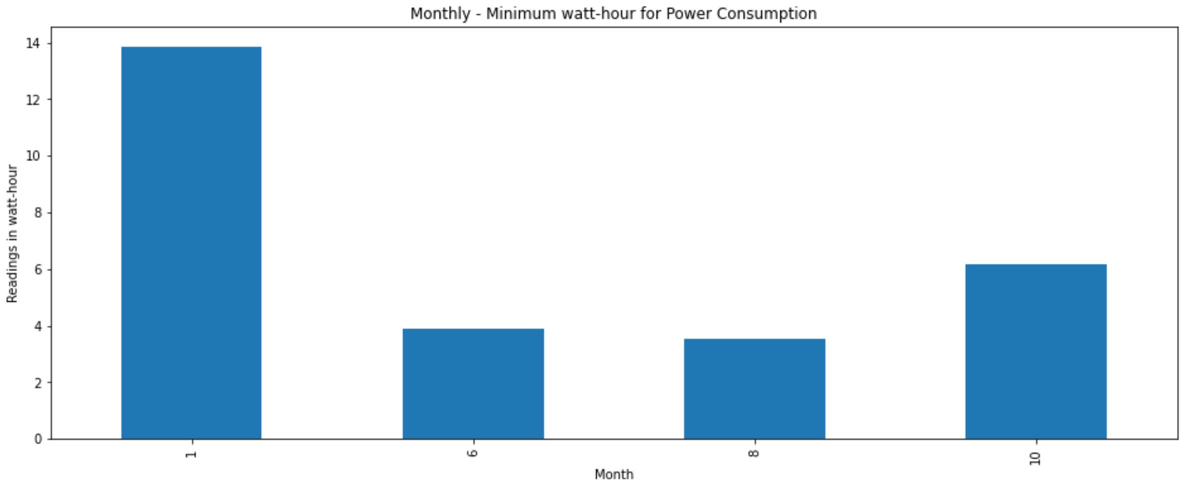
Monthly - Minimum watt-hour for Power Consumption

In [72]:

```
th',ylabel='Readings in watt-hour',title="Monthly - Minimum watt-hour for Power Consumption
```

Out[72]:

```
<AxesSubplot:title={'center':'Monthly - Minimum watt-hour for Power Consumpt
ion'}, xlabel='Month', ylabel='Readings in watt-hour'>
```



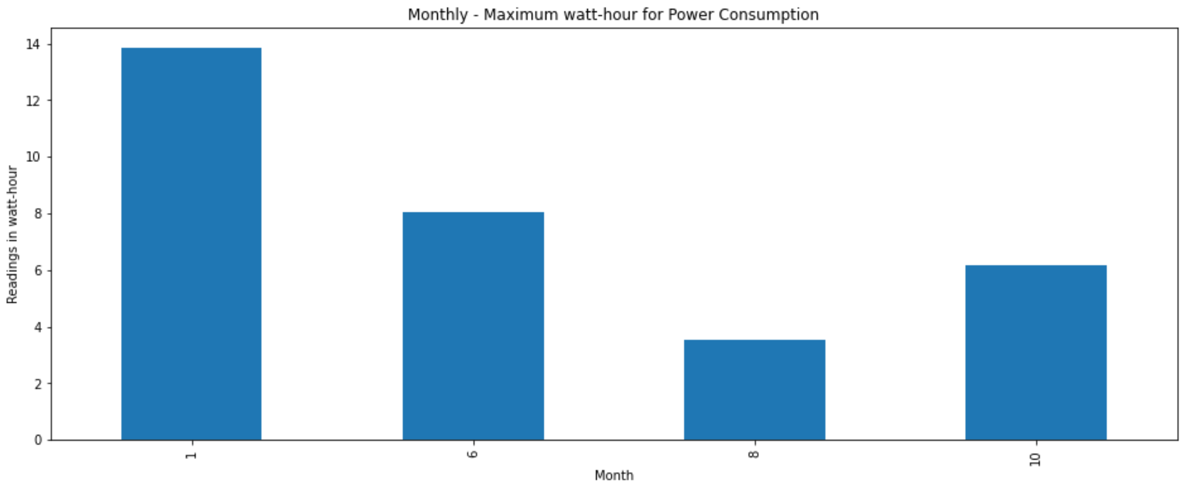
Monthly - Maximum watt-hour for Power Consumption

In [73]:

```
th',ylabel='Readings in watt-hour',title="Monthly - Maximum watt-hour for Power Consumption
```

Out[73]:

```
<AxesSubplot:title={'center':'Monthly - Maximum watt-hour for Power Consumpt
ion'}, xlabel='Month', ylabel='Readings in watt-hour'>
```



Checking Statistical summary of power consumption quarterly

In [74]:

```
df_power_consumption.groupby(df_power_consumption.index.quarter).describe()
```

Out[74]:

	count	mean	std	min	25%	50%	75%	max
Date								
1	1.0	13.866667	NaN	13.866667	13.866667	13.866667	13.866667	13.866667
2	2.0	5.966667	2.922708	3.900000	4.933333	5.966667	7.000000	8.033333
3	1.0	3.533333	NaN	3.533333	3.533333	3.533333	3.533333	3.533333
4	1.0	6.166667	NaN	6.166667	6.166667	6.166667	6.166667	6.166667

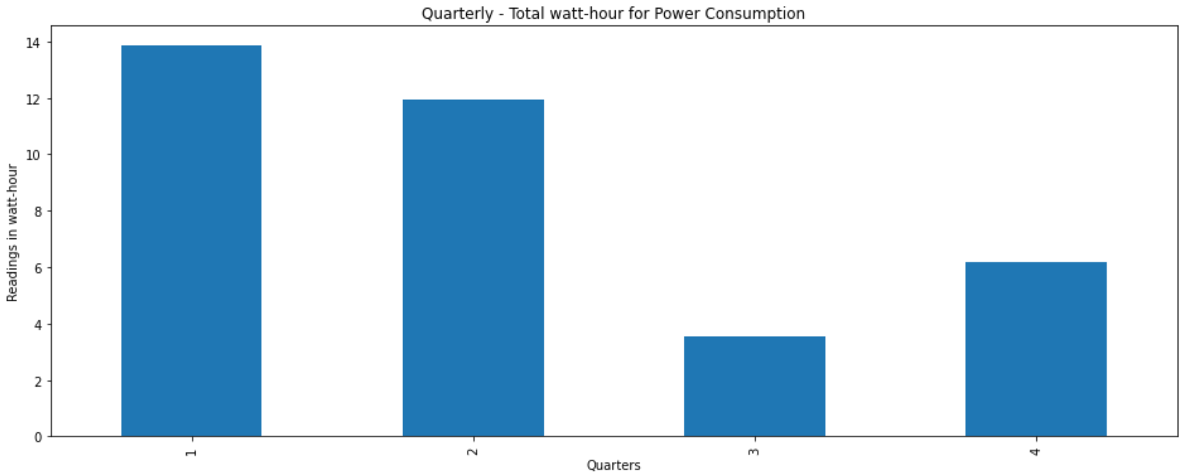
Quarterly - Total watt-hour for Power Consumption

In [75]:

```
s',ylabel='Readings in watt-hour',title="Quarterly - Total watt-hour for Power Consumption"
```

Out[75]:

```
<AxesSubplot:title={'center':'Quarterly - Total watt-hour for Power Consumpt
ion'}, xlabel='Quarters', ylabel='Readings in watt-hour'>
```



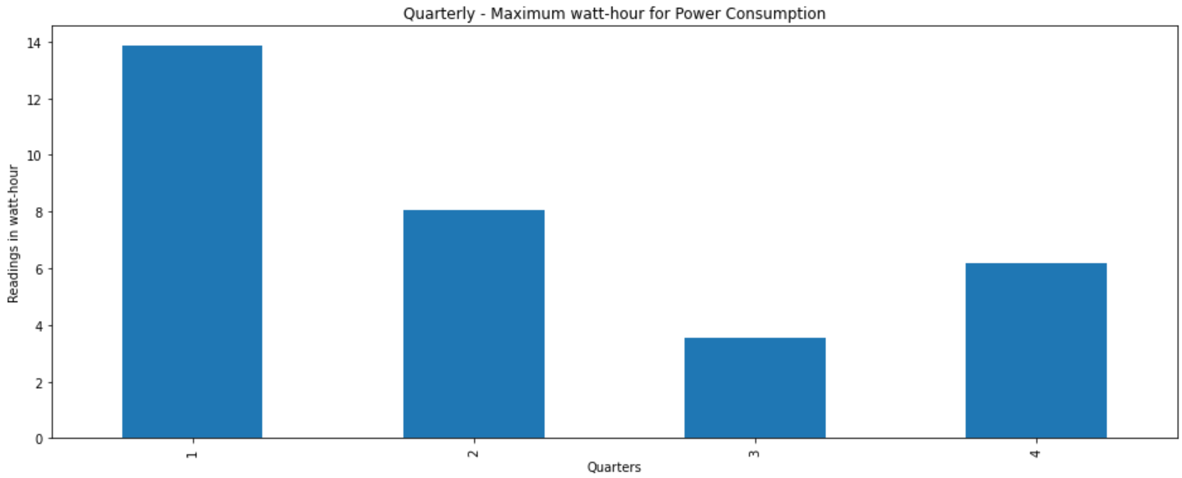
Quarterly - Maximum watt-hour for Power Consumption

In [76]:

```
df_power_consumption.groupby(df_power_consumption.index.quarter).max().plot(kind="bar",xlab
```

Out[76]:

```
<AxesSubplot:title={'center':'Quarterly - Maximum watt-hour for Power Consum
ption'}, xlabel='Quarters', ylabel='Readings in watt-hour'>
```



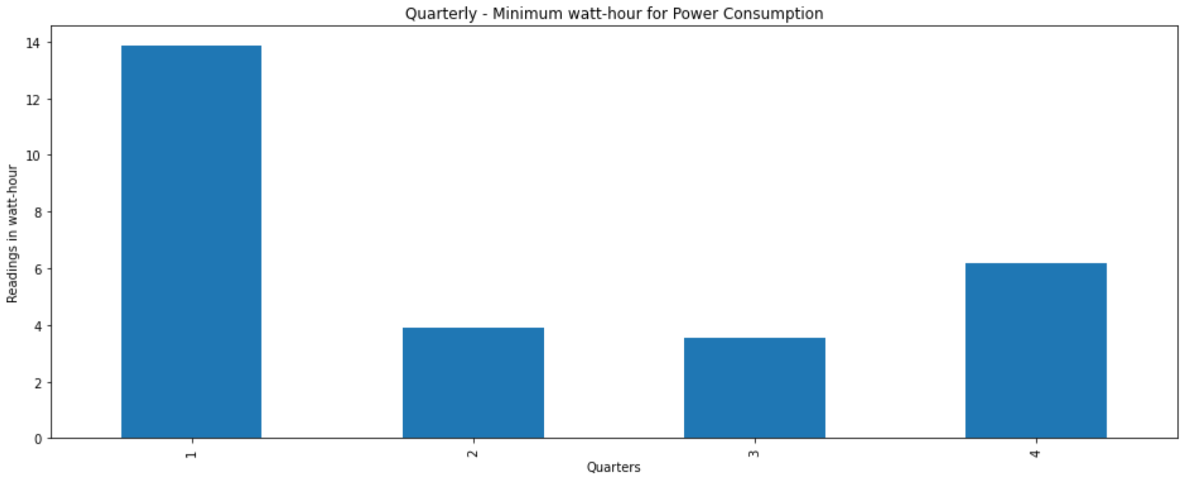
Quarterly - Minimum watt-hour for Power Consumption

In [77]:

```
,ylabel='Readings in watt-hour',title="Quarterly - Minimum watt-hour for Power Consumption",
```

Out[77]:

```
<AxesSubplot:title={'center':'Quarterly - Minimum watt-hour for Power Consum
ption'}, xlabel='Quarters', ylabel='Readings in watt-hour'>
```



Checking Statistical summary of power consumption weekly

In [78]:

```
df_power_consumption.groupby(df_power_consumption.index.week).describe()
```

Out[78]:

Date	count	mean	std	min	25%	50%	75%	max
3	1.0	13.866667	NaN	13.866667	13.866667	13.866667	13.866667	13.866667
25	2.0	5.966667	2.922708	3.900000	4.933333	5.966667	7.000000	8.033333
31	1.0	3.533333	NaN	3.533333	3.533333	3.533333	3.533333	3.533333
40	1.0	6.166667	NaN	6.166667	6.166667	6.166667	6.166667	6.166667

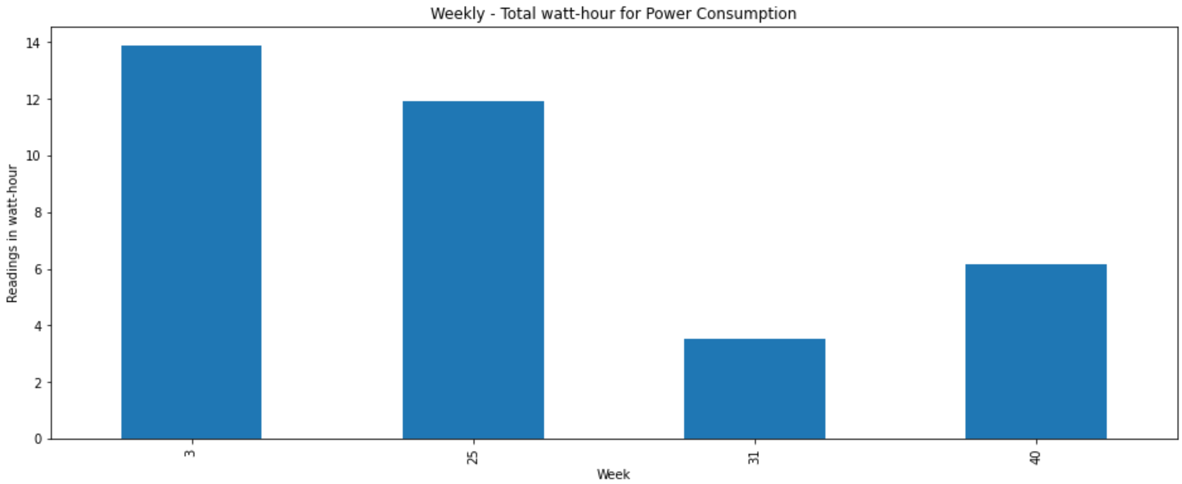
Weekly - Total watt-hour for Power Consumption

In [79]:

```
l='Week',ylabel='Readings in watt-hour',title="Weekly - Total watt-hour for Power Consumptio
```

Out[79]:

```
<AxesSubplot:title={'center':'Weekly - Total watt-hour for Power Consumptio
n'}, xlabel='Week', ylabel='Readings in watt-hour'>
```



Analysis of Sub_metering for yearly , quarterly, monthly and weekly

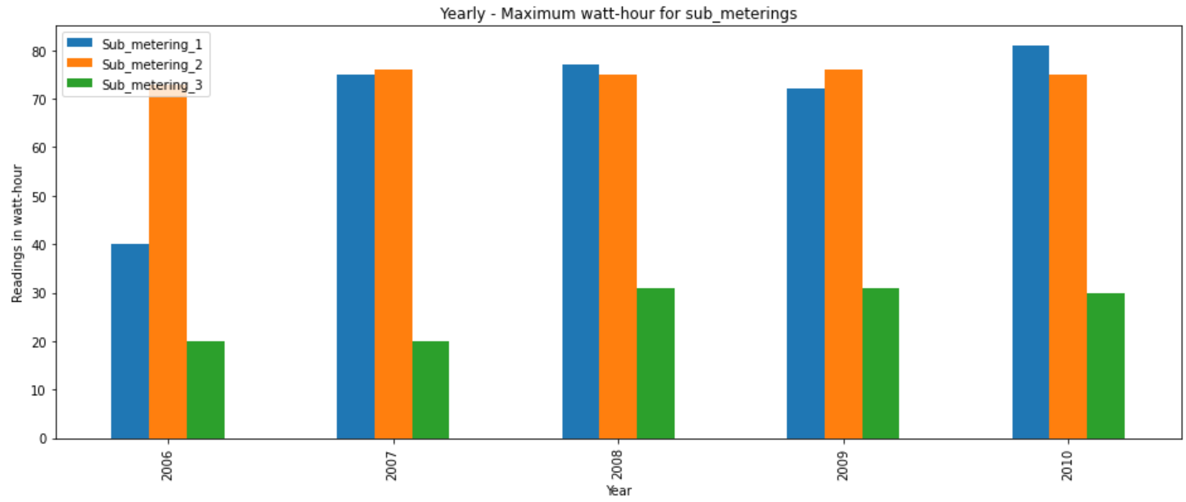
Yearly - Maximum watt-hour for sub_meterings

In [80]:

```
xlabel='Year',ylabel='Readings in watt-hour',title="Yearly - Maximum watt-hour for sub_mete
```

Out[80]:

```
<AxesSubplot:title={'center':'Yearly - Maximum watt-hour for sub_metering
s'}, xlabel='Year', ylabel='Readings in watt-hour'>
```



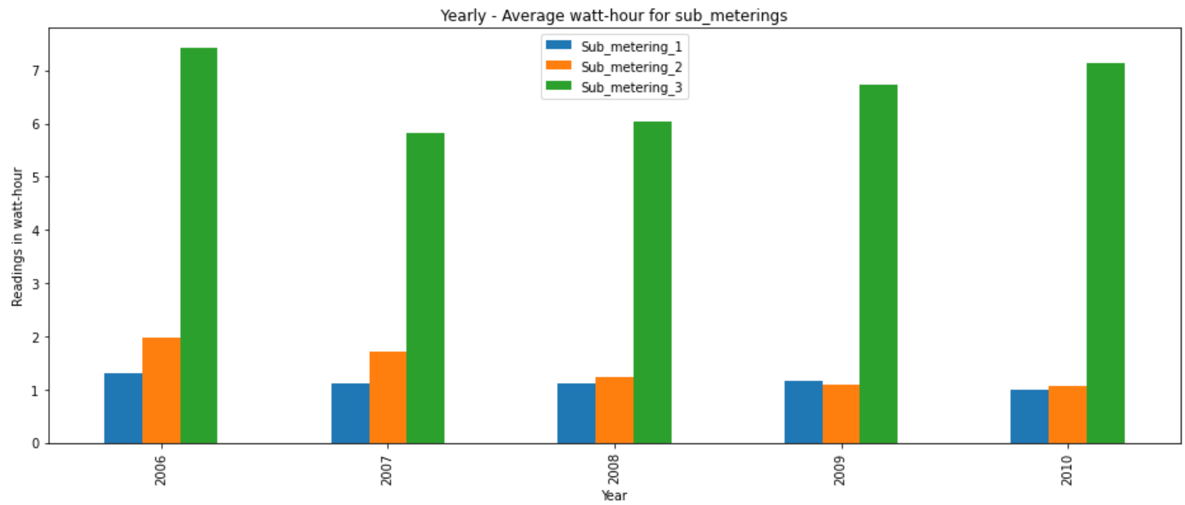
Yearly - Average watt-hour for sub_meterings

In [81]:

```
xlabel='Year',ylabel='Readings in watt-hour',title="Yearly - Average watt-hour for sub_mete
```

Out[81]:

```
<AxesSubplot:title={'center':'Yearly - Average watt-hour for sub_metering
s'}, xlabel='Year', ylabel='Readings in watt-hour'>
```



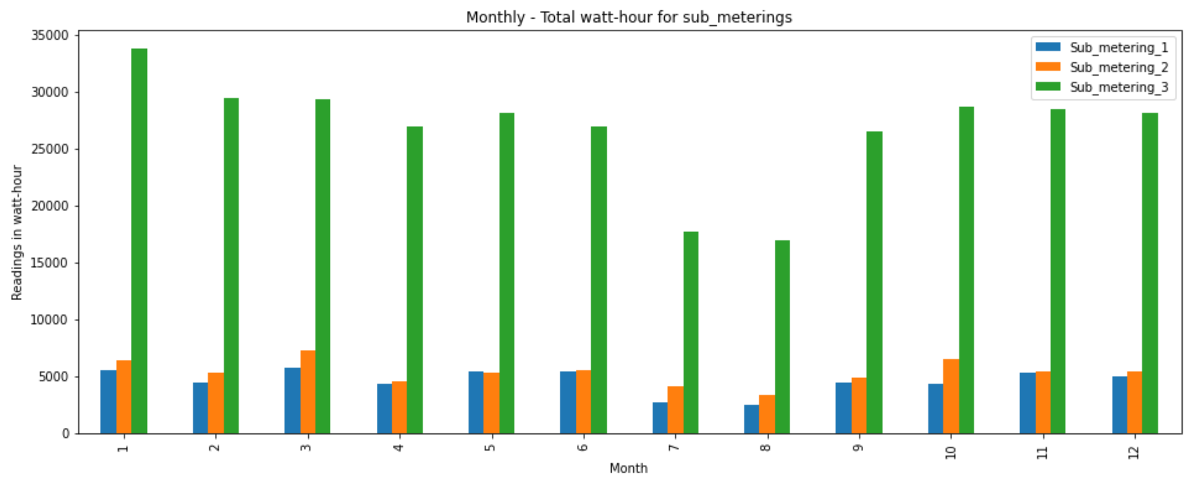
Monthly - total watt hour for sub_meterings

In [82]:

```
xlabel='Month',ylabel='Readings in watt-hour',title="Monthly - Total watt-hour for sub_mete
```

Out[82]:

```
<AxesSubplot:title={'center':'Monthly - Total watt-hour for sub_meterings'},
xlabel='Month', ylabel='Readings in watt-hour'>
```



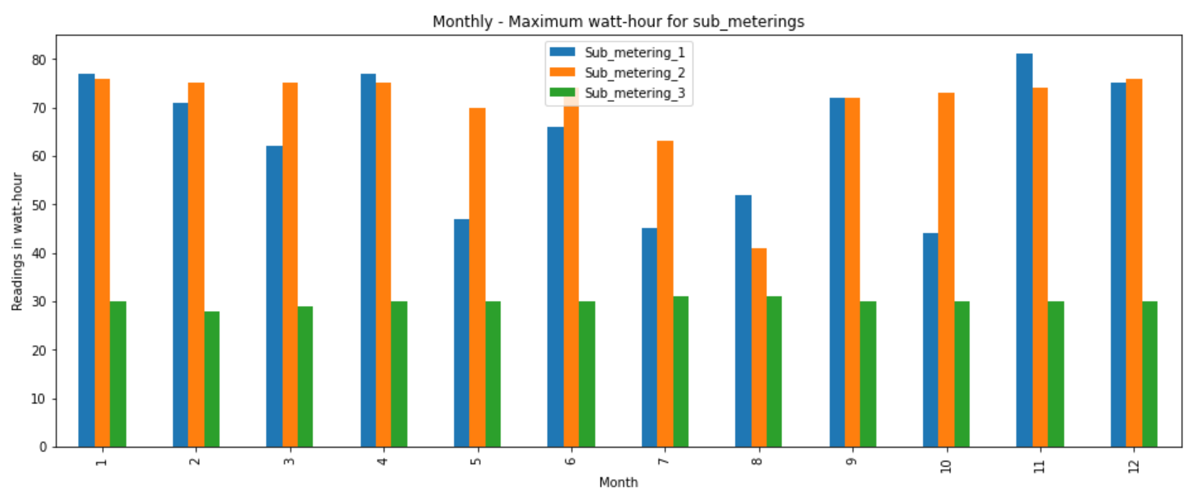
Monthly - Maximum watt-hour for sub_meterings

In [83]:

```
abel='Month',ylabel='Readings in watt-hour',title="Monthly - Maximum watt-hour for sub_meter
```

Out[83]:

```
<AxesSubplot:title={'center':'Monthly - Maximum watt-hour for sub_metering
s'}, xlabel='Month', ylabel='Readings in watt-hour'>
```



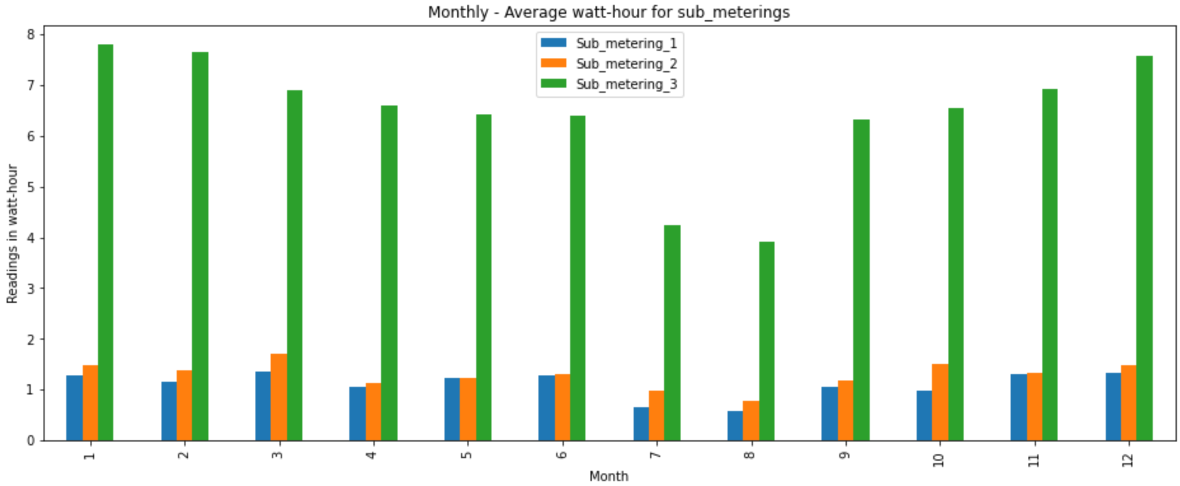
Monthly - Average watt-hour for sub_meterings

In [84]:

```
abel='Month',ylabel='Readings in watt-hour',title="Monthly - Average watt-hour for sub_meter
```

Out[84]:

```
<AxesSubplot:title={'center':'Monthly - Average watt-hour for sub_metering
s'}, xlabel='Month', ylabel='Readings in watt-hour'>
```



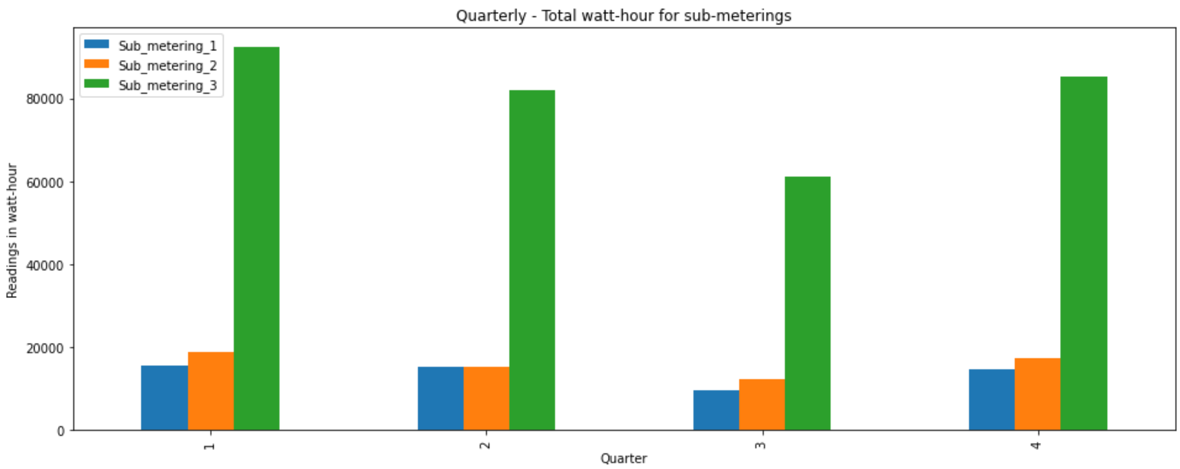
Quarterly - Total watt-hour for sub-meterings

In [85]:

```
l='Quarter',ylabel='Readings in watt-hour',figsize=(16,6),title="Quarterly - Total watt-hour
```

Out[85]:

```
<AxesSubplot:title={'center':'Quarterly - Total watt-hour for sub-metering
s'}, xlabel='Quarter', ylabel='Readings in watt-hour'>
```



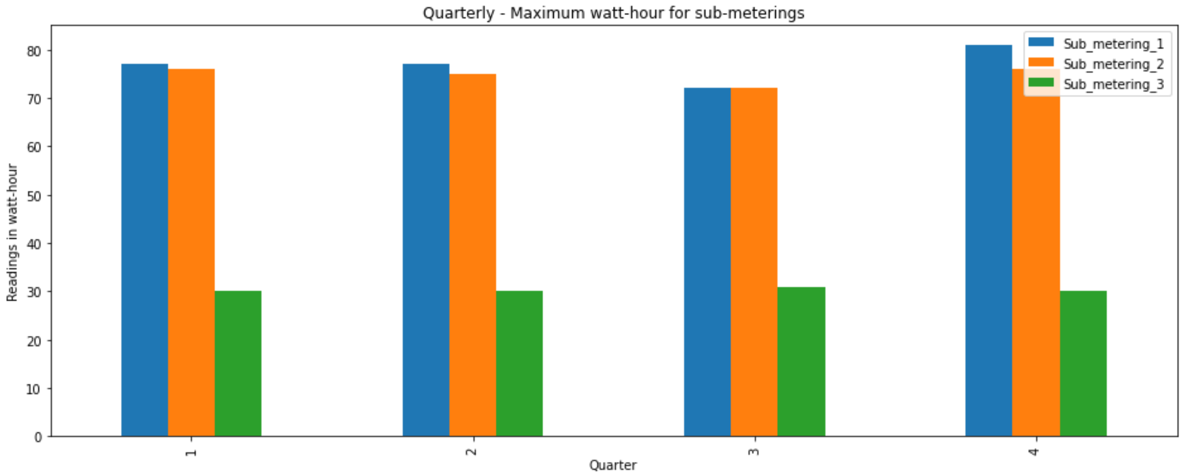
Quarterly - Maximum watt-hour for sub-meterings

In [86]:

```
Quarter',ylabel='Readings in watt-hour',figsize=(16,6),title="Quarterly - Maximum watt-hour
```

Out[86]:

```
<AxesSubplot:title={'center':'Quarterly - Maximum watt-hour for sub-metering
s'}, xlabel='Quarter', ylabel='Readings in watt-hour'>
```



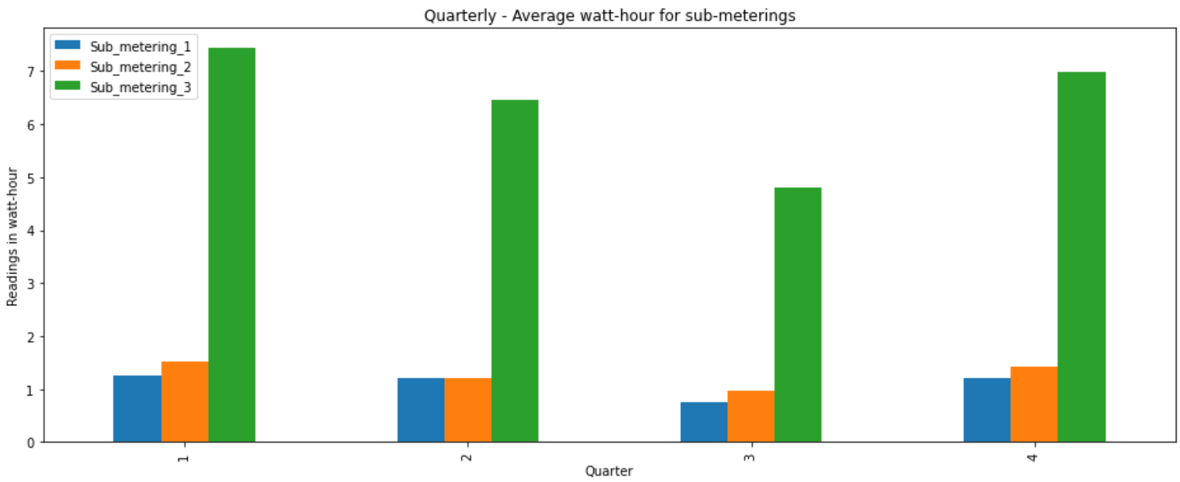
Quarterly - Average watt-hour for sub-meterings

In [87]:

```
Quarter',ylabel='Readings in watt-hour',figsize=(16,6),title="Quarterly - Average watt-hour
```

Out[87]:

```
<AxesSubplot:title={'center':'Quarterly - Average watt-hour for sub-metering
s'}, xlabel='Quarter', ylabel='Readings in watt-hour'>
```



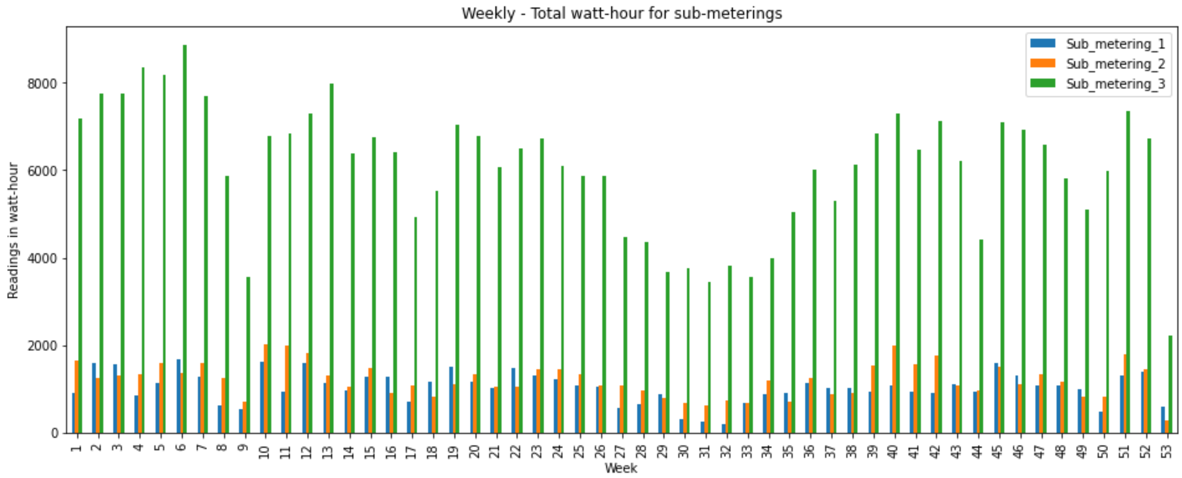
Weekly - Total watt-hour for sub-meterings

In [88]:

```
r",xlabel='Week',ylabel='Readings in watt-hour',figsize=(16,6),title="Weekly - Total watt-ho
```

Out[88]:

```
<AxesSubplot:title={'center':'Weekly - Total watt-hour for sub-meterings'},
xlabel='Week', ylabel='Readings in watt-hour'>
```



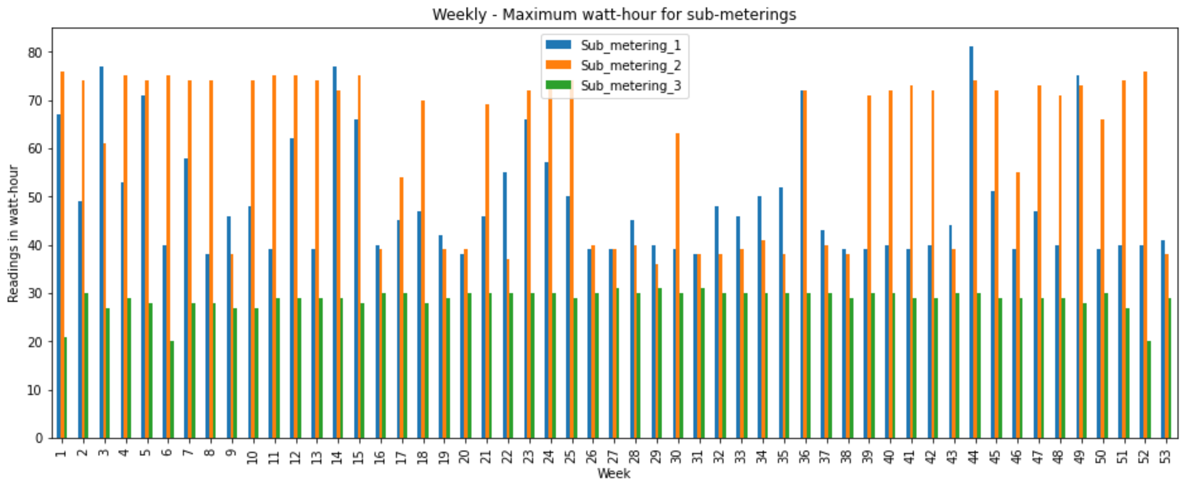
Weekly - Maximum watt-hour for sub-meterings

In [89]:

```
df_sub_meterings.groupby(df_sub_meterings.index.week).max().plot(kind="bar",xlabel='Week',y
```

Out[89]:

```
<AxesSubplot:title={'center':'Weekly - Maximum watt-hour for sub-metering
s'}, xlabel='Week', ylabel='Readings in watt-hour'>
```




```
Global_active_power, Global_reactive_power and Global Intensity analysis for Yearl
y, Quarterly, Monthly and Weekly
(Global_active_power and Global_reactive_power measured in kilowatt whereas, Globa
```

l_intensity measured in Ampere)

kilowatt = (ampere * volt) / 1000

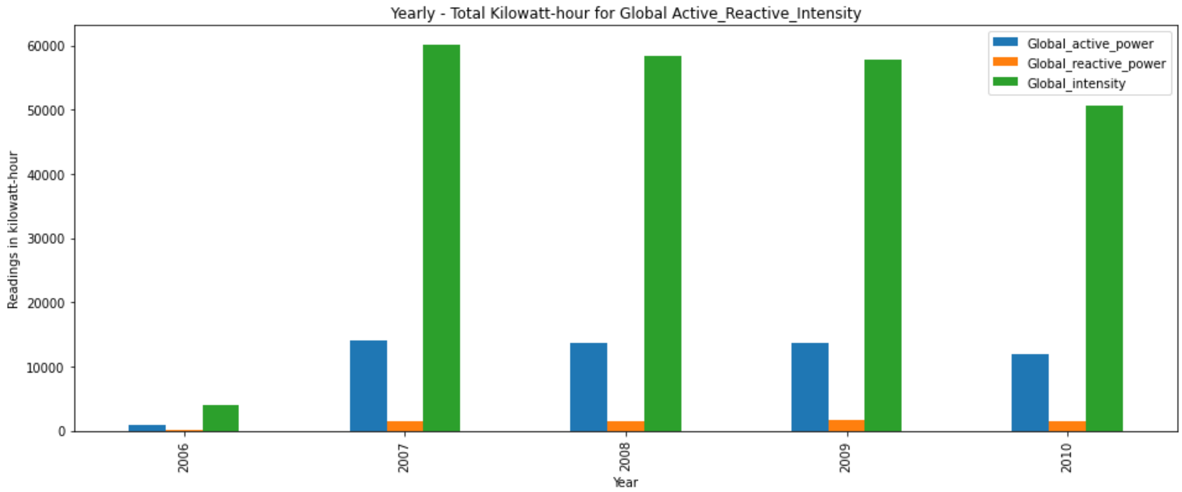
Yearly - Total Kilowatt-hour for Global Active_Reactive_Intensity

In [90]:

```
n kilowatt-hour',figsize=(16,6),title="Yearly - Total Kilowatt-hour for Global Active_React:
```

Out[90]:

```
<AxesSubplot:title={'center':'Yearly - Total Kilowatt-hour for Global Active
_Reactive_Intensity'}, xlabel='Year', ylabel='Readings in kilowatt-hour'>
```



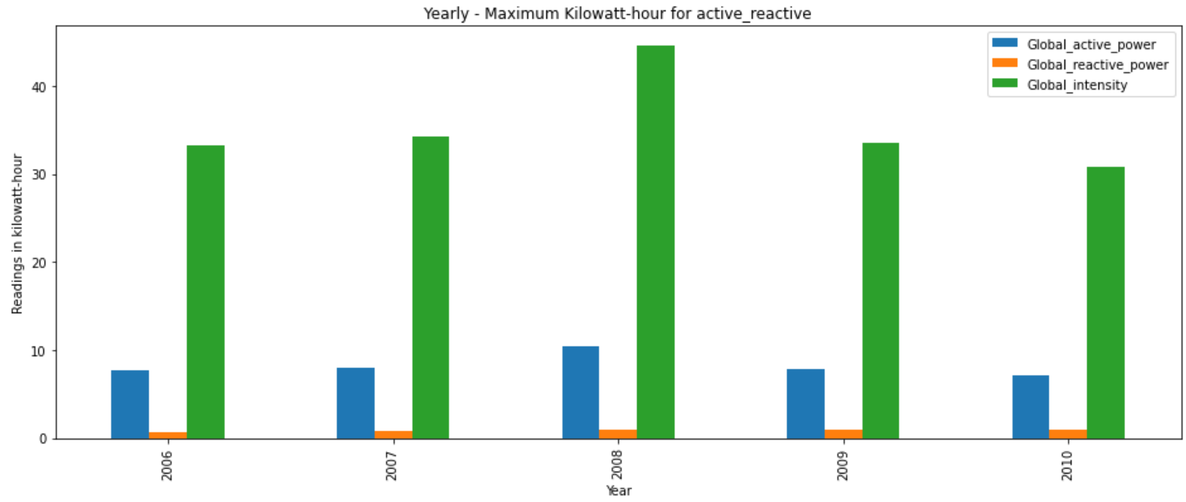
Yearly - Maximum Kilowatt-hour for active_reactive

In [91]:

```
,ylabel='Readings in kilowatt-hour',figsize=(16,6),title="Yearly - Maximum Kilowatt-hour for
```

Out[91]:

```
<AxesSubplot:title={'center':'Yearly - Maximum Kilowatt-hour for active_reac
tive'}, xlabel='Year', ylabel='Readings in kilowatt-hour'>
```



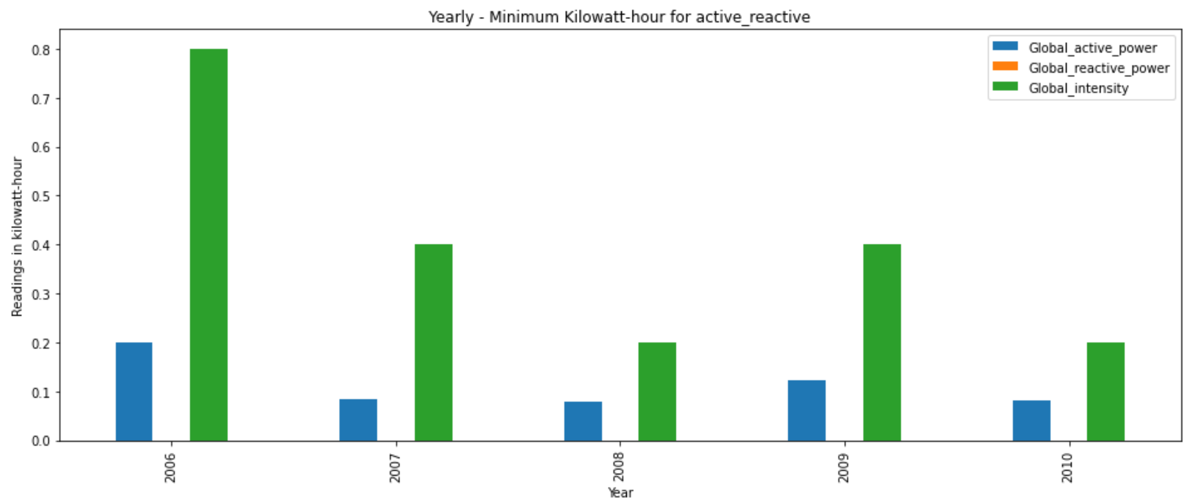
Yearly - Minimum Kilowatt-hour for active_reactive

In [92]:

```
,ylabel='Readings in kilowatt-hour',figsize=(16,6),title="Yearly - Minimum Kilowatt-hour for
```

Out[92]:

```
<AxesSubplot:title={'center':'Yearly - Minimum Kilowatt-hour for active_reac
tive'}, xlabel='Year', ylabel='Readings in kilowatt-hour'>
```



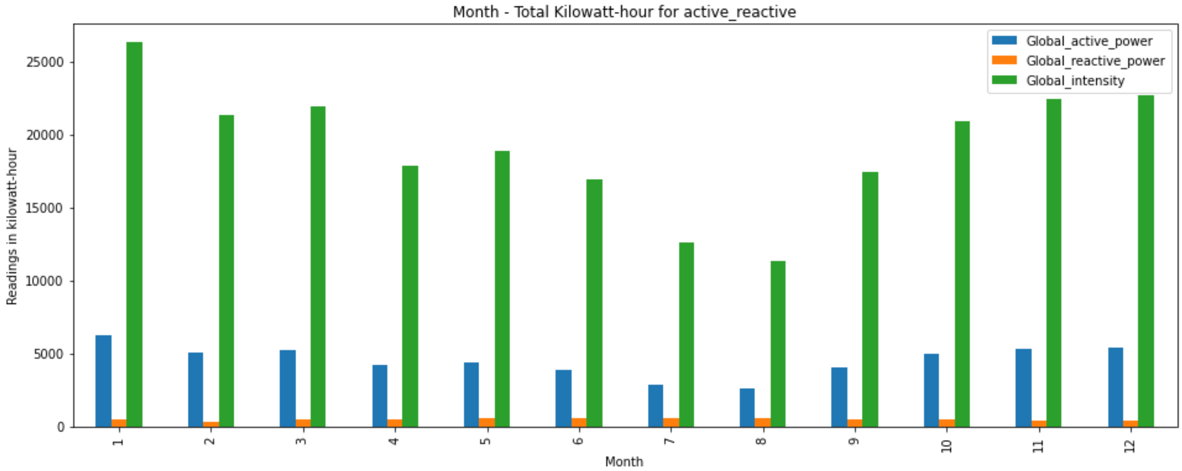
Month - Total Kilowatt-hour for active_reactive

In [93]:

```
th',ylabel='Readings in kilowatt-hour',figsize=(16,6),title="Month - Total Kilowatt-hour for
```

Out[93]:

```
<AxesSubplot:title={'center':'Month - Total Kilowatt-hour for active_reactiv
e'}, xlabel='Month', ylabel='Readings in kilowatt-hour'>
```



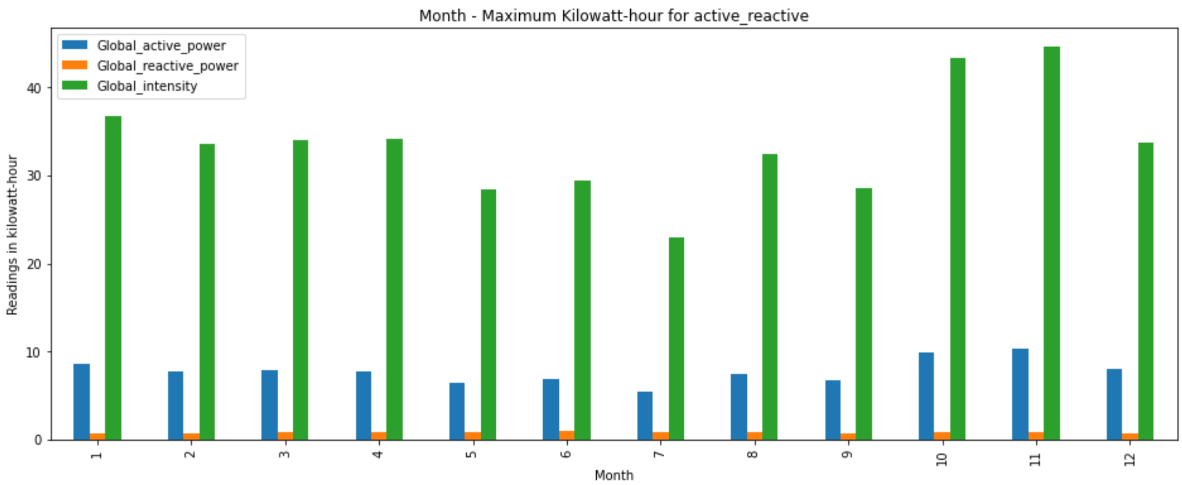
Month - Maximum Kilowatt-hour for active_reactive

In [94]:

```
',ylabel='Readings in kilowatt-hour',figsize=(16,6),title="Month - Maximum Kilowatt-hour for
```

Out[94]:

```
<AxesSubplot:title={'center':'Month - Maximum Kilowatt-hour for active_react
ive'}, xlabel='Month', ylabel='Readings in kilowatt-hour'>
```



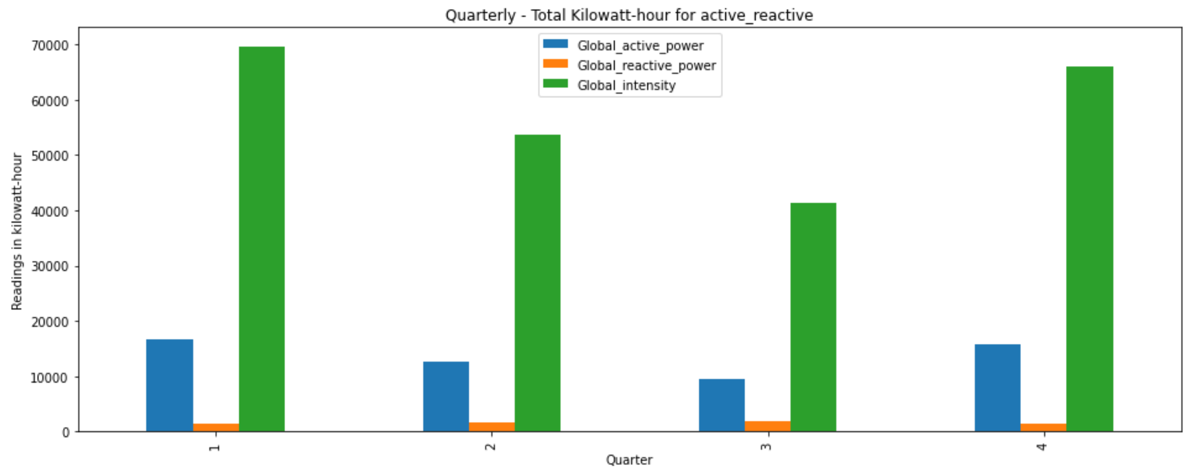
Quarterly - Total Kilowatt-hour for active_reactive

In [95]:

```
abel='Readings in kilowatt-hour',figsize=(16,6),title="Quarterly - Total Kilowatt-hour for a
```

Out[95]:

```
<AxesSubplot:title={'center':'Quarterly - Total Kilowatt-hour for active_rea
ctive'}, xlabel='Quarter', ylabel='Readings in kilowatt-hour'>
```



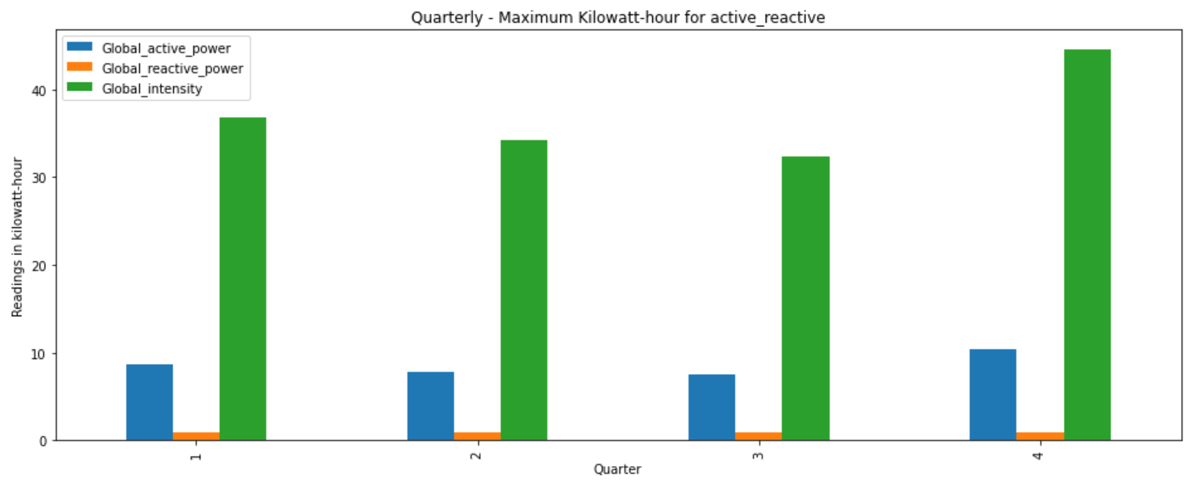
Quarterly - Maximum Kilowatt-hour for active_reactive

In [96]:

```
el='Readings in kilowatt-hour',figsize=(16,6),title="Quarterly - Maximum Kilowatt-hour for a
```

Out[96]:

```
<AxesSubplot:title={'center':'Quarterly - Maximum Kilowatt-hour for active_r
eactive'}, xlabel='Quarter', ylabel='Readings in kilowatt-hour'>
```



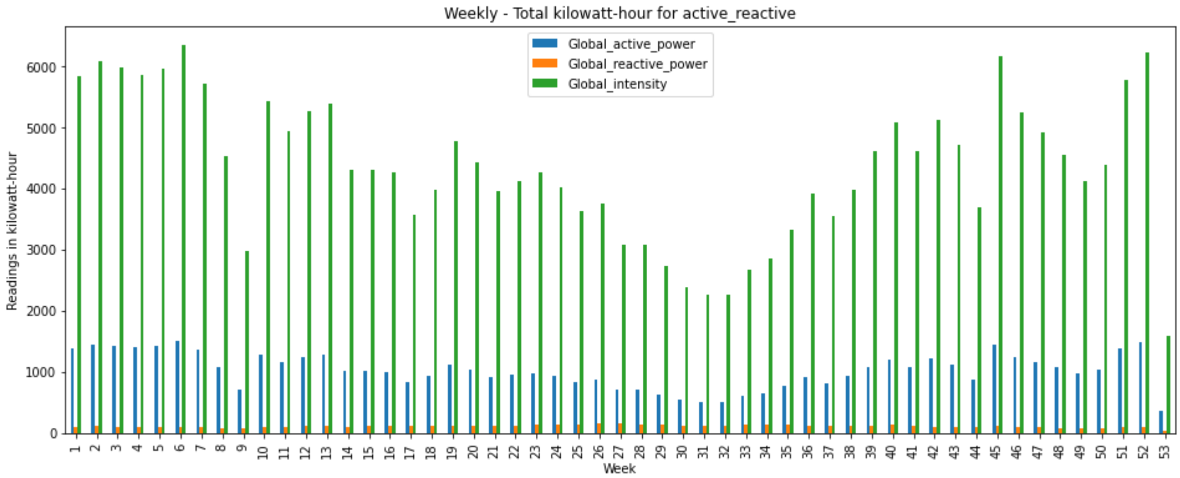
Weekly - Total kilowatt-hour for active_reactive

In [97]:

```
k',ylabel='Readings in kilowatt-hour',figsize=(16,6),title="Weekly - Total kilowatt-hour fo
```

Out[97]:

```
<AxesSubplot:title={'center':'Weekly - Total kilowatt-hour for active_reacti
ve'}, xlabel='Week', ylabel='Readings in kilowatt-hour'>
```



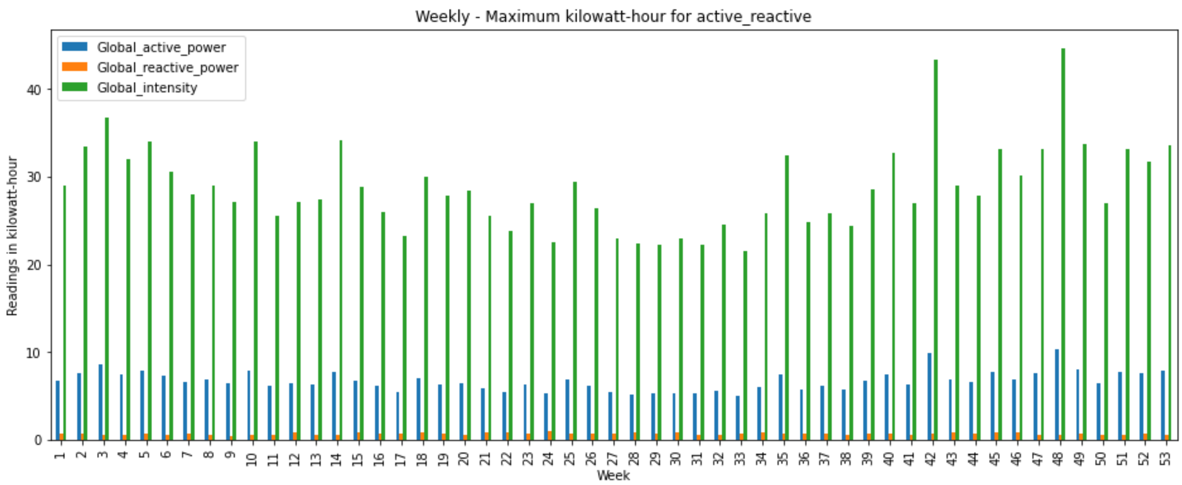
Weekly - Maximum kilowatt-hour for active_reactive

In [98]:

```
,ylabel='Readings in kilowatt-hour',figsize=(16,6),title="Weekly - Maximum kilowatt-hour for
```

Out[98]:

```
<AxesSubplot:title={'center':'Weekly - Maximum kilowatt-hour for active_reac
tive'}, xlabel='Week', ylabel='Readings in kilowatt-hour'>
```



Pair plot to see the relationship between variables in dataset

In [99]:

```
#sns.pairplot(data=df_sub_meterings,kind="scatter")
```

Pair plot active recative

In [100]:

```
#sns.pairplot(data=df_active_reactive,kind="scatter")
```

In [ ]:

```

```

In [112]:

```
dfs
```

Out[112]:

	Date	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_meteri
0	2009-06-16	0.234	0.112	241.93	1.2	
1	2007-08-01	0.212	0.112	235.87	1.0	
2	2010-10-10	0.490	0.274	240.10	2.2	
3	2009-06-18	0.542	0.058	241.25	2.2	
4	2010-01-18	2.032	0.132	242.42	8.4	
...	...	...	...	...	...	
49995	2008-06-20	0.328	0.254	242.66	1.6	
49996	2009-09-29	4.042	0.068	236.18	17.0	
49997	2010-04-26	0.340	0.114	241.75	1.4	
49998	2007-04-10	0.500	0.108	240.27	2.4	
49999	2007-11-30	0.406	0.220	247.63	1.8	

50000 rows × 10 columns

In [113]:

```
dfs.drop(['Date','time'],axis=1,inplace=True)
```

In [134]:

```
dfs.head()
```

Out[134]:

	level_0	index	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_m
0	0	0	0.234	0.112	241.93	1.2	
1	1	1	0.212	0.112	235.87	1.0	
2	2	2	0.490	0.274	240.10	2.2	
3	3	3	0.542	0.058	241.25	2.2	
4	4	4	2.032	0.132	242.42	8.4	

In [ ]:

```

```

In [243]:

```
#pip install --upgrade pip
```

Uploding data to MongoDB

In [244]:

```
#pip install pymongo
```

In [245]:

```
#pip install "pymongo[srv]"
```

In [105]:

```
import pymongo
```

In [135]:

```

client = pymongo.MongoClient("mongodb+srv://nameuser:password@cluster0.dyoaqq5.mongodb.net/
```

we are Creating database and collection in MongoDB

In [138]:

```
db=client['House_h_p']
colle = db['Household_p_c_d']
```

In [139]:

```
### Converting dataframe to dict so it can be uploaded to MongoDB
dfs.reset_index(inplace=True)
data_dict = dfs.to_dict("records")
```

In [140]:

```
colle.insert_many(data_dict)
```

Out[140]:

```
<pymongo.results.InsertManyResult at 0x1d8a1327820>
```

Load data from MongoDB to pandas dataframe

In [142]:

```
data_from_mongodb=collection.find()
### converting data from MongoDb to Dataframe in pandas
df2=pd.DataFrame(data_from_mongodb)
### first 5 records in dataset
df2.head()
```

Out[142]:

	_id	level_0	index	Time	Global_active_power	Global_reactive_power
0	636a4f4e9a42fdfaaaa93c72	9.0	9	160.0	0.432	0.238
1	636a4f4e9a42fdfaaaa93c69	0.0	0	1064.0	1.632	0.532
2	636a4f4e9a42fdfaaaa93c74	11.0	11	329.0	0.326	0.114
3	636a4f4e9a42fdfaaaa93c85	28.0	28	361.0	0.246	0.108
4	636a4f4e9a42fdfaaaa93c8b	34.0	34	1049.0	0.984	0.212

In [122]:

```
### Drop the id and level
```

In [154]:

```
df2.drop(['_id','level_0','index',],axis=1,inplace=True)
```

In [143]:

```
df2.isnull().sum()
```

Out[143]:

```
_id                          0
level_0                  79617
index                        0
Time                     50000
Global_active_power          0
Global_reactive_power        0
Voltage                      0
Global_intensity             0
day                      50000
month                    50000
year                     50000
Sub_metering             79630
Sub_metering_1           14805
Sub_metering_2           14805
Sub_metering_3           14805
power_consumption        29607
dtype: int64
```

In [144]:

```
df2.ffill(axis=0,inplace=True)
```

In [148]:

```
df2.isnull().sum().sum()
```

Out[148]:

```
74022
```

In [147]:

```
df2.ffill(axis=0,inplace=True)
```

In [156]:

```
df2.dropna(inplace=True)
```

In [157]:

```
df2.head()
```

Out[157]:

bal_reactive_power	Voltage	Global_intensity	day	month	year	Sub_metering	Sub_metering_1
0.096	239.22	3.2	8.0	7.0	2008.0	18.0	0.0
0.600	240.00	12.0	8.0	1.0	2007.0	18.0	0.0
0.052	239.99	2.2	14.0	9.0	2008.0	18.0	0.0
0.000	236.50	0.6	14.0	8.0	2007.0	18.0	0.0
0.202	239.26	1.8	9.0	11.0	2010.0	18.0	0.0

In [158]:

```
df2.drop(['Sub_metering'],axis=1,inplace=True)
```

In [ ]:

```

```

Independent and Dependent feature seperation

independent feature

In [159]:

```
X = df2.drop("power_consumption",axis=1)
```

dependent feature

In [161]:

```
y = df2['power_consumption']
```

In [162]:

```
### Checking
```

In [163]:

```
X.head(3)
```

Out[163]:

	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	day	mor
29607	609.0	0.764	0.096	239.22	3.2	8.0	7
29608	991.0	2.786	0.600	240.00	12.0	8.0	1
29609	1058.0	0.480	0.052	239.99	2.2	14.0	9

In [164]:

```
y.head(3)
```

Out[164]:

```
29607    12.733333
29608    29.433333
29609     8.000000
Name: power_consumption, dtype: float64
```

In [165]:

```
## Spliting the data into train and split
```

In [166]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33,random_state=42)
X_train.shape
```

Out[166]:

```
(53352, 11)
```

In [167]:

```
y_train.shape
```

Out[167]:

```
(53352,)
```

In [168]:

```
X_test.shape
```

Out[168]:

```
(26278, 11)
```

In [169]:

```
y_test.shape
```

Out[169]:

```
(26278,)
```

Standardizing or Feature scalling the dataset

In [171]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler
```

Out[171]:

```
▾ StandardScaler
StandardScaler()
```

In [172]:

```
## Apply
```

In [173]:

```
X_train = scaler.fit_transform(X_train)
```

In [174]:

```
X_test = scaler.transform(X_test)
```

In [175]:

```
X_train
```

Out[175]:

```
array([[-0.5372009 ,  1.13200147,  0.92108983, ..., -0.18412197,
        -0.22276164, -0.77201723],
       [ 1.82341055, -0.76167644, -0.28418035, ..., -0.18412197,
        -0.22276164, -0.77201723],
       [-0.82376552, -0.72402876,  0.88564071, ..., -0.18412197,
        -0.22276164, -0.65370524],
       ...,
       [-0.5372009 , -0.77485312,  1.08061088, ..., -0.18412197,
        -0.22276164, -0.77201723],
       [ 0.08132469,  2.66049695, -1.09951017, ..., -0.18412197,
        -0.22276164,  1.3575986 ],
       [-0.12863354, -0.77485312, -1.09951017, ..., -0.18412197,
        -0.22276164, -0.65370524]])
```

In [176]:

```
X_test
```

Out[176]:

```
array([[-0.5372009 , -0.53202564, -0.12465929, ..., -0.18412197,
        -0.22276164, -0.77201723],
       [-0.5372009 , -0.80120649, -0.55004877, ..., -0.18412197,
        -0.22276164, -0.77201723],
       [ 1.786526  , -0.81061841, -0.12465929, ..., -0.18412197,
        -0.22276164, -0.77201723],
       ...,
       [ 0.04160287, -0.80685364, -0.10693473, ..., -0.18412197,
        -0.22276164, -0.65370524],
       [-0.5372009 ,  1.73059945, -0.53232421, ..., -0.18412197,
        -0.22276164,  1.23928661],
       [-0.5372009 , -0.60167383,  0.69067053, ..., -0.18412197,
        -0.22276164, -0.77201723]])
```

Model Training

In [179]:

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression
```

Out[179]:

```
▾ LinearRegression
LinearRegression()
```

In [180]:

```
regression.fit(X_train,y_train)
```

Out[180]:

```
▾ LinearRegression
LinearRegression()
```

Coefficient and intercept

In [182]:

```
regression.coef_
```

Out[182]:

```
array([-3.37067127e-15,  1.77080449e+01, -1.11022302e-15, -6.66133815e-15,
       -7.81597009e-14,  1.77635684e-15,  3.55271368e-15,  5.32907052e-15,
       -6.14998548e+00, -5.78420515e+00, -8.45222869e+00])
```

In [183]:

```
regression.intercept_
```

Out[183]:

```
9.375021242565104
```

In [184]:

```
reg_pred = regression.predict(X_test)
reg_pred
```

Out[184]:

```
array([ 8.9       ,  4.13333333,  3.96666667, ...,  3.03333333,
       31.96666667,  7.66666667])
```

Performance Metrics

R Squared

In [188]:

```
from sklearn.metrics import r2_score
linear_score = r2_score(y_test,reg_pred)
linear_score
```

Out[188]:

```
1.0
```

Adjusted R Squared

In [190]:

```
1 - (1-linear_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[190]:

```
1.0
```

Ridge Regression:

Train the Model

In [191]:

```
from sklearn.linear_model import Ridge
ridgeR = Ridge(alpha=.99)
ridgeR.fit(X_train,y_train)
```

Out[191]:

```
▾      Ridge
Ridge(alpha=0.99)
```

Coefficient and Intercept

In [194]:

```
ridgeR.coef_
```

Out[194]:

```
array([ 4.02409552e-04,  1.74913774e+01, -4.01874065e-03,  2.98363765e-03,
        2.16242564e-01,  1.38547656e-04,  6.82834768e-05, -1.40043546e-04,
       -6.15011355e+00, -5.78446596e+00, -8.44804491e+00])
```

In [195]:

```
ridgeR.intercept_
```

Out[195]:

```
9.375021242565104
```

Prediction

In [197]:

```
ridgeR_pred = ridgeR.predict(X_test)
ridgeR_pred
```

Out[197]:

```
array([ 8.89397503,  4.1285594 ,  3.96177063, ...,  3.02826943,
       31.96976315,  7.6642777 ])
```

Performance Metrics

R Square

In [201]:

```
from sklearn.metrics import r2_score
ridgeR_score = r2_score(y_test,ridgeR_pred)
```

In [202]:

```
print(ridgeR_score)
```

0.999999234147855

Adjusted R Squared

In [204]:

```
1-(1-ridgeR_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[204]:

0.999999233827122

In [ ]:

```

```

Lasso Regression

Train the model

In [207]:

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.05)
lasso.fit(X_train,y_train)
```

Out[207]:

```
▾ Lasso
Lasso(alpha=0.05)
```

Coefficient and Intercept

In [209]:

```
print(lasso.coef_)
```

```
[ 0.         17.17990799  0.          -0.          0.1143701   0.
 -0.         -0.         -5.93952874 -5.58995185 -8.17306296]
```

In [210]:

```
print(lasso.intercept_)
```

9.375021242565104

Prediction

In [212]:

```
lasso_pred = lasso.predict(X_test)
lasso_pred
```

Out[212]:

```
array([ 8.82046796,  4.16524899,  4.0035531 , ...,  3.10126011,
       31.51473746,  7.61879897])
```

Performance Metrics

In [214]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,lasso_pred))
print(mean_absolute_error(y_test,lasso_pred))
print(np.sqrt(mean_squared_error(y_test,lasso_pred)))
```

```
0.053705706092843745
0.14933996049248893
0.23174491600215041
```

R Square

In [216]:

```
from sklearn.metrics import r2_score
lasso_score = r2_score(y_test,lasso_pred)
```

In [217]:

```
lasso_score
```

Out[217]:

```
0.9994080190161271
```

Adjusted R Square

In [219]:

```
1 - (1-lasso_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[219]:

```
0.9994077710990167
```

ElasticNet Regression

Train the model

In [221]:

```
from sklearn.linear_model import ElasticNet
elas_net = ElasticNet(alpha=0.2,l1_ratio=.2)
elas_net.fit(X_train,y_train)
```

Out[221]:

```
▾ ElasticNet
ElasticNet(alpha=0.2, l1_ratio=0.2)
```

Coefficient and Intercept

In [223]:

```
print(elas_net.coef_)
```

```
[ 0.32948216  5.6307423   0.22373443 -0.21744713  5.58423546  0.01409332
  0.         -0.21793364 -3.17180191 -3.06839505 -4.14146025]
```

In [224]:

```
print(elas_net.intercept_)
```

```
9.37502124256512
```

Prediction

In [226]:

```
elastic_pred = elas_net.predict(X_test)
elastic_pred
```

Out[226]:

```
array([ 7.6561681 ,  4.59687381,  4.95948975, ...,  4.08315031,
       25.04036028,  7.08701824])
```

Performance Metrics

R Square

In [229]:

```
from sklearn.metrics import r2_score
elastic_score = r2_score(y_test,elastic_pred)
print(elastic_score)
```

```
0.8684523076802381
```

Adjusted R Square

In [231]:

```
1 - (1-elastic_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[231]:

```
0.8683972165123588
```

In [ ]:

```

```

SVM (SVR)

In [232]:

```
from sklearn.svm import SVR
```

In [233]:

```
svr =  SVR()
```

In [234]:

```
svr.fit(X_train,y_train)
```

Out[234]:

```
▾ SVR
SVR()
```

In [235]:

```
svr_predict = svr.predict(X_test)
svr_predict
```

Out[235]:

```
array([ 8.84476323,  4.0674052 ,  3.81346357, ...,  2.98007868,
       32.04685699,  7.62600578])
```

Performance Metrics

In [237]:

```
from sklearn.metrics import r2_score
```

R Square

In [240]:

```
svr_r2_score = r2_score(y_test,svr_predict)
svr_r2_score
```

Out[240]:

0.9751668281104175

Adjusted R Square

In [242]:

```
1 - (1-svr_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[242]:

0.9751564281678764

In [ ]:

```
```