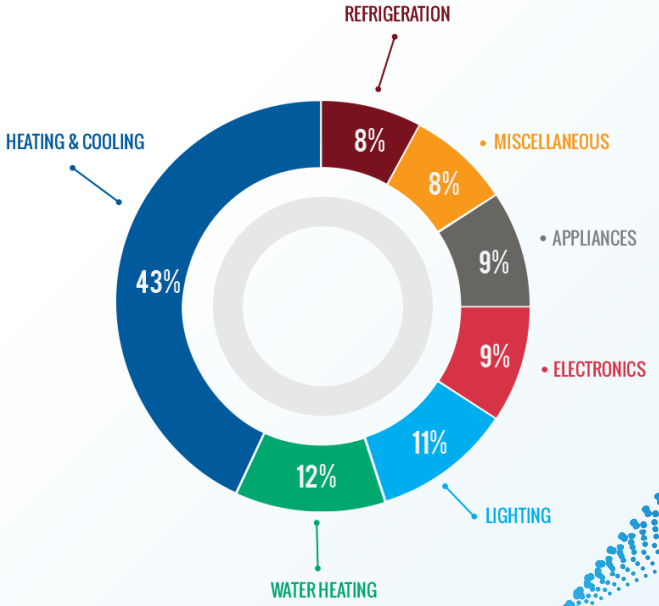
Big Data Fundamentals

Household Electric Power Consumption



Dataset

- Measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years.

- This dataset contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months).

Attributes

1. date
2. time
3. global_active_power
4. global_reactive_power
5. voltage
6. global_intensity
7. sub_metering_1
8. sub_metering_2
9. sub_metering_3

Attributes Explanation

Global Intensity	Global Active Power	Global Reactive Power
The electrical charge that passes through a section of the conductor per second.	The active power is the real power which is consumed by all the electrical loads	The power which moves back and forth between the load and source which is useless
Electric Charge(Q)/ Time(s)	Voltage (V) * Current (I) * Power Factor (PF)	Voltage (V) * Current (I) * Sine of Power Factor Angle (θ)
Ampere	Watts	VAR
I	P	Q
Ammeter	Wattmeter	VAR Meter

Attributes Explanation

Sub_metering_1	Sub_metering_2	Sub_metering_3
Kitchen	Laundry Room	Bath and Bedroom

Understanding the dataset

About dataset

This dataset contains 2075259 measurements (2 Million) gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months).

```
In [4]: df.shape
Out[4]: (2075259, 9)

In [5]: df.columns
Out[5]: Index(['Date', 'Time', 'Global_active_power', 'Global_reactive_power',
               'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
               'Sub_metering_3'],
              dtype='object')
```

Data type

```
In [6]: #data type of each column
        df.dtypes
```

```
Out[6]: Date                    object
        Time                    object
        Global_active_power     object
        Global_reactive_power   object
        Voltage                 object
        Global_intensity        object
        Sub_metering_1          object
        Sub_metering_2          object
        Sub_metering_3         float64
        dtype: object
```

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

Out[7]:

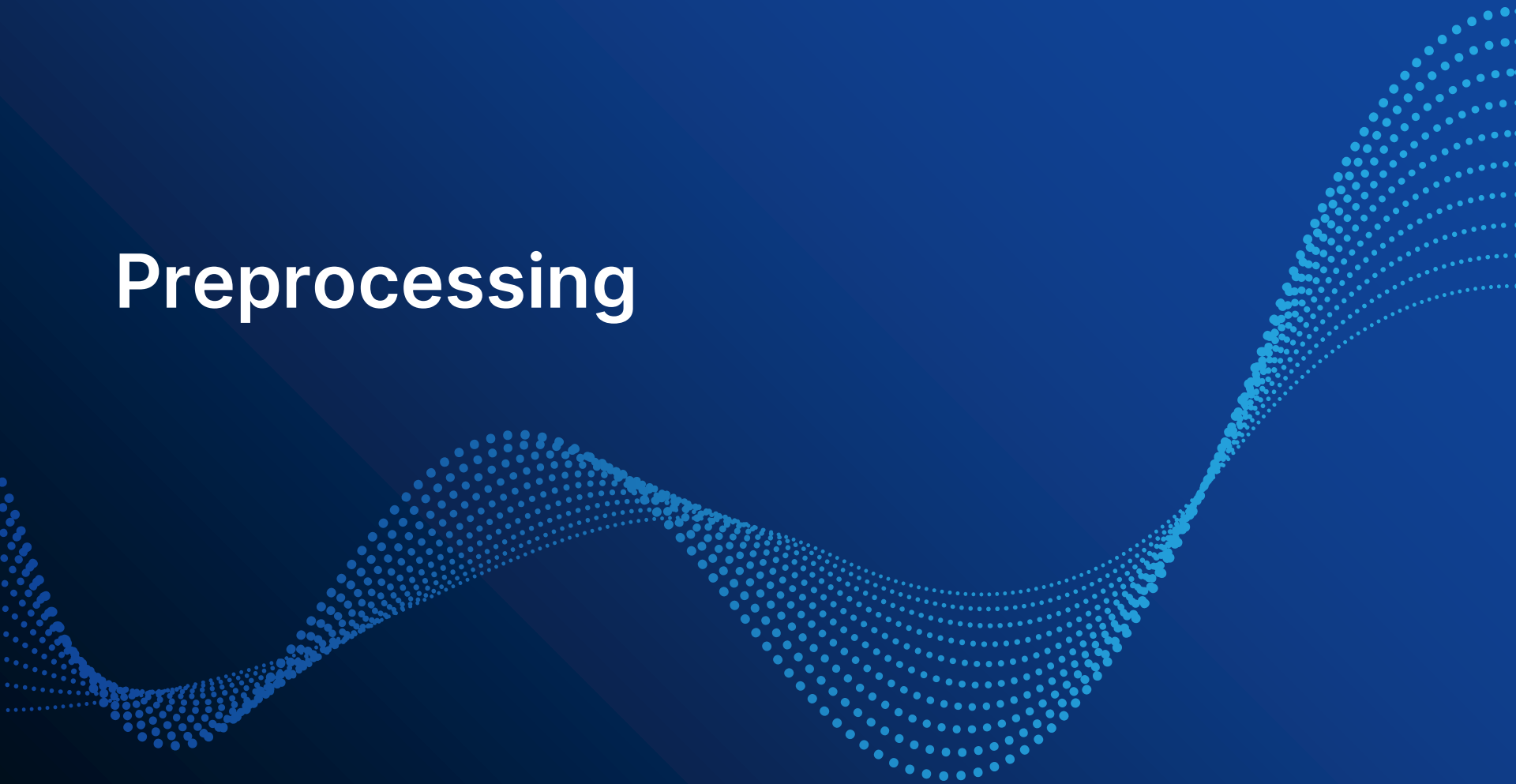
	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	16/12/2006	17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17.0
1	16/12/2006	17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16.0
2	16/12/2006	17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17.0
3	16/12/2006	17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17.0
4	16/12/2006	17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17.0

Define the target column and predictors

In [8]:
```
TARGET = "Global_active_power"
columns_predictors = [col for col in df.columns if col not in [TARGET]]

print(f"TARGET: {TARGET}")
print(f"Columns predictors: {columns_predictors}")
```

```
TARGET: Global_active_power
Columns predictors: ['Date', 'Time', 'Global_reactive_power', 'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2
', 'Sub_metering_3']
```

Preprocessing



Handling missing value

Handling misiing value

In [9]:
```
# Count the number of null values
df.isnull().sum()
```

Out[9]:
```
Date                       0
Time                       0
Global_active_power        0
Global_reactive_power      0
Voltage                    0
Global_intensity           0
Sub_metering_1             0
Sub_metering_2             0
Sub_metering_3         25979
dtype: int64
```

In [10]:
```
# is the total number of rows in df that contain at least one missing value.
df.isnull().any(axis = 1).sum()
```

Out[10]:
```
25979
```

Using Mean imputation method for missing value

In [12]:
```
# m and n are assigned the number of rows and columns
m, n = df.shape

# Computes the total number of missing values in the DataFrame using df.isnull().sum().sum()
# Divides this total by the number of rows (m) and assigns the result to the variable df_per
df_per = (df.isnull().sum().sum())/m
col_pers = {}

# Determines the number of missing values in the current column (i)
for i in df.columns:
    col_pers[i] = (df[i].isnull().sum())/m

print(df_per)
print(col_pers)
```

```
0.012518437457686004
{'Date': 0.0, 'Time': 0.0, 'Global_active_power': 0.0, 'Global_reactive_power': 0.0, 'Voltage': 0.0, 'Global_intensity': 0.0, '
Sub_metering_1': 0.0, 'Sub_metering_2': 0.0, 'Sub_metering_3': 0.012518437457686004}
```

```
# Use Simple Imputer for Mean imputation
imp = SimpleImputer(missing_values=-1, strategy='mean')
df[num_vars] = imp.fit_transform(df[num_vars])
```

In [13]:
```
# Count the number of null values
df.isnull().sum()
```

Out[13]:
```
Date                     0
Time                     0
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
```

Converting the data type of the columns

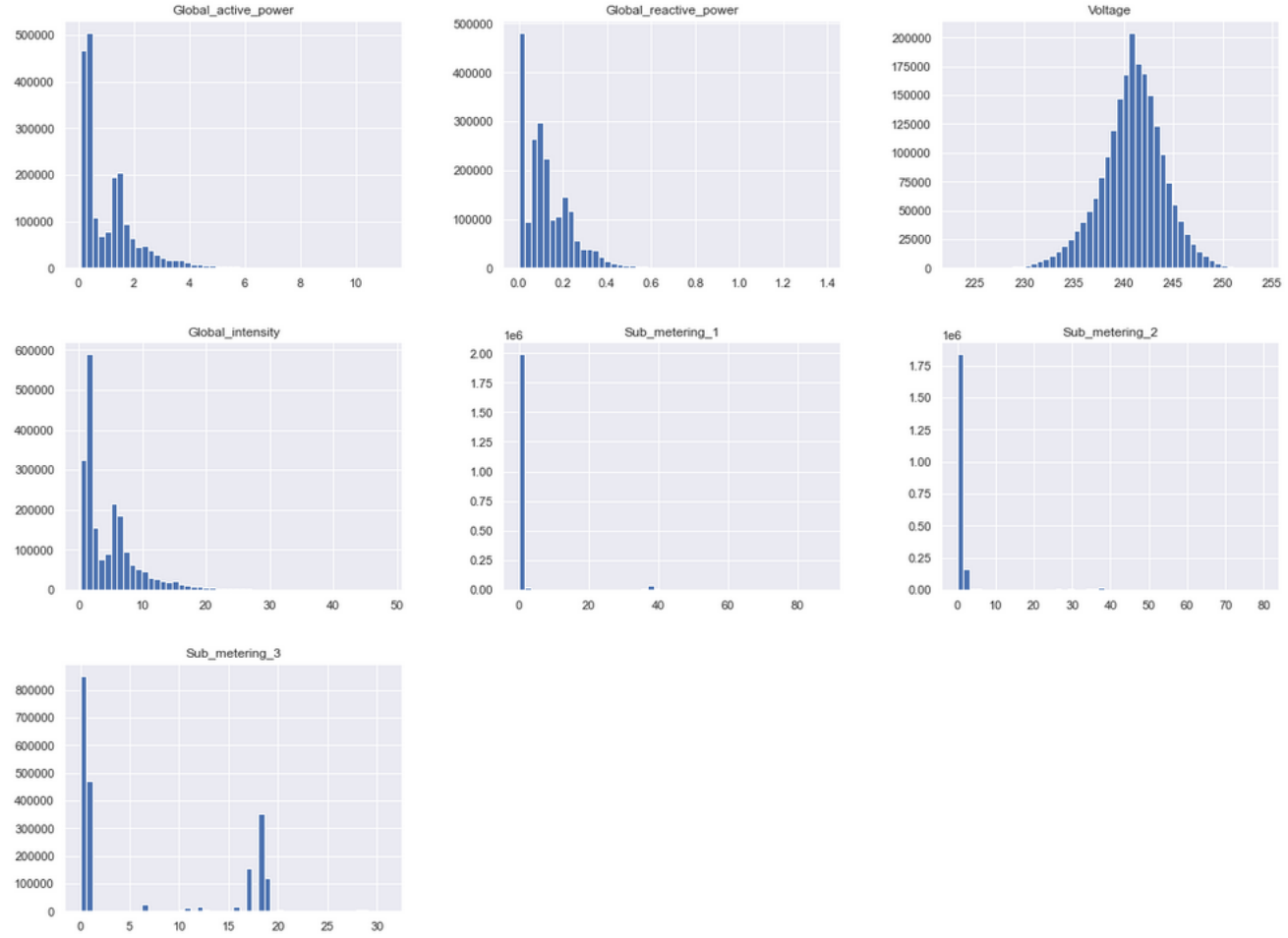
In [11]:
```
# Replace the missing value with nan
df.replace(['?', 'nan', np.nan], -1, inplace=True)


num_vars= ['Global_active_power', 'Global_reactive_power', 'Voltage',
           'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']

# Convert the columns data type
for i in num_vars:
    df[i] = pd.to_numeric(df[i])

# Use Simple Imputer for Mean imputation
imp = SimpleImputer(missing_values=-1, strategy='mean')
df[num_vars] = imp.fit_transform(df[num_vars])
```

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Date                   object
 1   Time                   object
 2   Global_active_power    float64
 3   Global_reactive_power  float64
 4   Voltage                float64
 5   Global_intensity       float64
 6   Sub_metering_1         float64
 7   Sub_metering_2         float64
 8   Sub_metering_3         float64
dtypes: float64(7), object(2)
memory usage: 142.5+ MB
```

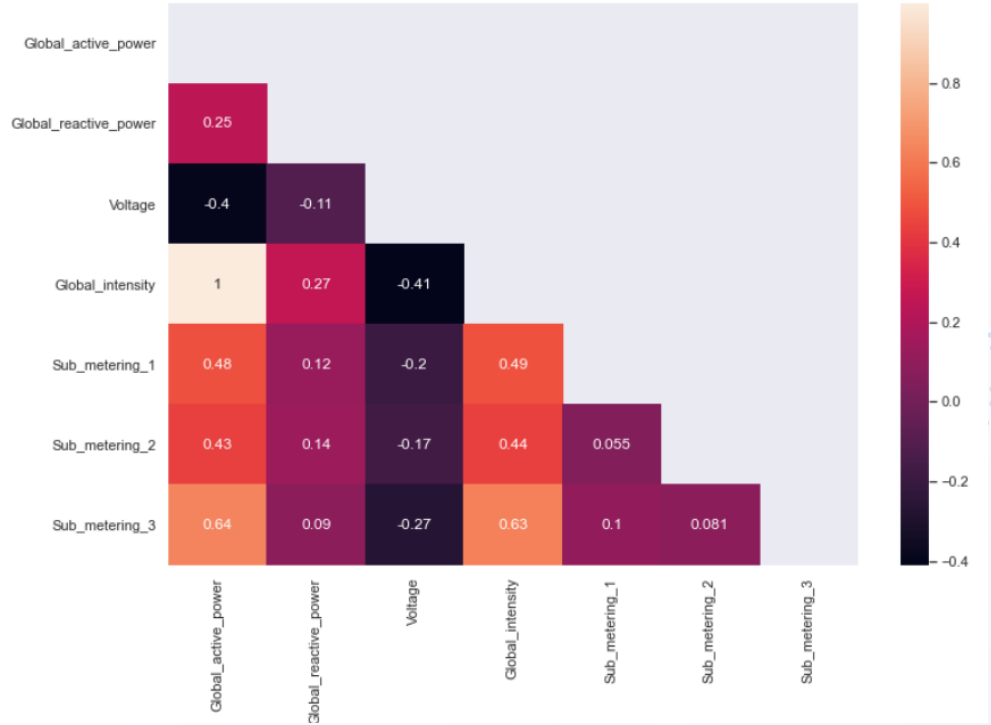
Understanding the Distribution

In [16]:
```
#Part 1 - Explore the data - Credit Cards
df.hist(bins=50, figsize=(20,15))
plt.show()
```



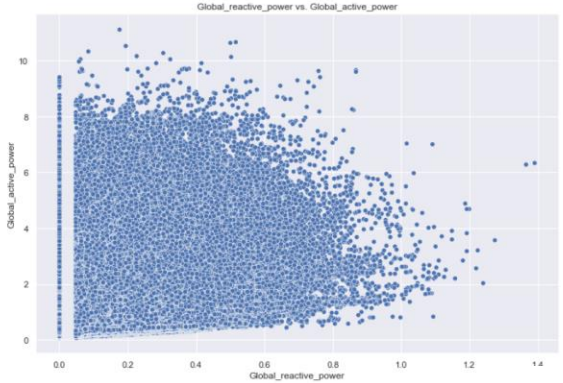
Correlation matrix

```
corr = np.corrcoef(df.corr())
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(df.corr(), annot=True, mask=mask)
plt.show()
```

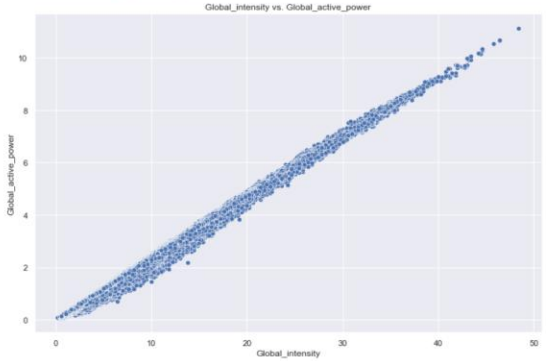


Scatter plot

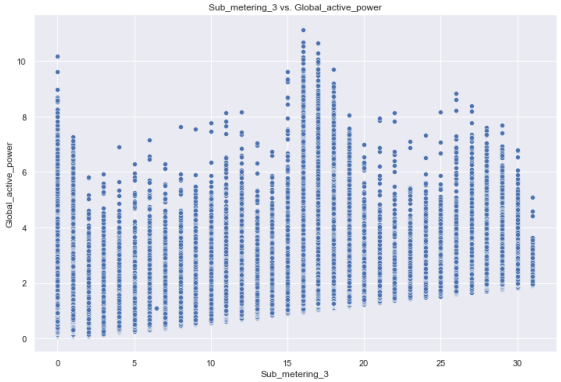
In [22]:
```
# Scatter plot
ax = sns.scatterplot(x="Global_reactive_power", y="Global_active_power", data=df)
ax.set_title("Global_reactive_power vs. Global_active_power")
ax.set_xlabel("Global_reactive_power");
```



In [23]:
```
# Scatter plot
ax = sns.scatterplot(x="Global_intensity", y="Global_active_power", data=df)
ax.set_title("Global_intensity vs. Global_active_power")
ax.set_xlabel("Global_intensity");
```



In [24]:
```
# Scatter plot
ax = sns.scatterplot(x="Sub_metering_3", y="Global_active_power", data=df)
ax.set_title("Sub_metering_3 vs. Global_active_power")
ax.set_xlabel("Sub_metering_3");
```



Training the baseline model

Baseline model with Global intensity

Training the baseline model

In [49]:
```
# Feature selection
X = df.drop(['Date', 'Time','Global_active_power'], axis=1)

# Target variable selection
y = df[['Global_active_power']]
```

In [50]:
```
# Data splitting into train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

# Fitting the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and Evaluate
model.score(X_train, y_train)
model.score(X_test, y_test)


y_pred =model.predict(X_test)
score = r2_score(y_test,y_pred)
print("Prediction score for baseline model", score)

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Prediction score for baseline model 0.9985296648908296
Mean Squared Error: 0.0016226422293213864
```

Cross validation

```
[ ]  # Using K-Fold Cross Validation methodology
     print(cross_val_score(model, X_train, y_train, cv=5))
```

```
[0.99853879 0.9985365  0.99852595 0.99849477 0.99853162]
```

Baseline model without Global intensity

In [51]:
```
# Feature selection
X = df.drop(['Date', 'Time','Global_active_power','Global_intensity'], axis=1)

# Target variable selection
y = df[['Global_active_power']]
```

In [52]:
```
# Data splitting into train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

# Fitting the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and Evaluate
model.score(X_train, y_train)
model.score(X_test, y_test)


y_pred =model.predict(X_test)
score = r2_score(y_test,y_pred)
print("Prediction score for baseline model without Global_intensity", score)

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Prediction score for baseline model without Global_intensity 0.7410220932767613
Mean Squared Error: 0.28577713077296263
```

Cross validation

```
[ ]  # Using K-Fold Cross Validation methodology
     print(cross_val_score(model, X_train, y_train, cv=5))
```

```
[0.73844814 0.7401824  0.73881536 0.73886254 0.73997534]
```

Scaling

Using Standard scaler

In [67]:
```
# Scale the data using StandardScaler

scale = StandardScaler().fit(X_train)
X_train_stand = scale.transform(X_train)
X_test_stand = scale.transform(X_test)

y_train_stand  = StandardScaler().fit_transform(y_train).flatten()
y_test_stand = StandardScaler().fit_transform(y_test).flatten()
```

In [68]:
```
# Train the regression model

scaled_model = LinearRegression()
scaled_model.fit(X_train_stand, y_train_stand)

scaled_model.score(X_train_stand, y_train_stand)
scaled_model.score(X_test_stand, y_test_stand)

y_pred =scaled_model.predict(X_test_stand)
score2 = r2_score(y_test_stand,y_pred)
print("Prediction score after scaling is",score2)

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test_stand, y_pred)
print("Mean Squared Error:", mse)
```

```
Prediction score after scaling is 0.7388224077190668
Mean Squared Error: 0.2611775922809331
```

Using Normalizer

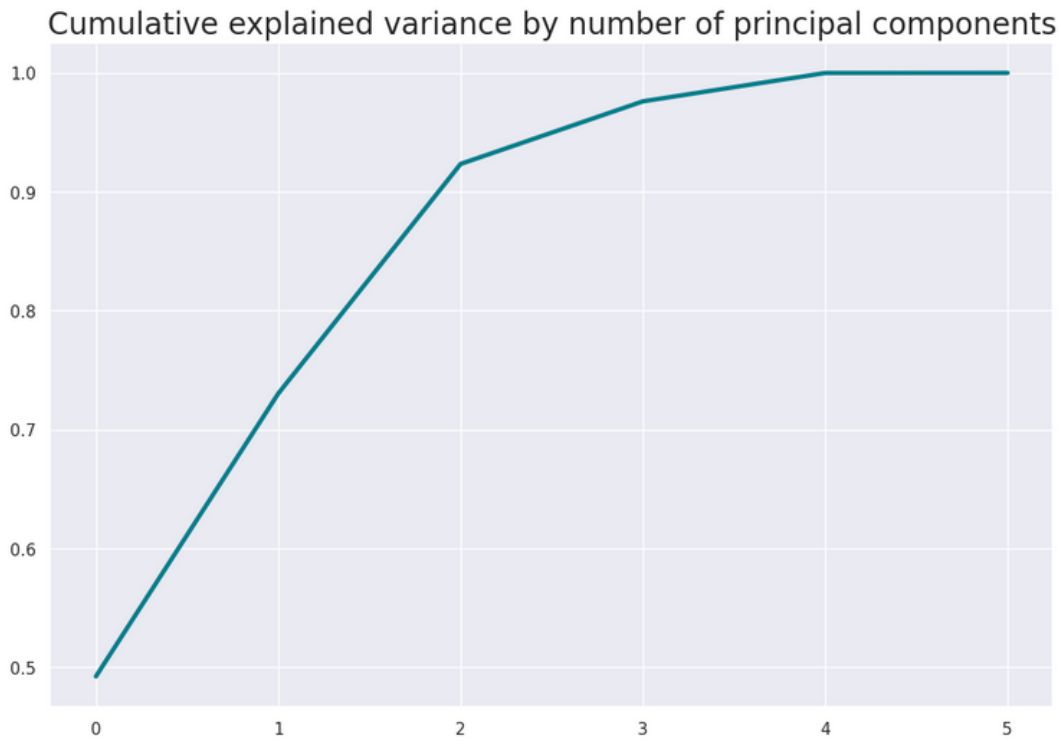
In [69]:
```
# Using Normalizer

scale = Normalizer().fit(X_train)
X_train_stand = scale.transform(X_train)
X_test_stand = scale.transform(X_test)

y_train_stand  = Normalizer().fit_transform(y_train).flatten()
y_test_stand = Normalizer().fit_transform(y_test).flatten()
```

In [70]:
```
# Train the regression model

scaled_model = LinearRegression()
scaled_model.fit(X_train_stand, y_train_stand)

scaled_model.score(X_train_stand, y_train_stand)
scaled_model.score(X_test_stand, y_test_stand)

y_pred =scaled_model.predict(X_test_stand)
score2 = r2_score(y_test_stand,y_pred)
print("Prediction score after scaling is",score2)

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test_stand, y_pred)
print("Mean Squared Error:", mse)
```

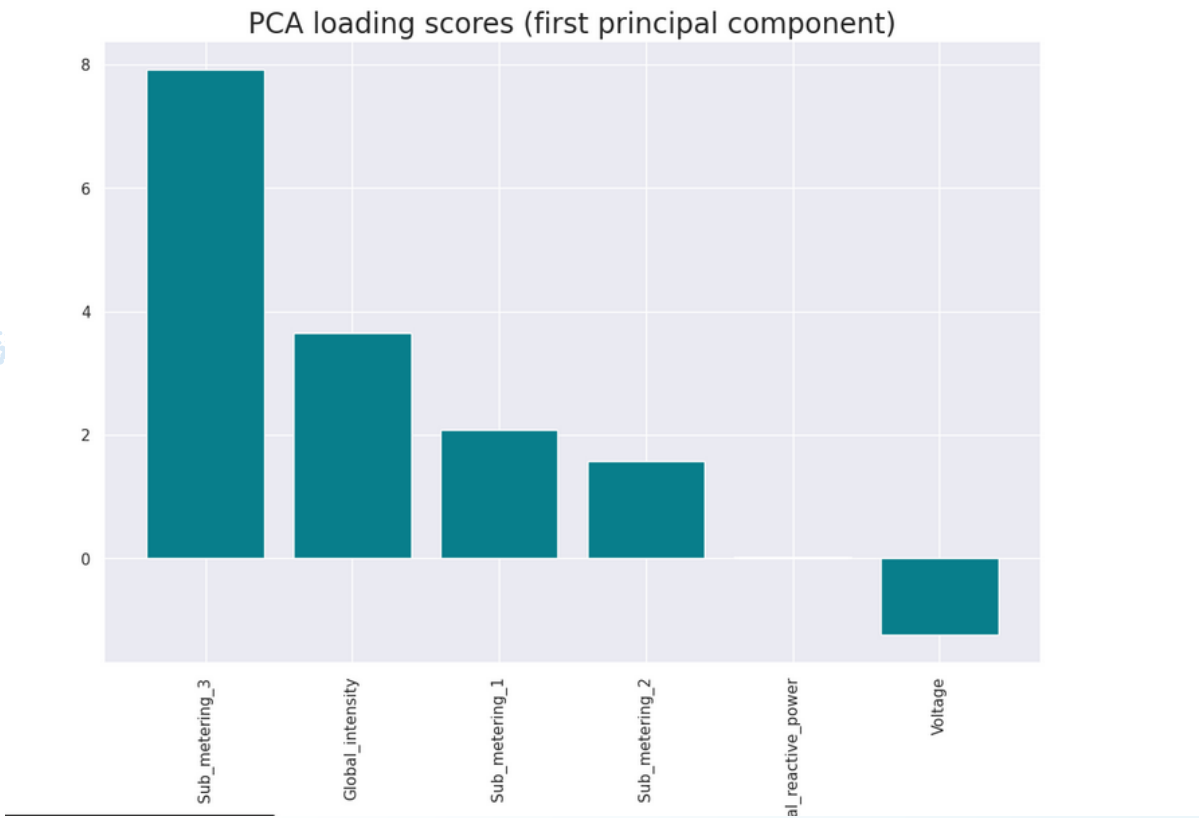
```
Prediction score after scaling is 1.0
Mean Squared Error: 0.0
```

Principal Component Analysis

```python
# Use PCA to find the N
from sklearn.decomposition import PCA

pca = PCA().fit(X_train)



# Plot the N and explained variance ratio

plt.plot(pca.explained_variance_ratio_.cumsum(), lw=3, color='#087E8B')
plt.title('Cumulative explained variance by number of principal components', size=20)
plt.show()
```



```
[105] # Visualize the correlations between all of the input features and the first principal components

      pc1_loadings = loadings.sort_values(by='PC1', ascending=False)[['PC1']]
      pc1_loadings = pc1_loadings.reset_index()
      pc1_loadings.columns = ['Attribute', 'CorrelationWithPC1']

      plt.bar(x=pc1_loadings['Attribute'], height=pc1_loadings['CorrelationWithPC1'], color='#087E8B')
      plt.title('PCA loading scores (first principal component)', size=20)
      plt.xticks(rotation='vertical')
      plt.show()
```



```python
# data splitting into train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

# Initialize the PCA
pca = PCA(n_components=2)

# fit and transform data
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Fitting the model
model_3 = LinearRegression()
model_3.fit(X_train_pca, y_train)


# Predict
y_pred =model_3.predict(X_test_pca)

# Evaluate the model based on R2 score
from sklearn.metrics import r2_score
score = r2_score(y_test,y_pred)
print("Prediction score after using PCA is",score)
```

```
Prediction score after using PCA is 0.828904924448761
```

Model Building and Evaluations

Multiple Linear Regression

1. Multiple Linear Regression

1s

```
class linmodel():
    def __init__(self, df, target):
        self.df = df
        self.target = target

    def pre_processing(self):
        cat = ['Date', 'Time', 'power_consumption']
        X = self.df.drop(cat+[self.target], axis=1).values
        Y = self.df[self.target].values

        self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(X, Y,
                                                               test_size = 0.2,
                                                               random_state = 2)
        return self

    def fit_pred_acc(self):
        reg = LinearRegression()
        reg.fit(self.X_train, self.Y_train)
        pred = reg.predict(self.X_test)
        mae = round(skm.mean_absolute_error(self.Y_test, pred), 2)
        rmse = round(skm.mean_squared_error(self.Y_test, pred, squared=False), 2)
        r2_score = round(skm.r2_score(self.Y_test, pred), 4)
        ev = round(skm.explained_variance_score(self.Y_test, pred), 4)

        return [mae, rmse, r2_score, ev]

lin = linmodel(df, 'Global_active_power')
lin = lin.pre_processing()
models["Mult. Reg"] = lin.fit_pred_acc()
```

Ridge

2. Shrinkage Technique: Ridge

0s

```
class rdgmodel():
    def __init__(self, df, target):
        self.df = df
        self.target = target

    def pre_processing(self):
        cat = ['Date', 'Time', 'power_consumption']
        X = self.df.drop(cat+[self.target], axis=1).values
        Y = self.df[self.target].values

        self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(X, Y,
                                                               test_size = 0.2,
                                                               random_state = 2)
        return self

    def fit_pred_acc(self):
        reg = Ridge(alpha=0.0001)
        reg.fit(self.X_train, self.Y_train)
        pred = reg.predict(self.X_test)
        mae = round(skm.mean_absolute_error(self.Y_test, pred), 2)
        rmse = round(skm.mean_squared_error(self.Y_test, pred, squared=False), 2)
        r2_score = round(skm.r2_score(self.Y_test, pred), 4)
        ev = round(skm.explained_variance_score(self.Y_test, pred), 4)

        return [mae, rmse, r2_score, ev]

rdg = rdgmodel(df, 'Global_active_power')
rdg = rdg.pre_processing()
models["Ridge Reg"] = rdg.fit_pred_acc()
```

LASSO

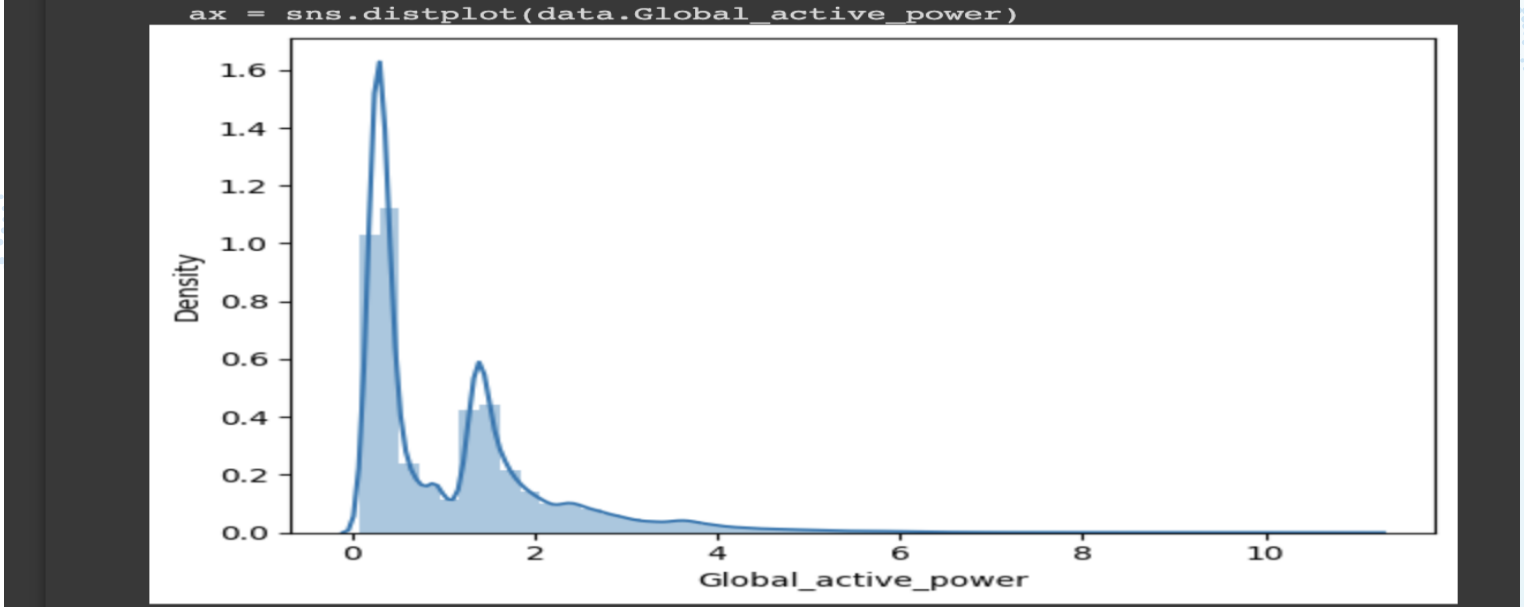
3. Shrinkage Technique: Lasso

```
class lasmodel():
    def __init__(self, df, target):
        self.df = df
        self.target = target

    def pre_processing(self):
        cat = ['Date', 'Time', 'power_consumption']
        X = self.df.drop(cat+[self.target], axis=1).values
        Y = self.df[self.target].values

        self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(X, Y,
                                                               test_size = 0.3,
                                                               random_state = 72)
        return self

    def fit_pred_acc(self):
        reg = Lasso()
        reg.fit(self.X_train, self.Y_train)
        pred = reg.predict(self.X_test)
        mae = round(skm.mean_absolute_error(self.Y_test, pred), 2)
        rmse = round(skm.mean_squared_error(self.Y_test, pred, squared=False), 2)
        r2_score = round(skm.r2_score(self.Y_test, pred), 4)
        ev = round(skm.explained_variance_score(self.Y_test, pred), 4)

        return [mae, rmse, r2_score, ev]

las = lasmodel(df, 'Global_active_power')
las = las.pre_processing()
models["Lasso Reg"] = las.fit_pred_acc()
```

XG Boost Approach

Identifying the pattern

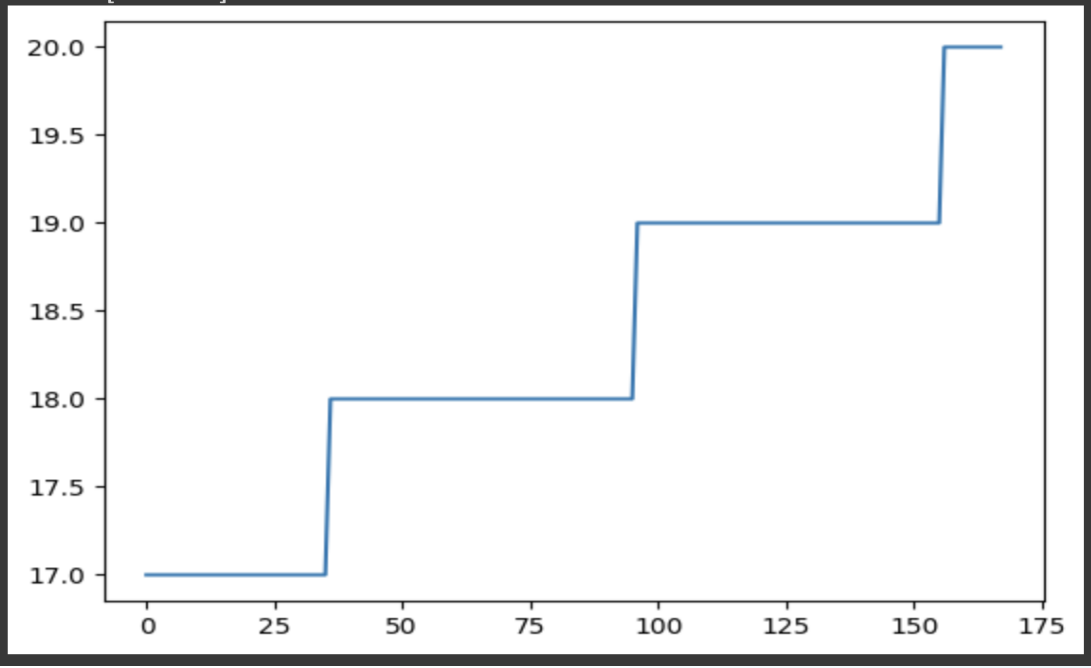
```
#plotting the graph to check the cyclical pattern of the target variable

import seaborn as sns
print(data.Global_active_power.describe())
ax = sns.distplot(data.Global_active_power)
```



Analyzing the pattern

```
data['hour'] = data.dt.dt.hour
sample = data[:168] # roughly the first week of the data
ax = sample['hour'].plot()
```

```
data['hour'] = data.dt.dt.hour
```



Encoding and Lagging the Data

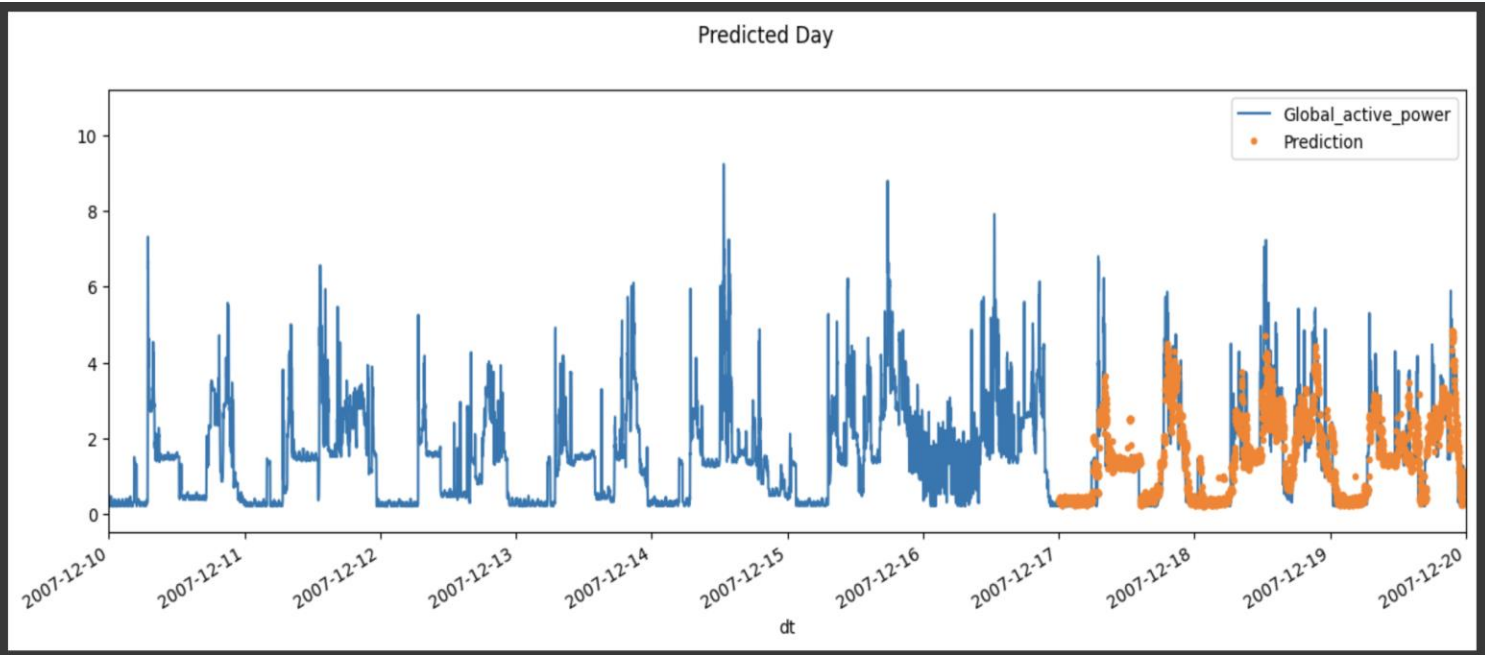
Encoding Cyclical Features

0s

```
def encode(data, col, max_val):
    data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
    data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
    data.drop(col,axis=1,inplace = True)
    return data
```

```
def get_lag(data, col, lagtime):
    for i in range(1,lagtime+1):
        if len(pd.Series(col)) == 1:
            data[col+"_lag"+str(i)] = data[col].shift(i*15)
        else:
            for col_j in col:
                data[col_j+"_lag"+str(i)] = data[col_j].shift(i*15)
    return data
```

Prediction




```
[34] np.sqrt(mean_squared_error(df_test['Prediction'], df_test['Global_active_power']))

     0.687663864847101
```

Result Comparison

Model Type	RMSE
Baseline Model With Global Intensity	0.001
Baseline Model Without Global	0.285
Multiple Linear Regression	0.04
Ridge Regression	0.04
LASSO Regression	0.23
XG Boost	0.687

References

- https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption
- https://circuitglobe.com/difference-between-active-and-reactive-power.html
- https://solar-energy.technology/electricity/electric-current/current-intensity
- https://www.kaggle.com/datasets/uciml/electric-power-consumption-data-set

References

- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- https://towardsdatascience.com/mastering-linear-regression-the-definitive-guide-for-aspiring-data-scientists-7abd37fcb9ed
- https://towardsdatascience.com/cross-validation-430d9a5fee22
- https://scikit-learn.org/stable/modules/cross_validation.html
- https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

THANK YOU