In [1]:

```
# Data manipulation
# ==============================================================================
import numpy as np
import pandas as pd

# Plots
# ==============================================================================
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
plt.style.use('fivethirtyeight')

# Modelling and Forecasting
# ==============================================================================
from sklearn.linear_model import Ridge
#from lightgbm import LGBMRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
from skforecast.ForecasterAutoreg import ForecasterAutoreg
from skforecast.ForecasterAutoregMultiOutput import ForecasterAutoregMultiOutput
from skforecast.model_selection import grid_search_forecaster
from skforecast.model_selection import backtesting_forecaster
# Warnings configuration
# ==============================================================================
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#Importing the dataset
df = pd.read_csv('dataset.csv')

#Checking the dataset first 10 rows
df.head(10)
```

Out[2]:

	Unnamed: 0	demand [MW]	solar_actual [MW]	solar_forecast [MW]	solar_inferred_capacity [MW]	wind_actual [MW]	w
0	2017-01-01 00:00:00+01:00	76345.25	0.00	NaN	5756.44	597.50	
1	2017-01-01 01:00:00+01:00	75437.00	0.00	NaN	5756.44	597.50	
2	2017-01-01 02:00:00+01:00	73368.25	0.00	NaN	5756.44	635.25	
3	2017-01-01 03:00:00+01:00	72116.00	0.00	NaN	5756.44	628.50	
4	2017-01-01 04:00:00+01:00	68593.75	0.00	NaN	5756.44	608.50	
5	2017-01-01 05:00:00+01:00	65865.75	0.00	NaN	5756.44	634.25	
6	2017-01-01 06:00:00+01:00	64856.50	0.00	NaN	5756.44	662.75	
7	2017-01-01 07:00:00+01:00	64406.00	0.00	NaN	5756.44	681.00	
8	2017-01-01 08:00:00+01:00	64462.25	13.75	NaN	5756.44	742.50	
9	2017-01-01 09:00:00+01:00	64828.75	84.00	NaN	5756.44	792.25	

In [3]:

```
#Checking the dataset columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45432 entries, 0 to 45431
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Unnamed: 0                   45432 non-null  object 
 1   demand [MW]                  45429 non-null  float64
 2   solar_actual [MW]            45413 non-null  float64
 3   solar_forecast [MW]          45210 non-null  float64
 4   solar_inferred_capacity [MW] 45432 non-null  float64
 5   wind_actual [MW]             45413 non-null  float64
 6   wind_inferred_capacity [MW]  45432 non-null  float64
 7   albedo [%]                   45415 non-null  float64
 8   cloud_cover [%]              45416 non-null  float64
 9   frozen_precipitation [%]     45422 non-null  float64
 10  pressure [Pa]                45421 non-null  float64
 11  radiation [W/m2]             45416 non-null  float64
 12  air_tmp [Kelvin]             45422 non-null  float64
 13  ground_tmp [Kelvin]          45422 non-null  float64
 14  apparent_tmp [Kelvin]        45422 non-null  float64
 15  wind_direction [angle]       45421 non-null  float64
 16  wind_speed [m/s]             45421 non-null  float64
dtypes: float64(16), object(1)
memory usage: 5.9+ MB
```

In [4]:

```
#Rename Columns
df.rename(columns = {'Unnamed: 0':'Date'}, inplace = True)
df.rename(columns = {'demand [MW]':'Y'}, inplace = True)
```

In [5]:

```
#Check duplicated
df[df.duplicated(keep=False)]
```

Out[5]:

Date	Y	solar_actual [MW]	solar_forecast [MW]	solar_inferred_capacity [MW]	wind_actual [MW]	wind_inferred_capa [M

In [6]:

```
#Drop NA
df = df.dropna(axis=0)
```

In [7]:

```
#Checking the dataset size after dropping NA
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45202 entries, 192 to 45431
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Date                         45202 non-null  object 
 1   Y                            45202 non-null  float64
 2   solar_actual [MW]            45202 non-null  float64
 3   solar_forecast [MW]          45202 non-null  float64
 4   solar_inferred_capacity [MW] 45202 non-null  float64
 5   wind_actual [MW]             45202 non-null  float64
 6   wind_inferred_capacity [MW]  45202 non-null  float64
 7   albedo [%]                   45202 non-null  float64
 8   cloud_cover [%]              45202 non-null  float64
 9   frozen_precipitation [%]     45202 non-null  float64
 10  pressure [Pa]                45202 non-null  float64
 11  radiation [W/m2]             45202 non-null  float64
 12  air_tmp [Kelvin]             45202 non-null  float64
 13  ground_tmp [Kelvin]          45202 non-null  float64
 14  apparent_tmp [Kelvin]        45202 non-null  float64
 15  wind_direction [angle]       45202 non-null  float64
 16  wind_speed [m/s]             45202 non-null  float64
dtypes: float64(16), object(1)
memory usage: 6.2+ MB
```

In [8]:

```
#Format datetime
df.Date=pd.to_datetime(df.Date,utc=True)

# Sorting data in ascending order by the date
df = df.sort_values(by='Date')

#Set Date as index
df.set_index('Date', inplace=True)
```

In [9]:

```
df.head(10)
```

Out[9]:

	Y	solar_actual [MW]	solar_forecast [MW]	solar_inferred_capacity [MW]	wind_actual [MW]	winc
Date						
2017-01-08 23:00:00+00:00	72921.75	0.00	0.55	5756.44	1151.00	
2017-01-09 00:00:00+00:00	70956.00	0.00	0.55	5756.44	1103.75	
2017-01-09 01:00:00+00:00	68422.50	0.00	0.55	5756.44	1111.00	
2017-01-09 02:00:00+00:00	67520.50	0.00	0.06	5756.44	1165.00	
2017-01-09 03:00:00+00:00	64729.25	0.00	0.06	5756.44	1210.75	
2017-01-09 04:00:00+00:00	63864.50	0.00	0.06	5756.44	1185.25	
2017-01-09 05:00:00+00:00	66086.75	0.00	0.55	5756.44	1168.00	
2017-01-09 06:00:00+00:00	71651.00	0.00	0.55	5756.44	1241.00	
2017-01-09 07:00:00+00:00	78221.25	27.75	17.27	5756.44	1320.00	
2017-01-09 08:00:00+00:00	81002.00	108.25	105.75	5756.44	1389.50	

In [10]:

```
df.describe()
```

Out[10]:

	Y	solar_actual [MW]	solar_forecast [MW]	solar_inferred_capacity [MW]	wind_actual [MW]	wind_i
count	45202.000000	45202.000000	45202.000000	45202.000000	45202.000000	
mean	53426.420418	1289.892801	1278.937576	8266.976570	3624.040546	
std	11733.759596	1784.932856	1761.382969	1612.296701	2710.513500	
min	29415.000000	0.000000	0.000000	5756.440000	391.000000	
25%	44446.812500	0.000000	0.000000	6864.480000	1590.062500	
50%	51707.625000	176.000000	154.575000	7992.890000	2722.625000	
75%	61565.375000	2272.437500	2332.165000	9595.960000	4936.250000	
max	94587.250000	8511.750000	7900.170000	11244.010000	14475.750000	

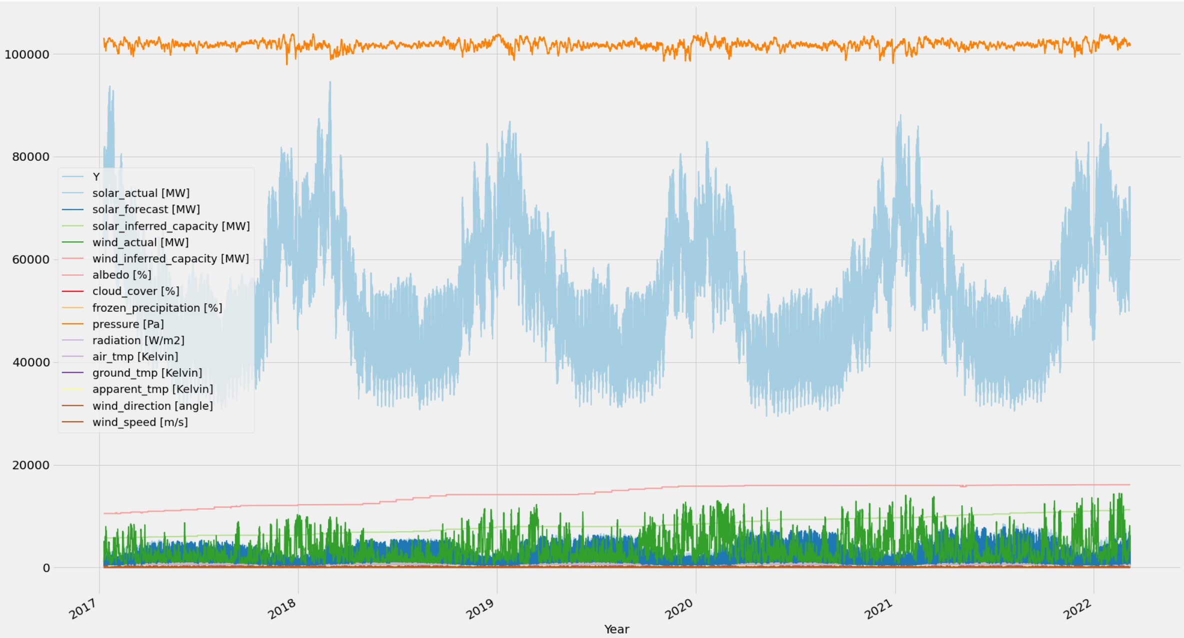
In [11]:

```
#Checking the dataset first 10 rows
df.head(10)
```

2017-01-09 04:00:00+00:00	63864.50	0.00	0.06	5756.44	1185.25	1(
2017-01-09 05:00:00+00:00	66086.75	0.00	0.55	5756.44	1168.00	1(
2017-01-09 06:00:00+00:00	71651.00	0.00	0.55	5756.44	1241.00	1(
2017-01-09 07:00:00+00:00	78221.25	27.75	17.27	5756.44	1320.00	1(
2017-01-09 08:00:00+00:00	81002.00	108.25	105.75	5756.44	1389.50	1(

In [12]:

```
# setting the graph size globally
plt.rcParams['figure.figsize'] = (30,20)


# Visulize 5 years data

df.plot(colormap='Paired', linewidth=2, fontsize=20)
plt.xlabel('Year', fontsize=20)
plt.ylabel('', fontsize=20)
plt.legend(fontsize=18)
plt.show()
```



In [13]:

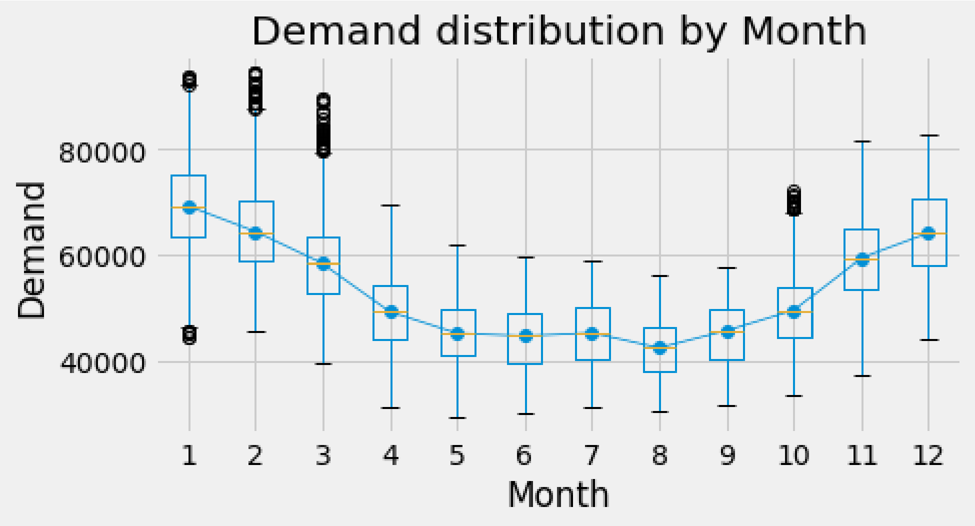
```
#Add new columns
df['Month'] = df.index.month
df['Weekday'] = df.index.weekday + 1 #in pandas, the day of the week with Monday=0, Sunday=
df['Hour'] = df.index.hour
```

In [14]:

```
# setting the graph size globally
plt.rcParams['figure.figsize'] = (30,20)
```

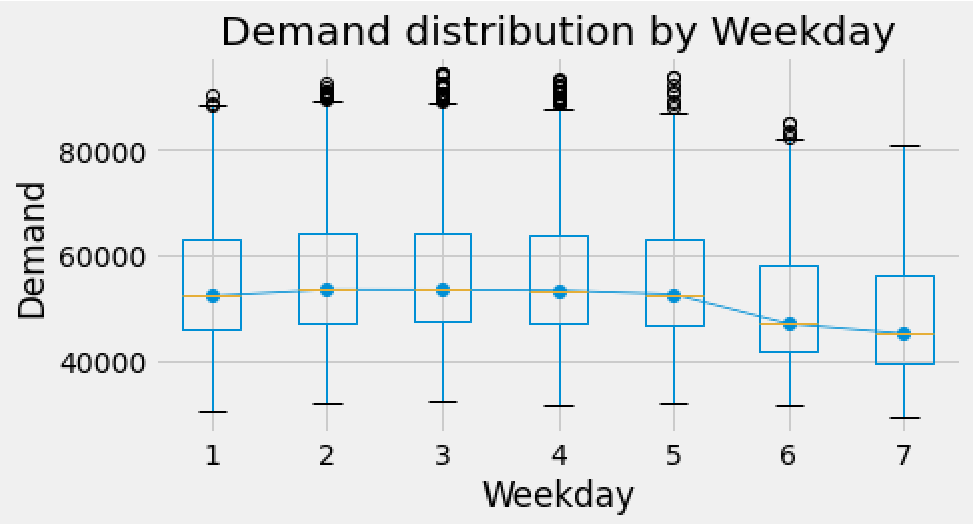
In [15]:

```
# Boxplot for annual seasonality
# ==============================================================================
fig, ax = plt.subplots(figsize=(7, 3.5))
df.boxplot(column='Y', by='Month', ax=ax,)
df.groupby('Month')['Y'].median().plot(style='o-', linewidth=0.8, ax=ax)
ax.set_ylabel('Demand')
ax.set_title('Demand distribution by Month')
fig.suptitle('');
```



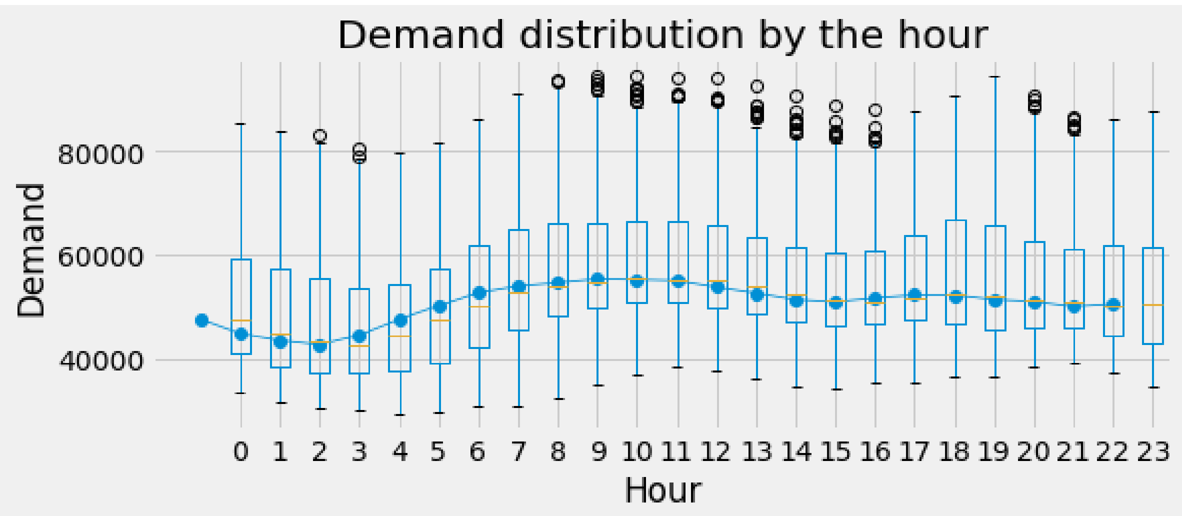
In [16]:

```
# Boxplot for weekly seasonality
# ==============================================================================
fig, ax = plt.subplots(figsize=(7, 3.5))
df.boxplot(column='Y', by='Weekday', ax=ax)
df.groupby('Weekday')['Y'].median().plot(style='o-', linewidth=0.8, ax=ax)
ax.set_ylabel('Demand')
ax.set_title('Demand distribution by Weekday')
fig.suptitle('');
```



In [17]:

```
# Boxplot for daily seasonality
# ==============================================================================
fig, ax = plt.subplots(figsize=(9, 3.5))
df.boxplot(column='Y', by='Hour', ax=ax)
df.groupby('Hour')['Y'].median().plot(style='o-', linewidth=0.8, ax=ax)
ax.set_ylabel('Demand')
ax.set_title('Demand distribution by the hour')
fig.suptitle('');
plt.show()
```

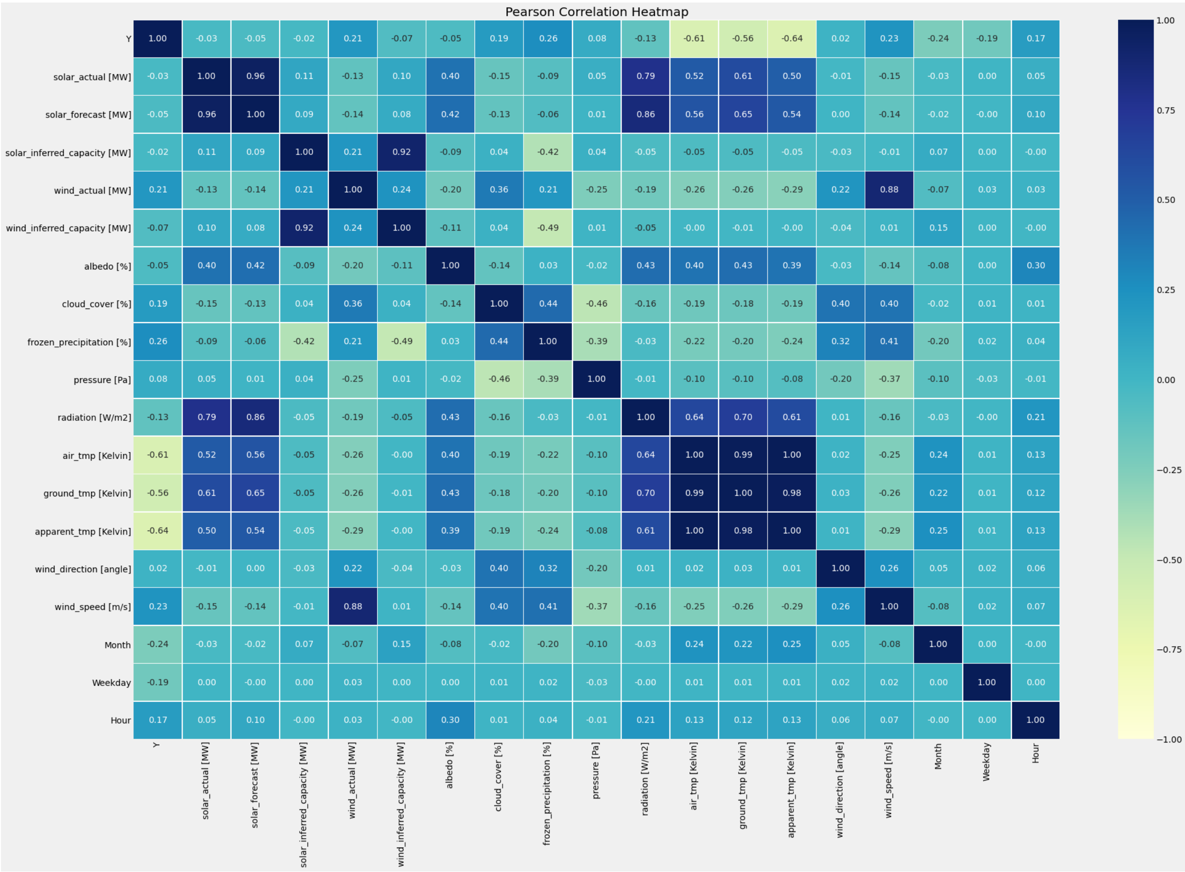


In [18]:

```
# Pearson Correlation Heatmap
corr_matrix = df.corr(method="pearson")
sns.heatmap(corr_matrix, vmin=-1., vmax=1., annot=True, fmt='.2f', cmap="YlGnBu", cbar=True
plt.title("Pearson Correlation Heatmap")
```

Out[18]:

```
Text(0.5, 1.0, 'Pearson Correlation Heatmap')
```



In [19]:

```
# MLR import
from sklearn import model_selection, preprocessing, feature_selection, ensemble, linear_mod
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [20]:

```
#Identifiy Y
x = "Y"
```

In [21]:

```
## split data
#df_train, df_test = model_selection.train_test_split(df,
#                      test_size=0.2)
df_train, df_test= np.split(df, [int(.8 *len(df))])
```

In [22]:

```
## print info
print("X_train shape:", df_train.drop("Y",axis=1).shape, "| X_test shape:", df_test.drop("Y
print("y_train mean:", round(np.mean(df_train["Y"]),2), "| y_test mean:", round(np.mean(df_
print(df_train.shape[1], "features:", df_train.drop("Y",axis=1).columns.to_list())
```

```
X_train shape: (36161, 18) | X_test shape: (9041, 18)
y_train mean: 53475.21 | y_test mean: 53231.27
19 features: ['solar_actual [MW]', 'solar_forecast [MW]', 'solar_inferred_ca
pacity [MW]', 'wind_actual [MW]', 'wind_inferred_capacity [MW]', 'albedo
[%]', 'cloud_cover [%]', 'frozen_precipitation [%]', 'pressure [Pa]', 'radia
tion [W/m2]', 'air_tmp [Kelvin]', 'ground_tmp [Kelvin]', 'apparent_tmp [Kelv
in]', 'wind_direction [angle]', 'wind_speed [m/s]', 'Month', 'Weekday', 'Hou
r']
```

In [23]:

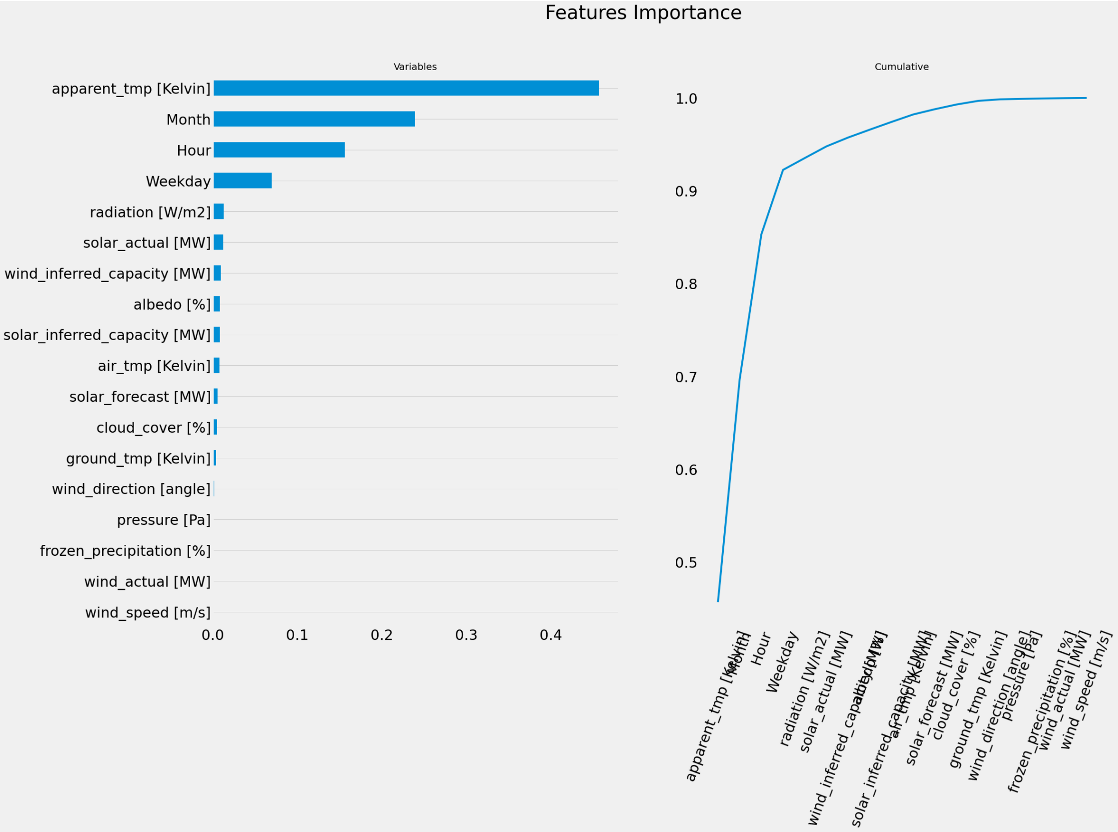
```
##### Ensemble methods   gradient boosting algorithm
X = df_train.drop("Y", axis=1).values

# ## scale X
scalerX = preprocessing.RobustScaler(quantile_range=(20.0, 80.0))
X = scalerX.fit_transform(df_train.drop("Y", axis=1))
df_scaled= pd.DataFrame(X, columns=df_train.drop("Y",
                         axis=1).columns, index=df_train.index)
# ## scale Y
y="Y"
scalerY = preprocessing.RobustScaler(quantile_range=(20.0, 80.0))
df_scaled[y] = scalerY.fit_transform(
                     df_train[y].values.reshape(-1,1))
df_scaled.head()
```

Out[23]:

	solar_actual [MW]	solar_forecast [MW]	solar_inferred_capacity [MW]	wind_actual [MW]	wind_inferred_(
Date					
2017-01-08 23:00:00+00:00	-0.057092	-0.049611	-0.811021	-0.359696	-(
2017-01-09 00:00:00+00:00	-0.057092	-0.049611	-0.811021	-0.371011	-(
2017-01-09 01:00:00+00:00	-0.057092	-0.049611	-0.811021	-0.369275	-(
2017-01-09 02:00:00+00:00	-0.057092	-0.049796	-0.811021	-0.356343	-(
2017-01-09 03:00:00+00:00	-0.057092	-0.049796	-0.811021	-0.345387	-(

In [24]:

```
## Use gradient boosting algorithm calculate importance
## define y value
y = df_train["Y"].values
feature_names = df_train.drop("Y", axis=1).columns.tolist()

## call model
model = ensemble.GradientBoostingRegressor()

## Importance calculation
model.fit(X,y)
importances = model.feature_importances_
## Put in a pandas df
df_importances = pd.DataFrame({"IMPORTANCE":importances,
            "VARIABLE":feature_names}).sort_values("IMPORTANCE",
            ascending=False)
df_importances['cumsum'] = df_importances['IMPORTANCE'].cumsum(axis=0)
df_importances = df_importances.set_index("VARIABLE")

## Plot
fig, ax = plt.subplots(nrows=1, ncols=2, sharex=False, sharey=False)
fig.suptitle("Features Importance", fontsize=40)
ax[0].title.set_text('Variables')
df_importances[["IMPORTANCE"]].sort_values(by="IMPORTANCE").plot(
                kind="barh", legend=False, ax=ax[0],fontsize=30).grid(axis="x")
ax[0].set(ylabel="")
ax[1].title.set_text('Cumulative')
df_importances[["cumsum"]].plot(kind="line", linewidth=4,
                                 legend=False, ax=ax[1],fontsize=30)
ax[1].set(xlabel="", xticks=np.arange(len(df_importances)),
          xticklabels=df_importances.index)
plt.xticks(rotation=70)
plt.grid(axis='both')
plt.show()
```


Features Importance
Variables
apparent_tmp [Kelvin]
Month
Hour
Weekday
radiation [W/m2]
solar_actual [MW]
wind_inferred_capacity [MW]
albedo [%]
solar_inferred_capacity [MW]
air_tmp [Kelvin]
solar_forecast [MW]
cloud_cover [%]
ground_tmp [Kelvin]
wind_direction [angle]
pressure [Pa]
frozen_precipitation [%]
wind_actual [MW]
wind_speed [m/s]
0.0
0.1
0.2
0.3
0.4
Cumulative
1.0
0.9
0.8
0.7
0.6
0.5

In [25]:

```
#-Use MLR train and test data
X_names = ["apparent_tmp [Kelvin]", "Month", "Hour", "Weekday", "radiation [W/m2]"
           ,"air_tmp [Kelvin]", "solar_actual [MW]", "albedo [%]"]
X_train = df_train[X_names].values
y_train = df_train["Y"].values

X_test = df_test[X_names].values
y_test = df_test["Y"].values

## call model
model = linear_model.LinearRegression()

## K fold validation
scores = []
cv = model_selection.KFold(n_splits=5, shuffle=True)
fig = plt.figure()
i = 1
for train, test in cv.split(X_train, y_train):
    prediction = model.fit(X_train[train],
                 y_train[train]).predict(X_train[test])
    true = y_train[test]
    score = metrics.r2_score(true, prediction)
    scores.append(score)
    plt.scatter(prediction, true, lw=2, alpha=0.3,
                label='Fold %d (R2 = %0.2f)' % (i,score))
    i = i+1
plt.plot([min(y_train),max(y_train)], [min(y_train),max(y_train)],
         linestyle='--', lw=2, color='black')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('K-Fold Validation')
plt.legend()
plt.show()
```

In [26]:

```
## train
model.fit(X_train, y_train)
## test
predicted = model.predict(X_test)
```

In [57]:

```
## Kpi
print('MLR')
print('------------------------------')
print("R2 (explained variance):", round(metrics.r2_score(y_test, predicted), 3))
print("Mean Absolute Perc Error (Σ(|y-pred|/y)/n):", round(np.mean(np.abs((y_test-predicted
print("Mean Absolute Error (Σ|y-pred|/n):", "{:,.0f}".format(metrics.mean_absolute_error(y_
print("Root Mean Squared Error (sqrt(Σ(y-pred)^2/n)):", "{:,.0f}".format(np.sqrt(metrics.me
## residuals
residuals = y_test - predicted
max_error = max(residuals) if abs(max(residuals)) > abs(min(residuals)) else min(residuals)
max_idx = list(residuals).index(max(residuals)) if abs(max(residuals)) > abs(min(residuals)
max_true, max_pred = y_test[max_idx], predicted[max_idx]
print("Max Error:", "{:,.0f}".format(max_error))
```

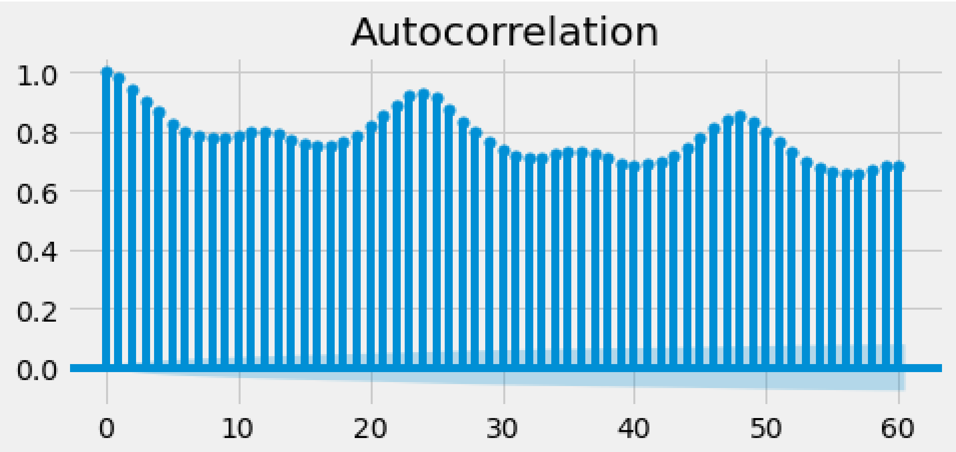
```
MLR
------------------------------
R2 (explained variance): 0.626
Mean Absolute Perc Error (Σ(|y-pred|/y)/n): 0.098
Mean Absolute Error (Σ|y-pred|/n): 5,461
Root Mean Squared Error (sqrt(Σ(y-pred)^2/n)): 6,883
Max Error: -27,682
```

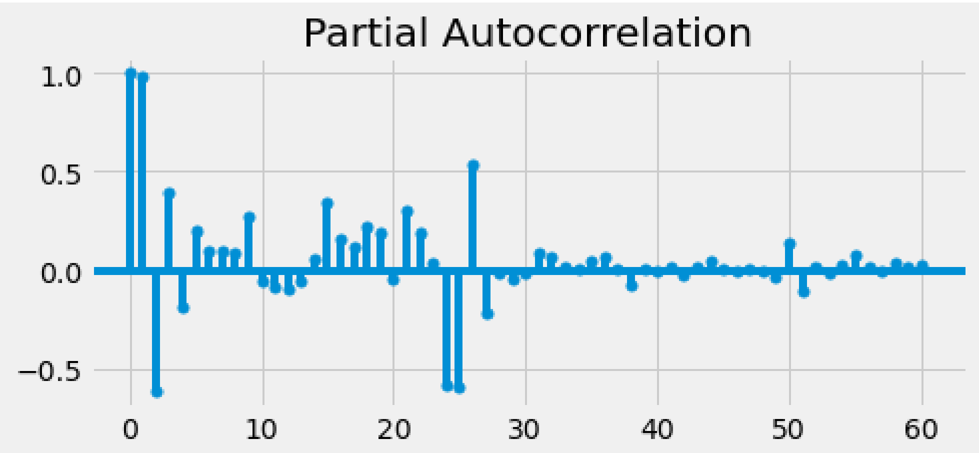
In [62]:

```
## Plot predicted vs true
fig, ax = plt.subplots(nrows=1, ncols=2)
from statsmodels.graphics.api import abline_plot
ax[0].scatter(predicted, y_test, color="black")
abline_plot(intercept=0, slope=1, color="red", ax=ax[0])
ax[0].vlines(x=max_pred, ymin=max_true, ymax=max_true-max_error, color='red', linestyle='--
ax[0].grid(True)
ax[0].set(xlabel="Predicted", ylabel="True", title="MLR Predicted vs True")
ax[0].legend()

## Plot predicted vs residuals
ax[1].scatter(predicted, residuals, color="red")
ax[1].vlines(x=max_pred, ymin=0, ymax=max_error, color='black', linestyle='--', alpha=0.7,
ax[1].grid(True)
ax[1].set(xlabel="Predicted", ylabel="Residuals", title="Predicted vs Residuals")
ax[1].hlines(y=0, xmin=np.min(predicted), xmax=np.max(predicted))
ax[1].legend()
plt.show()
# ===========================================================================
```

In [31]:

```
#---------------------------------------------------
#Autoregressive(AR)

# Autocorrelation plot
# ==============================================================================
fig, ax = plt.subplots(figsize=(7, 3))
plot_acf(df.Y, ax=ax, lags=60)
plt.show()


# Partial autocorrelation plot
# ==============================================================================
fig, ax = plt.subplots(figsize=(7, 3))
plot_pacf(df.Y, ax=ax, lags=60)
plt.show()
```





In [32]:

```
from sklearn import model_selection, preprocessing, feature_selection, ensemble, linear_mod
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [33]:

```
#Identifiy Y
x = "Y"
```

In [34]:

```
# Create and train forecaster
# ==============================================================================
forecaster = ForecasterAutoreg(
                regressor = Ridge(normalize=True),
                lags      = 24 #previous 24 hours are used as predictors
             )

forecaster.fit(y=df_train.Y)
```

In [35]:

```
# Backtest
# ==============================================================================
metric, predictions = backtesting_forecaster(
                            forecaster = forecaster,
                            y          = df.Y,
                            initial_train_size = len(df_train),
                            steps      = 24,
                            #metric     = 'mean_absolute_error',
                            metric     = 'mean_absolute_percentage_error',
                            verbose    = True
                        )
```

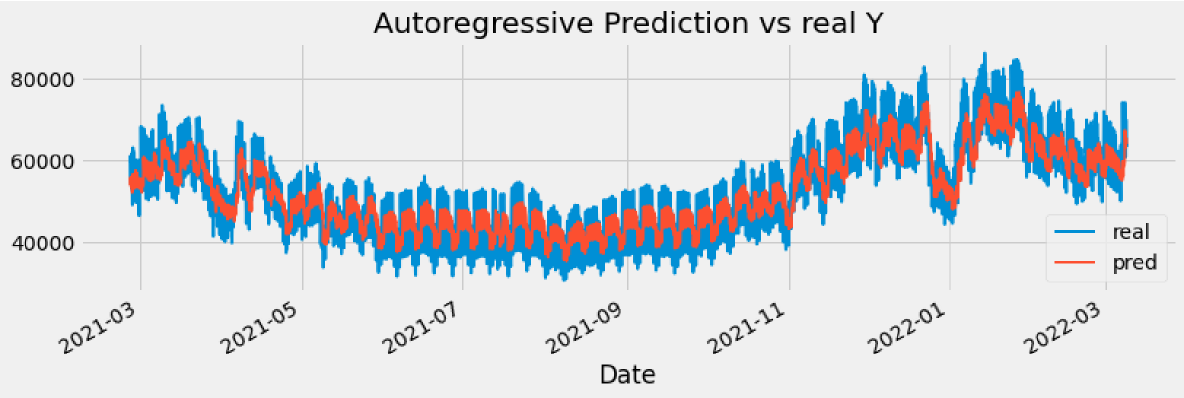
```
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-10 05:00:00+00:00 -- 2021-05-11 04:00:00+00:00
Data partition in fold: 75
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-11 05:00:00+00:00 -- 2021-05-12 04:00:00+00:00
Data partition in fold: 76
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-12 05:00:00+00:00 -- 2021-05-13 04:00:00+00:00
Data partition in fold: 77
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-13 05:00:00+00:00 -- 2021-05-14 04:00:00+00:00
Data partition in fold: 78
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-14 05:00:00+00:00 -- 2021-05-15 04:00:00+00:00
Data partition in fold: 79
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-15 05:00:00+00:00 -- 2021-05-16 04:00:00+00:00
Data partition in fold: 80
    Training:   2017-01-08 23:00:00+00:00 -- 2021-02-25 04:00:00+00:00
    Validation: 2021-05-16 05:00:00+00:00 -- 2021-05-17 04:00:00+00:00
```

In [36]:

```
#Add index
predictions = predictions.set_index(df_test.index)
```

In [46]:

```
# Plot
# ==============================================================================
fig, ax = plt.subplots(figsize=(12, 3.5))
df.loc[df_test.index, 'Y'].plot(ax=ax, linewidth=2, label='real')
predictions.plot(linewidth=2, label='prediction', ax=ax)
ax.set_title('Autoregressive Prediction vs real Y')
ax.legend();
```



In [56]:

```
# ## Kpi
print('Autoregressive')
print('------------------------------')
print("R2 (explained variance):", round(metrics.r2_score(df_test["Y"], predictions), 3))
print("Mean Absolute Error (Σ|y-pred|/n):", "{:,.000f}".format(metrics.mean_absolute_error(
print("Root Mean Squared Error (sqrt(Σ(y-pred)^2/n)):", "{:,.000f}".format(np.sqrt(metrics.
print(f'Mean Absolute Perc Error : {metric}')
```

```
Autoregressive
------------------------------
R2 (explained variance): 0.829
Mean Absolute Error (Σ|y-pred|/n): 3,674
Root Mean Squared Error (sqrt(Σ(y-pred)^2/n)): 4,653
Mean Absolute Perc Error : 0.07106777102246506
```

In [39]:

```
#-----------------------------------
#Support Bector Regressor
ft_train, ft_test= np.split(df, [int(.8 *len(df))])
lb_train, lb_test= np.split(df.Y, [int(.8 *len(df))])
```

In [40]:

```
import numpy as np
np.random.seed(seed=5)
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
```

In [41]:

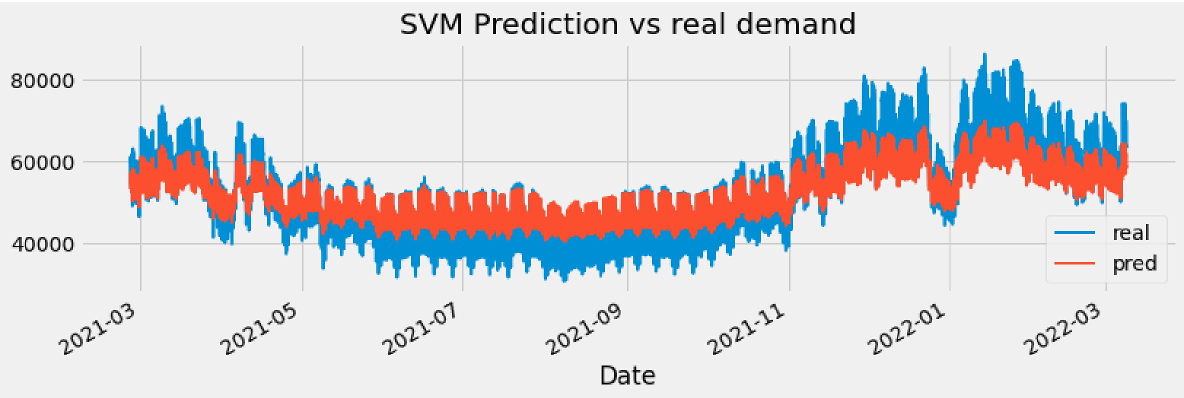
```
# Train the model using the training sets for c=10 and e=0.4
regr = SVR(C=10, epsilon=0.4)
regr.fit(ft_train, lb_train)

# Make predictions using the testing set
lb_pred = regr.predict(ft_test)
```

In [42]:

```
#predicted output values for test inputs
pred = lb_pred
# output values from the test set
test = lb_test

#Transefer test to 2d dataframe and add index
pred = pd. DataFrame(pred, columns=['pred'])
```

In [47]:

```
#Add index
pred = pred.set_index(test.index)

# Plot
# ==============================================================================
fig, ax = plt.subplots(figsize=(12, 3.5))
test.plot(ax=ax, linewidth=2, label='real')
pred.plot(linewidth=2, label='prediction', ax=ax)
ax.set_title('SVM Prediction vs real demand')
ax.legend();
```



In [45]:

```
from sklearn.metrics import r2_score, mean_absolute_error
from sklearn.metrics import mean_squared_error
from math import sqrt

print('Support Vector Regressor')
print('-------------------------------')

print('Accuracy : {}'.format(regr.score(ft_test, lb_test)))

MAE = mean_absolute_error(test, pred)
print('MAE : {}'.format(round(MAE, 2)))

MSE = mean_squared_error(test, pred)
print('MSE : {}'.format(round(MSE, 2)))

RMSE = sqrt(MSE)
print('RMSE  : %f' % RMSE)

R2_SCORE=r2_score(test, pred)
print('R2_SCORE  : %f' % R2_SCORE)
```

```
Support Vector Regressor
------------------------------
Accuracy : 0.800956018073169
MAE : 4032.66
MSE : 25194626.09
RMSE  : 5019.424876
R2_SCORE  : 0.800956
```

In [48]:

```
#---------------------------------------
#KNeighborsRegressor

target = pd.DataFrame(df['Y'])
labels = pd.DataFrame(df.drop('Y', axis=1 ))

#Check data status
print(labels.info())


# performing PCA on the dataset - simplifying the data dimentionality but retians the trend

from sklearn.decomposition import PCA

pca = PCA(n_components=17)
pca.fit(labels)

data = pca.transform(labels)

# view shape of thefeatures and labels
print(data.shape,labels.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 45202 entries, 2017-01-08 23:00:00+00:00 to 2022-03-08 22:00:
00+00:00
Data columns (total 18 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   solar_actual [MW]             45202 non-null  float64
 1   solar_forecast [MW]           45202 non-null  float64
 2   solar_inferred_capacity [MW]  45202 non-null  float64
 3   wind_actual [MW]              45202 non-null  float64
 4   wind_inferred_capacity [MW]   45202 non-null  float64
 5   albedo [%]                    45202 non-null  float64
 6   cloud_cover [%]               45202 non-null  float64
 7   frozen_precipitation [%]      45202 non-null  float64
 8   pressure [Pa]                 45202 non-null  float64
 9   radiation [W/m2]              45202 non-null  float64
 10  air_tmp [Kelvin]              45202 non-null  float64
 11  ground_tmp [Kelvin]           45202 non-null  float64
 12  apparent_tmp [Kelvin]         45202 non-null  float64
 13  wind_direction [angle]        45202 non-null  float64
 14  wind_speed [m/s]              45202 non-null  float64
 15  Month                         45202 non-null  int64  
 16  Weekday                       45202 non-null  int64  
 17  Hour                          45202 non-null  int64  
dtypes: float64(15), int64(3)
memory usage: 7.6 MB
None
(45202, 17) (45202, 18)
```

In [50]:

```
from sklearn.model_selection import train_test_split
#Split train and test
ft_train, ft_test, lb_train, lb_test = train_test_split(data , target, test_size=0.20, rand
print(ft_train.shape,ft_test.shape)
```

```
(36161, 17) (9041, 17)
```

In [52]:

```
from sklearn.neighbors import KNeighborsRegressor

# checking the accuracy while looping throught the neighbors count from 1 to 6
for n in range(1,6):
    knn = KNeighborsRegressor(n_neighbors = n)
    knn.fit(ft_train, lb_train)
    lb_pred = knn.predict(ft_test)
    print('KNeighborsRegressor: n = {} , Accuracy is: {}'.format(n,knn.score(ft_test, lb_te

#predicted output values for test inputs
pred = lb_pred
# output values from the test set
test = lb_test

print('KNeighborsRegressor')
print('------------------------------')
```

```
KNeighborsRegressor: n = 1 , Accuracy is: 0.5766530500832607
KNeighborsRegressor: n = 2 , Accuracy is: 0.6497238125094578
KNeighborsRegressor: n = 3 , Accuracy is: 0.6637257652856906
KNeighborsRegressor: n = 4 , Accuracy is: 0.6687391639391983
KNeighborsRegressor: n = 5 , Accuracy is: 0.6688536266183445
KNeighborsRegressor
------------------------------
```

In [55]:

```
# initialising the regressor for n=2
knn = KNeighborsRegressor(n_neighbors = 2)

# applying the model for the test values
knn.fit(ft_train, lb_train)

# predicting the out put values for test inputs
lb_pred = knn.predict(ft_test)

print('KNeighborsRegressor')
print('------------------------------')

print('Accuracy : {}'.format(knn.score(ft_test, lb_test)))

MAE = mean_absolute_error(test, pred)
print('MAE : {}'.format(round(MAE, 2)))

MSE = mean_squared_error(test, pred)
print('MSE : {}'.format(round(MSE, 2)))

RMSE = sqrt(MSE)
print('RMSE  : %f' % RMSE)

R2_SCORE=r2_score(test, pred)
print('R2_SCORE  : %f' % R2_SCORE)
```

```
KNeighborsRegressor
------------------------------
Accuracy : 0.6497238125094578
MAE : 5089.49
MSE : 45933791.64
RMSE  : 6777.447281
R2_SCORE  : 0.668854
```