Aniket Amar Thopte

Probabilistic Electric Load Forecasting

Introduction:

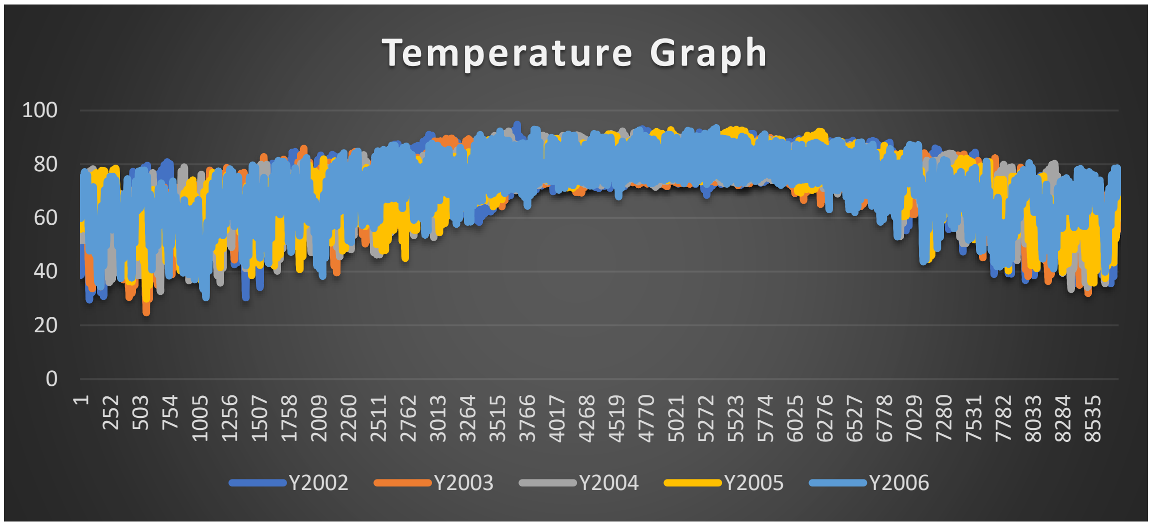
The problem statement deals with hourly probabilistic load forecasting. We have been given dataset from 2002 to 2006 (i.e. 5 years data) on hourly basis. We have to provide 9 percentiles (10,20,…..,90). In my case, I have used **Gradient Boosting Regressor** technique for forecasting the probabilistic load for 2007 year on hourly basis.

Dataset:

The Dataset file is named **'history.csv'**. The dataset provided has 4 variable columns namely 'Date' in dd/mm/yy (as per python date stamp), 'Hour' ranging from 1 to 24, 'Temperature' & 'Load'. The total **number of observations given is 43824**. Another excel file containing the 2007 data is provided named '**Fcst 2007.csv**' in which P10 to P90 percentile columns are kept blank to be forecasted. *Also, this time no temperature reading is given for the year 2007*.

Data-Preprocessing: For 2007 Temperature

In order to check if the temperature data has any seasonality, trend or just in simple terms is following the same pattern, we have plotted a line graph in Excel for years 2002 to 2006.



Hence, we can clearly see that the entire 5 years temperature data majorly (approximately) lies in a certain range. Hence in order to predict the 2007 temperature column we have taken the average of all 5 years temperature data for each day at hourly level, i.e. the temp for 1 Jan 2007 at hour 1 is the average of temp at hour 1 for 1st Jan (2002 to 2006).

Methodology:

The coding was done on Python (Jupyter Notebook). To generate prediction intervals using Scikit-Learn library, we'll use the Gradient Boosting Regressor technique. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

In our case the target variable is Load and the feature variables are Temperature , Hour & Date.

Code and Theory:

The following libraries have been used:-

NumPy:
It is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

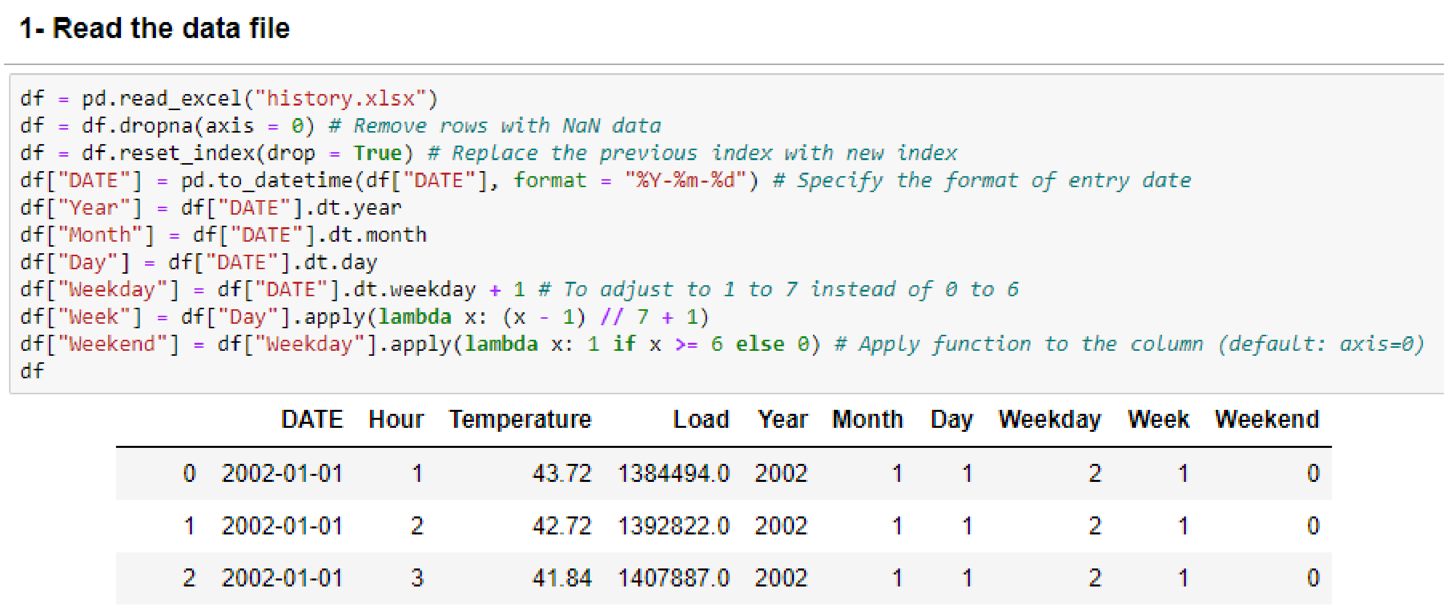
Pandas:
In computer programming, pandas are a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Scikit-learn:
Scikit-learn (also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Datetime:
The datetime module supplies classes for manipulating dates and times. While date and time arithmetic are supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation.

Calendar:
Python defines an inbuilt module calendar which handles operations related to calendar. Calendar module allows output calendars like the program and provides additional useful functions related to the calendar. Functions and classes defined in Calendar module use an idealized calendar, the current Gregorian calendar extended indefinitely in both directions. By default, these calendars have Monday as the first day of the week, and Sunday as the last (the European convention).

Step 1:
We first start by loading the data for reading and storing the data. From this data we first extract the year, month, day, weekday, week, weekend from the datestamp.

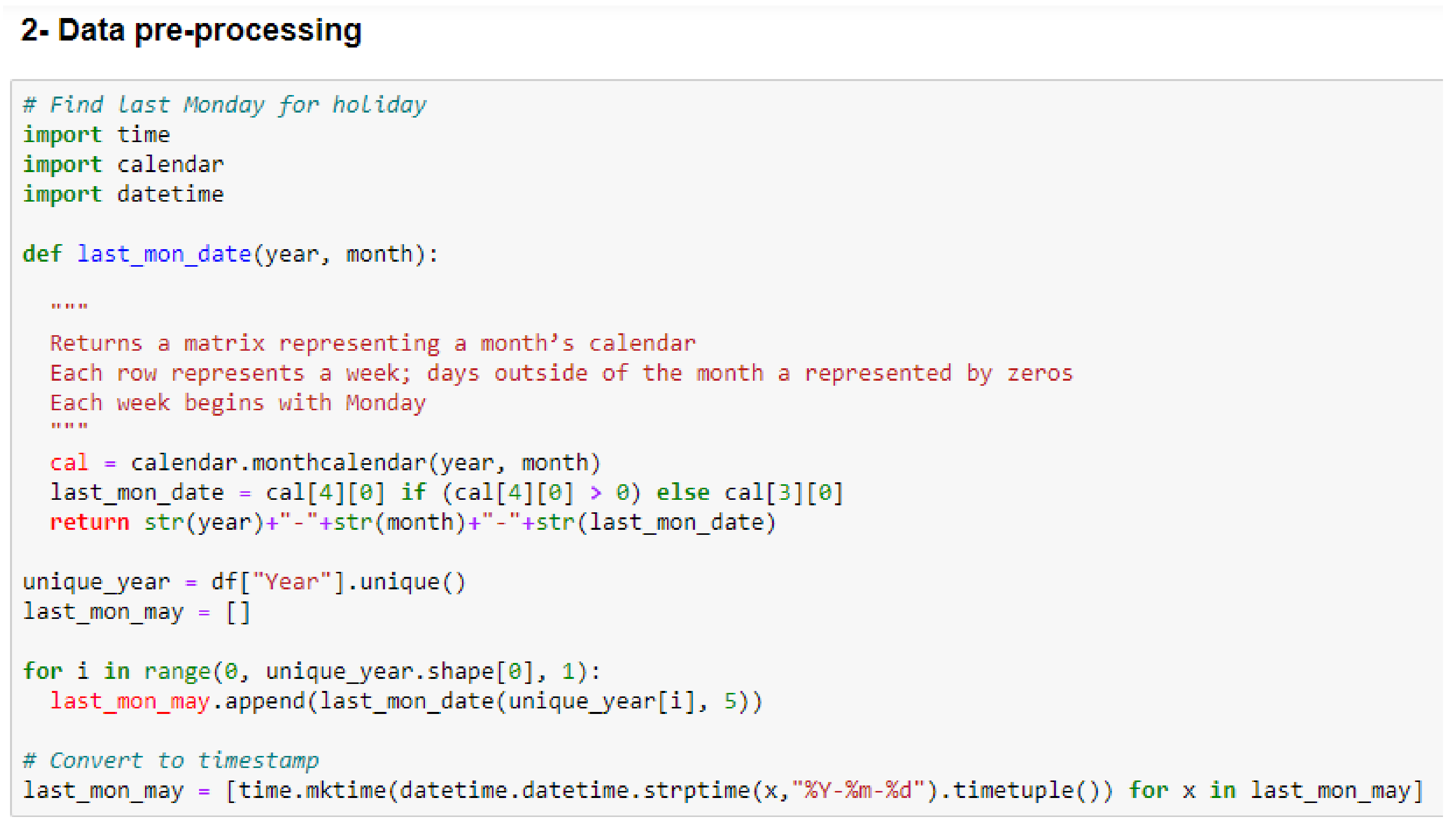


1- Read the data file

```
df = pd.read_excel("history.xlsx")
df = df.dropna(axis = 0) # Remove rows with NaN data
df = df.reset_index(drop = True) # Replace the previous index with new index
df["DATE"] = pd.to_datetime(df["DATE"], format = "%Y-%m-%d") # Specify the format of entry date
df["Year"] = df["DATE"].dt.year
df["Month"] = df["DATE"].dt.month
df["Day"] = df["DATE"].dt.day
df["Weekday"] = df["DATE"].dt.weekday + 1 # To adjust to 1 to 7 instead of 0 to 6
df["Week"] = df["Day"].apply(lambda x: (x - 1) // 7 + 1)
df["Weekend"] = df["Weekday"].apply(lambda x: 1 if x >= 6 else 0) # Apply function to the column (default: axis=0)
df
```

	DATE	Hour	Temperature	Load	Year	Month	Day	Weekday	Week	Weekend
0	2002-01-01	1	43.72	1384494.0	2002	1	1	2	1	0
1	2002-01-01	2	42.72	1392822.0	2002	1	1	2	1	0
2	2002-01-01	3	41.84	1407887.0	2002	1	1	2	1	0

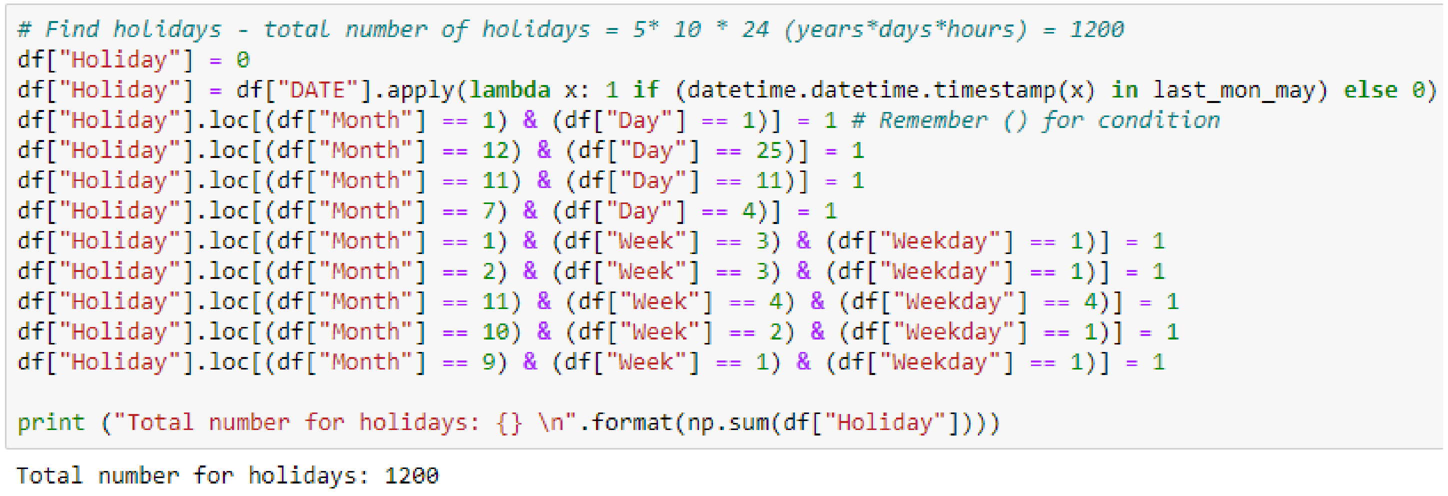
Step 2:
In the second step we set up the datestamp data in the calendar form using python calendar library. We set the Monday as 1st day of week by default.



2- Data pre-processing

```python
# Find last Monday for holiday
import time
import calendar
import datetime

def last_mon_date(year, month):

  """
  Returns a matrix representing a month’s calendar
  Each row represents a week; days outside of the month a represented by zeros
  Each week begins with Monday
  """
  cal = calendar.monthcalendar(year, month)
  last_mon_date = cal[4][0] if (cal[4][0] > 0) else cal[3][0]
  return str(year)+"-"+str(month)+"-"+str(last_mon_date)

unique_year = df["Year"].unique()
last_mon_may = []

for i in range(0, unique_year.shape[0], 1):
  last_mon_may.append(last_mon_date(unique_year[i], 5))

# Convert to timestamp
last_mon_may = [time.mktime(datetime.datetime.strptime(x,"%Y-%m-%d").timetuple()) for x in last_mon_may]
```

After this we process the data to find the number of holidays or laid off days in the datestamp to check the result of holiday feature on load forecasting.



```python
# Find holidays - total number of holidays = 5* 10 * 24 (years*days*hours) = 1200
df["Holiday"] = 0
df["Holiday"] = df["DATE"].apply(lambda x: 1 if (datetime.datetime.timestamp(x) in last_mon_may) else 0)
df["Holiday"].loc[(df["Month"] == 1) & (df["Day"] == 1)] = 1 # Remember () for condition
df["Holiday"].loc[(df["Month"] == 12) & (df["Day"] == 25)] = 1
df["Holiday"].loc[(df["Month"] == 11) & (df["Day"] == 11)] = 1
df["Holiday"].loc[(df["Month"] == 7) & (df["Day"] == 4)] = 1
df["Holiday"].loc[(df["Month"] == 1) & (df["Week"] == 3) & (df["Weekday"] == 1)] = 1
df["Holiday"].loc[(df["Month"] == 2) & (df["Week"] == 3) & (df["Weekday"] == 1)] = 1
df["Holiday"].loc[(df["Month"] == 11) & (df["Week"] == 4) & (df["Weekday"] == 4)] = 1
df["Holiday"].loc[(df["Month"] == 10) & (df["Week"] == 2) & (df["Weekday"] == 1)] = 1
df["Holiday"].loc[(df["Month"] == 9) & (df["Week"] == 1) & (df["Weekday"] == 1)] = 1

print ("Total number for holidays: {} \n".format(np.sum(df["Holiday"])))
```

```
Total number for holidays: 1200
```

Step 3:
Now since in the data pre-processing we have extracted the individual feature variables from the date, we now arrange them in an order of our own for our model.

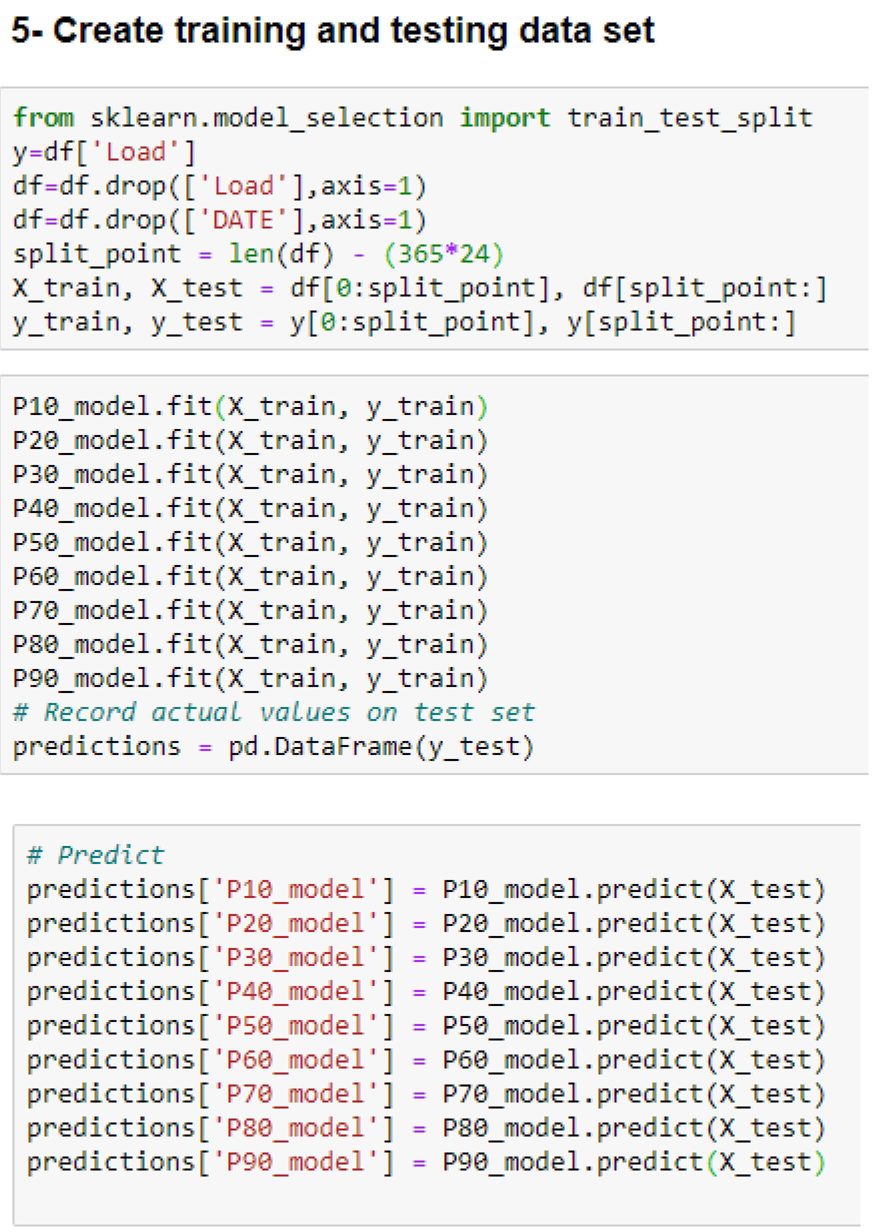
```
df = df[["DATE", "Hour", "Load","Temperature","Year","Month","Day","Weekday","Week","Weekend","Holiday"]]
df
```

	DATE	Hour	Load	Temperature	Year	Month	Day	Weekday	Week	Weekend	Holiday
0	2002-01-01	1	1384494.0	43.72	2002	1	1	2	1	0	1
1	2002-01-01	2	1392822.0	42.72	2002	1	1	2	1	0	1
2	2002-01-01	3	1407887.0	41.84	2002	1	1	2	1	0	1
3	2002-01-01	4	1438658.0	41.04	2002	1	1	2	1	0	1
4	2002-01-01	5	1484046.0	40.56	2002	1	1	2	1	0	1

Step 4:
We now build the Gradient Boosting Regressor (GBR) model using the Scikit-learn library. We have created 9 different individual models for the purpose of predicting 9 percentiles named P10 to P90. For the GBR model the value of alpha (α) is set according to the percentile, i.e. 0.1 for 10th percentile.

```
from sklearn.ensemble import GradientBoostingRegressor
N_ESTIMATORS = 100
MAX_DEPTH = 5
# Set P10 to P90  quantile
P10 = 0.1
P20=0.2
P30=0.3
P40=0.4
P50=0.5
P60=0.6
P70=0.7
P80=0.8
P90 = 0.9
# Each model has to be separate
P10_model = GradientBoostingRegressor(loss="quantile",
                                        alpha=P10, n_estimators=N_ESTIMATORS, max_depth=MAX_DEPTH)
P20_model = GradientBoostingRegressor(loss="quantile",
                                        alpha=P20, n_estimators=N_ESTIMATORS, max_depth=MAX_DEPTH)
P30_model = GradientBoostingRegressor(loss="quantile",
                                        alpha=P30, n_estimators=N_ESTIMATORS, max_depth=MAX_DEPTH)
```

Step 5:
After the forecast model is built, we now split the historical data of 5 years for training and testing the forecast model we have previously built.



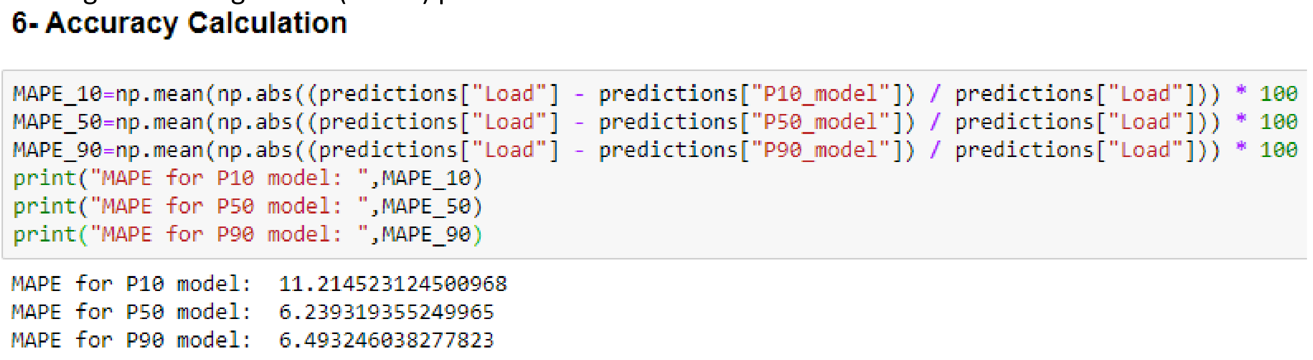
5- Create training and testing data set

```
from sklearn.model_selection import train_test_split
y=df['Load']
df=df.drop(['Load'],axis=1)
df=df.drop(['DATE'],axis=1)
split_point = len(df) - (365*24)
X_train, X_test = df[0:split_point], df[split_point:]
y_train, y_test = y[0:split_point], y[split_point:]
```

```
P10_model.fit(X_train, y_train)
P20_model.fit(X_train, y_train)
P30_model.fit(X_train, y_train)
P40_model.fit(X_train, y_train)
P50_model.fit(X_train, y_train)
P60_model.fit(X_train, y_train)
P70_model.fit(X_train, y_train)
P80_model.fit(X_train, y_train)
P90_model.fit(X_train, y_train)
# Record actual values on test set
predictions = pd.DataFrame(y_test)
```

```
# Predict
predictions['P10_model'] = P10_model.predict(X_test)
predictions['P20_model'] = P20_model.predict(X_test)
predictions['P30_model'] = P30_model.predict(X_test)
predictions['P40_model'] = P40_model.predict(X_test)
predictions['P50_model'] = P50_model.predict(X_test)
predictions['P60_model'] = P60_model.predict(X_test)
predictions['P70_model'] = P70_model.predict(X_test)
predictions['P80_model'] = P80_model.predict(X_test)
predictions['P90_model'] = P90_model.predict(X_test)
```

Step 6:
After the model has been trained and tested against the historical data, we check the accuracy of the model using Mean Average Percentage Error (MAPE) performance criteria.



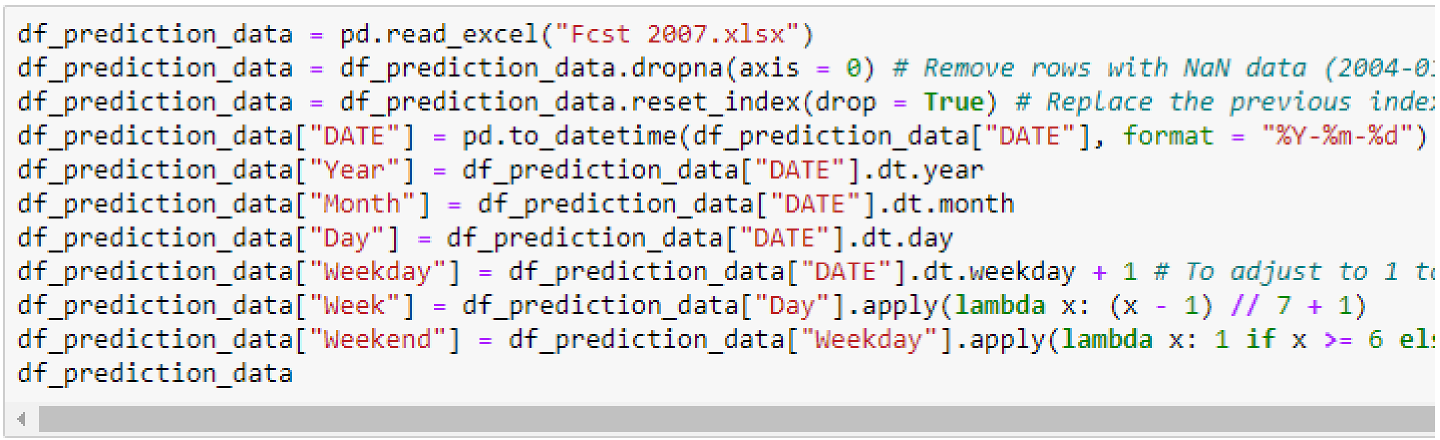
6- Accuracy Calculation

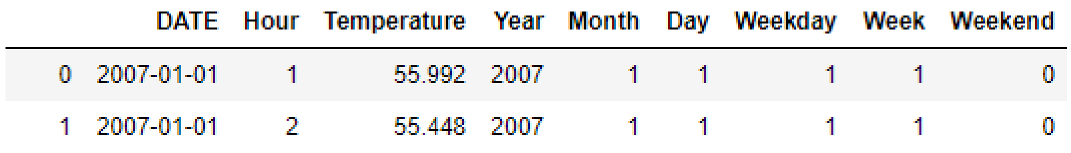
```
MAPE_10=np.mean(np.abs((predictions["Load"] - predictions["P10_model"]) / predictions["Load"])) * 100
MAPE_50=np.mean(np.abs((predictions["Load"] - predictions["P50_model"]) / predictions["Load"])) * 100
MAPE_90=np.mean(np.abs((predictions["Load"] - predictions["P90_model"]) / predictions["Load"])) * 100
print("MAPE for P10 model: ",MAPE_10)
print("MAPE for P50 model: ",MAPE_50)
print("MAPE for P90 model: ",MAPE_90)
```

```
MAPE for P10 model:  11.214523124500968
MAPE for P50 model:  6.239319355249965
MAPE for P90 model:  6.493246038277823
```

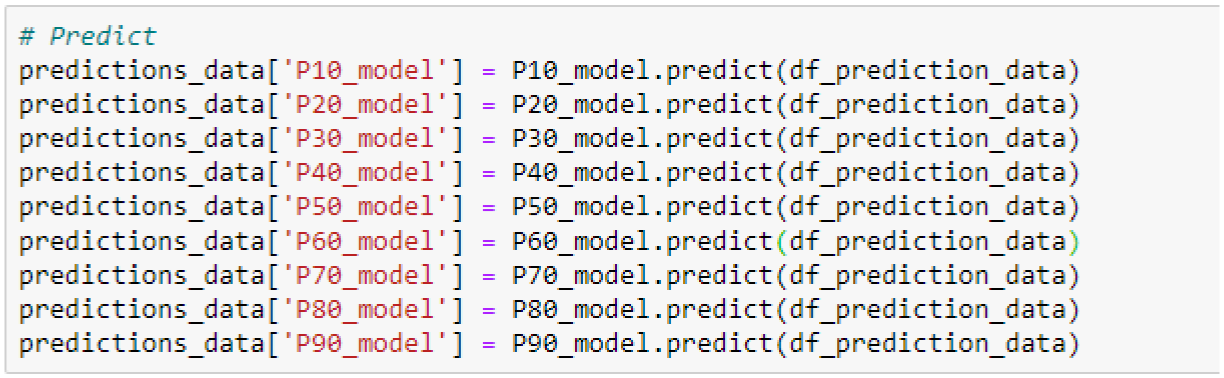
Step 7:
As we can clearly see the accuracy and error % of our model is less than 12% and even as low as 6% for P10, P50, P90, the model is acceptable. We now finally forecast the 9 quantiles for the year 2007. But before that we need to do the exact data pre-processing for the 2007 dataset and also arrange the data according to our order as specified in **Step 3.**

7- Making the Forecast


```
df_prediction_data = pd.read_excel("Fcst 2007.xlsx")
df_prediction_data = df_prediction_data.dropna(axis = 0) # Remove rows with NaN data (2004-0
df_prediction_data = df_prediction_data.reset_index(drop = True) # Replace the previous inde
df_prediction_data["DATE"] = pd.to_datetime(df_prediction_data["DATE"], format = "%Y-%m-%d")
df_prediction_data["Year"] = df_prediction_data["DATE"].dt.year
df_prediction_data["Month"] = df_prediction_data["DATE"].dt.month
df_prediction_data["Day"] = df_prediction_data["DATE"].dt.day
df_prediction_data["Weekday"] = df_prediction_data["DATE"].dt.weekday + 1 # To adjust to 1 t
df_prediction_data["Week"] = df_prediction_data["Day"].apply(lambda x: (x - 1) // 7 + 1)
df_prediction_data["Weekend"] = df_prediction_data["Weekday"].apply(lambda x: 1 if x >= 6 el
df_prediction_data
```

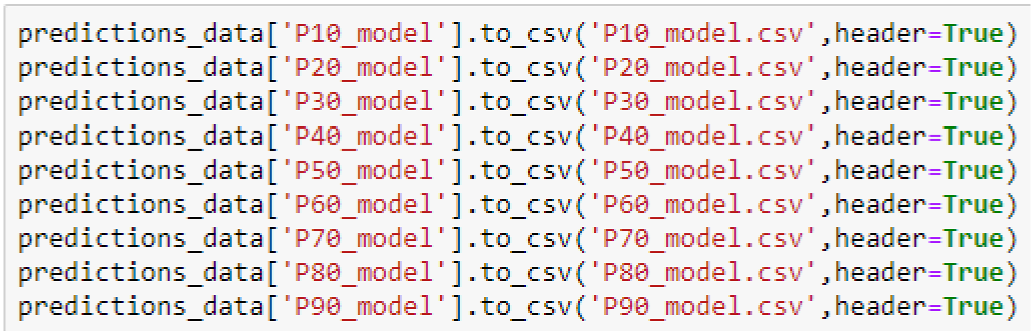


	DATE	Hour	Temperature	Year	Month	Day	Weekday	Week	Weekend
0	2007-01-01	1	55.992	2007	1	1	1	1	0
1	2007-01-01	2	55.448	2007	1	1	1	1	0


```
# Predict
predictions_data['P10_model'] = P10_model.predict(df_prediction_data)
predictions_data['P20_model'] = P20_model.predict(df_prediction_data)
predictions_data['P30_model'] = P30_model.predict(df_prediction_data)
predictions_data['P40_model'] = P40_model.predict(df_prediction_data)
predictions_data['P50_model'] = P50_model.predict(df_prediction_data)
predictions_data['P60_model'] = P60_model.predict(df_prediction_data)
predictions_data['P70_model'] = P70_model.predict(df_prediction_data)
predictions_data['P80_model'] = P80_model.predict(df_prediction_data)
predictions_data['P90_model'] = P90_model.predict(df_prediction_data)
```

Finally, we convert the predicted stored data into .csv files for completing the Fcst 2007 Excel file.

8- Converting the Forecasts into .csv file format


```
predictions_data['P10_model'].to_csv('P10_model.csv',header=True)
predictions_data['P20_model'].to_csv('P20_model.csv',header=True)
predictions_data['P30_model'].to_csv('P30_model.csv',header=True)
predictions_data['P40_model'].to_csv('P40_model.csv',header=True)
predictions_data['P50_model'].to_csv('P50_model.csv',header=True)
predictions_data['P60_model'].to_csv('P60_model.csv',header=True)
predictions_data['P70_model'].to_csv('P70_model.csv',header=True)
predictions_data['P80_model'].to_csv('P80_model.csv',header=True)
predictions_data['P90_model'].to_csv('P90_model.csv',header=True)
```

Conclusion:

Hence, we have used the GBR model for probabilistic forecasting, year ahead hourly based load. Our model is working.