Υλοποιητικό Project Data Mining

Φωτεινός Εμμανουήλ ΑΜ: 1067428 4ο Έτος
Βλαχογιάννης Δημήτρης ΑΜ:1067371 4ο Έτος

Το IDE που χρησιμοποιήθηκε είναι το Pycharm Community Edition, με τις εξής βιβλιοθήκες:

Ερώτημα 1:

```
import os
import glob
import numpy as np
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import
MinMaxScaler
from sklearn.metrics import
mean_absolute_error,
mean_squared_error, r2_score
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

Ερώτημα 2:

```
import numpy as np
import pandas as pd
from gensim.models import Word2Vec,
word2vec
from nltk.corpus import stopwords
from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import
precision_score, recall_score, f1_score
import re
import random
```

Είναι πιθανό να απαιτείται λήψη των Microsoft Visual C++ Redistributable και Ubuntu για την λειτουργία μερικών βιβλιοθηκών

ΕΡΩΤΗΜΑ Ι

Α:

Data cleanup:

Με την χρήση της βιβλιοθήκης pandas, ενώσαμε για κάθε ημερομηνία, κάθε csv αρχείο demand με το αντίστοιχο source του:

```python
if merge_files:
    i = 0
    exists = 1

    # first stage of merging, combine source and demand files
    while i < filenum:  # while there are files

        try:  # try to read a demand files
            a = pd.read_csv("demand/" + demand_filenames[i])  # save it in temporary variable a
            a = a.drop(index=288)  # remove the extra 0:00 timestamp at the end
        except pd.errors.EmptyDataError:  # if day is non-existent (eg. 31/02/2020)
            exists = 0
        try:  # repeat for source file
            b = pd.read_csv("sources/" + sources_filenames[i])
            if len(b) == 289:
                b = b.drop(index=288)
            b = b.drop(columns='Time')  # remove Time column so that it doesn't duplicate in merged file
            b = b.rename(columns={"Natural Gas": "Natural gas", "Large Hydro": "Large hydro"})  # fix inconsistency
        except pd.errors.EmptyDataError:
            exists = 0
        if exists == 1:  # if files/dates existed
            combined_csv = pd.concat([a, b], axis=1)  # merge
            combined_csv.insert(0, 'Date', demand_filenames[i][:9])  # Insert date/filename as a column
            # export as csv
            combined_csv.to_csv("combined/combined_csv_" + demand_filenames[i], index=False, encoding='utf-8-sig')
        i += 1
        exists = 1
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Date	Time	Day ahead	Hour ahea	Current de	Solar	Wind	Geotherm	Biomass	Biogas	Small hydr	Coal	Nuclear	Natural ga	Large hydr	Batteries	Imports	Other
2	20190101	0:00	23437	22290	22216	0	2810	993	380	225	200	11	2273	7326	1924	6	6254	0
3	20190101	0:05	22363	22089	22106	0	2862	993	381	226	201	11	2273	7200	1866	65	6266	0
4	20190101	0:10	22363	22089	22130	0	2916	993	380	226	202	11	2272	7057	1849	64	6319	0
5	20190101	0:15	22363	22089	22040	0	2920	993	378	223	203	11	2272	7007	1827	25	6354	0
6	20190101	0:20	22363	21867	21963	0	2902	993	379	223	203	11	2273	6970	1840	32	6360	0
7	20190101	0:25	22363	21867	21867	0	2874	992	379	223	205	11	2271	6926	1840	25	6302	0
8	20190101	0:30	22363	21867	21792	0	2845	993	379	223	203	11	2273	6919	1845	38	6270	0
9	20190101	0:35	22363	21626	21731	0	2807	993	378	224	203	11	2273	6885	1843	40	6263	0
10	20190101	0:40	22363	21626	21666	0	2763	992	377	224	198	11	2274	6789	1831	34	6338	0
11	20190101	0:45	22363	21626	21624	0	2735	992	377	218	197	11	2273	6761	1902	33	6352	0
12	20190101	0:50	22363	21415	21571	0	2693	992	377	207	193	11	2273	6693	1952	63	6323	0
13	20190101	0:55	22363	21415	21511	0	2618	992	377	207	192	11	2274	6728	1954	53	6270	0
14	20190101	1:00	22363	21415	21435	0	2554	992	377	207	191	11	2273	6808	1953	47	6225	0
15	20190101	1:05	21444	21304	21369	0	2515	992	377	207	191	10	2274	6788	1919	43	6232	0
16	20190101	1:10	21444	21304	21303	0	2496	992	378	208	193	11	2273	6802	1980	31	6158	0
17	20190101	1:15	21444	21304	21247	0	2488	992	378	208	194	11	2273	6796	2042	-1	6070	0
18	20190101	1:20	21444	21084	21184	0	2486	993	377	208	197	11	2272	6771	2017	-1	6036	0
19	20190101	1:25	21444	21084	21123	0	2514	992	379	207	198	11	2273	6746	1971	-1	6012	0
20	20190101	1:30	21444	21084	21063	0	2508	991	378	207	202	11	2274	6755	1953	-1	5979	0

Στην συνέχεια, συνδυάσαμε κάθε ενωμένο demand και source csv για κάθε ημερομηνία, σε ένα συνολικό csv αρχείο.

```python
# second stage of merging, merge combined source and demand files into one
i = 0
mega_csv = pd.read_csv("combined/combined_csv_" + demand_filenames[0])
i += 1
while i < filenum:
    try:
        temp = pd.read_csv("combined/combined_csv_" + demand_filenames[i])
        mega_csv = pd.concat([mega_csv, temp], axis=0)
    except FileNotFoundError:
        print(demand_filenames[i])
    i += 1
mega_csv.to_csv("mega/mega.csv", index=False, encoding="utf-8-sig")
print("merge ok")
```

Κατα την διάρκεια που προγραμματίζαμε το merge των αρχείων, παρατηρήσαμε πως υπήρχαν αρχεία/ημερομηνίες οπού το περιεχόμενο τους ήταν ένα κενό csv, όμως αυτές οι κενές μέρες ήταν μέρες που δεν υπήρχαν, όπως 31 Απρίλη ή 29 Φεβρουαρίου για μη δίσεκτα έτη. Έτσι απλά το αντιμετωπίσαμε παραλείποντας να τις συμπεριλάβουμε στο merge. Επίσης παρατηρήσαμε ότι υπήρχαν κενές τιμές σε τυχαία σημεία σε μερικά αρχεία, οπότε υπολογίσαμε τον μέσο όρο για κάθε timestamp για να τις συμπληρώσουμε

Με την χρήση του συνολικού csv αρχείου, υπολογίζουμε τον μέσο όρο για κάθε στήλη του αρχείου, για κάθε timestamp και το αποθηκεύουμε,:

```python
if create_total_mean:
    # load data
    mega_csv = pd.read_csv("mega/mega.csv")
    # create a blank csv with only the columns and their names
    mean_csv = mega_csv
    mean_csv = pd.DataFrame(columns=mean_csv.columns)

    index1 = 0  # index for hour
    index2 = 0  # index for minutes
    label = "00:00"  # variable to store timestamp
    while index1 < 24:  # loops used to parse through timestamps
        while index2 < 60:
            label = "00:00"
            if index2 < 10 and index1 < 10:
                label = "0" + str(index1) + ":0" + str(index2)
            elif index1 < 10:
                label = "0" + str(index1) + ":" + str(index2)
            elif index2 < 10:
                label = str(index1) + ":0" + str(index2)
            else:
                label = str(index1) + ":" + str(index2)
            fil = (mega_csv["Time"] == label)  # for all dates for this timestamp
            temp = mega_csv.loc[fil].mean(axis=0)  # calculate mean
            mean_csv = mean_csv.append(temp, ignore_index=True)  # insert the row into the dataframe
            mean_csv['Time'] = mean_csv['Time'].replace([np.nan], label)  # replace empty value with the timestamp
            index2 += 5  # + 5 minutes
        index2 = 00
        index1 += 1  # + 1 hour

    mean_csv.drop(mean_csv.columns[0], axis=1, inplace=True)  # remove Date column, not useful
    mean_csv = mean_csv.round(2)# round the result to 2 decimal digits
    mean_csv.to_csv("mega/mean.csv", index=False, encoding="utf-8-sig")  # export
    print("mean ok")
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Time	Day ahead	Hour ahea	Current de	Solar	Wind	Geotherm	Biomass	Biogas	Small hydr	Coal	Nuclear	Natural ga	Large hydr	Batteries	Imports	Other
2	0:00	24125.23	23351.04	23244.66	-14.61	2442.12	899.37	307.35	214.38	254.52	14.51	1866.74	8432.54	1876.92	-31.93	7375.4	0
3	0:05	22744.73	23068.22	23178.2	-14.67	2439.94	899.5	307.25	214.38	253.36	14.69	1866.64	8385.28	1825.21	-3.31	7402.85	0
4	0:10	22744.73	23068.22	23129.59	-14.72	2436.83	899.39	306.81	214.55	251.67	14.62	1866.53	8335.61	1801.08	-0.81	7419.3	0
5	0:15	22744.73	23068.22	23029.75	-14.77	2432.63	899.58	306.63	214.68	250.64	14.52	1866.53	8285.24	1785.93	-6.52	7404.2	0
6	0:20	22744.73	22784.66	22930.8	-14.79	2430.37	899.71	306.68	214.72	249.9	14.5	1866.53	8200.46	1778.45	-6.15	7408.29	0
7	0:25	22744.73	22784.66	22825.98	-14.82	2427.61	899.77	306.49	214.78	249.72	14.46	1866.48	8122.72	1770.63	-11.52	7393.82	0
8	0:30	22744.73	22784.66	22723.26	-14.84	2425.36	899.88	306.33	214.79	249.47	14.41	1866.52	8062.08	1760.98	-19.18	7372.81	0
9	0:35	22744.73	22482.52	22625.39	-14.92	2422.14	899.89	306.5	214.83	249.48	14.41	1866.53	8004	1748.51	-21.44	7356.92	0
10	0:40	22744.73	22482.52	22532.21	-14.81	2418.55	899.88	306.56	214.82	249.54	14.39	1866.61	7949.3	1737.48	-23.9	7339.59	0

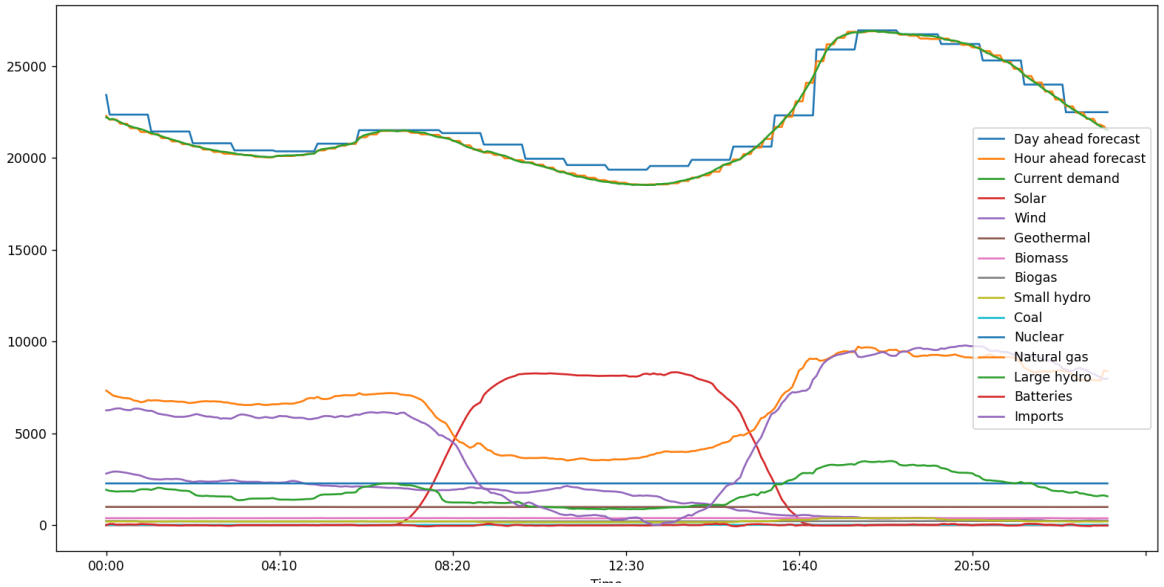
Στην συνέχεια συμπληρώσαμε τις κενές τιμές των αρχείων:

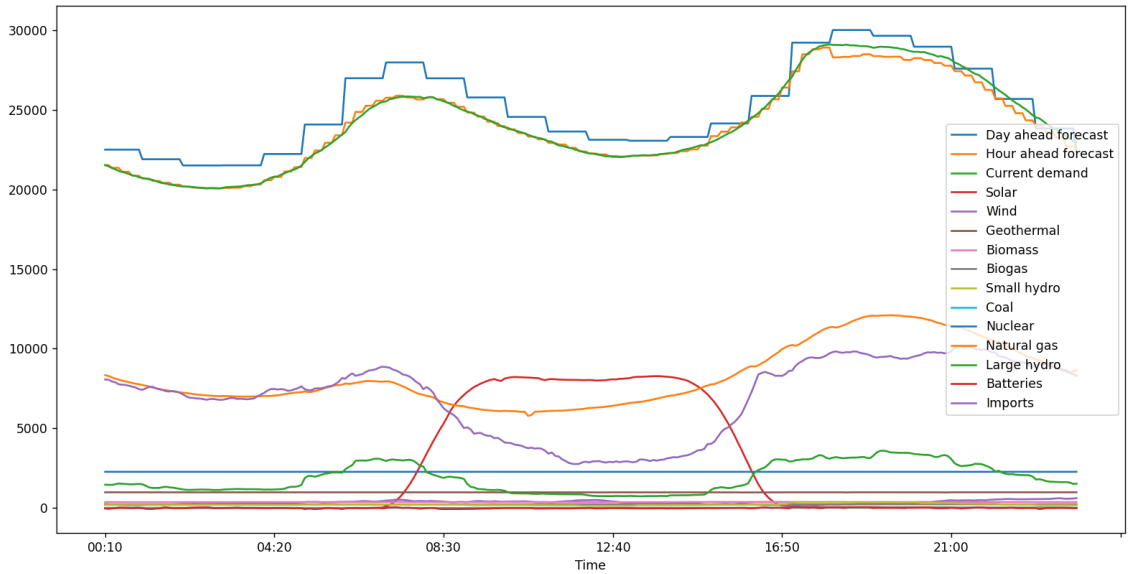
```python
if fill_missing_values:
    # read files
    mega_csv = pd.read_csv("mega/mega.csv")
    mean_csv = pd.read_csv("mega/mean.csv")

    # parse through whole file and for every missing value fill it with the mean of it's timestamp + column which is
    # already calculated in the mean.csv file
    for i in range(len(mega_csv.columns)):
        for j in range(len(mega_csv)):
            if pd.isnull(mega_csv.iloc[j, i]):
                time = mega_csv.iloc[j, 1]
                print(time)
                mega_csv.iloc[j, i] = mean_csv.loc[mega_csv["Time"] == time].iloc[0, i - 1]

    mega_csv.to_csv("mega/mega.csv", index=False, encoding="utf-8-sig")
```

Data analysis:

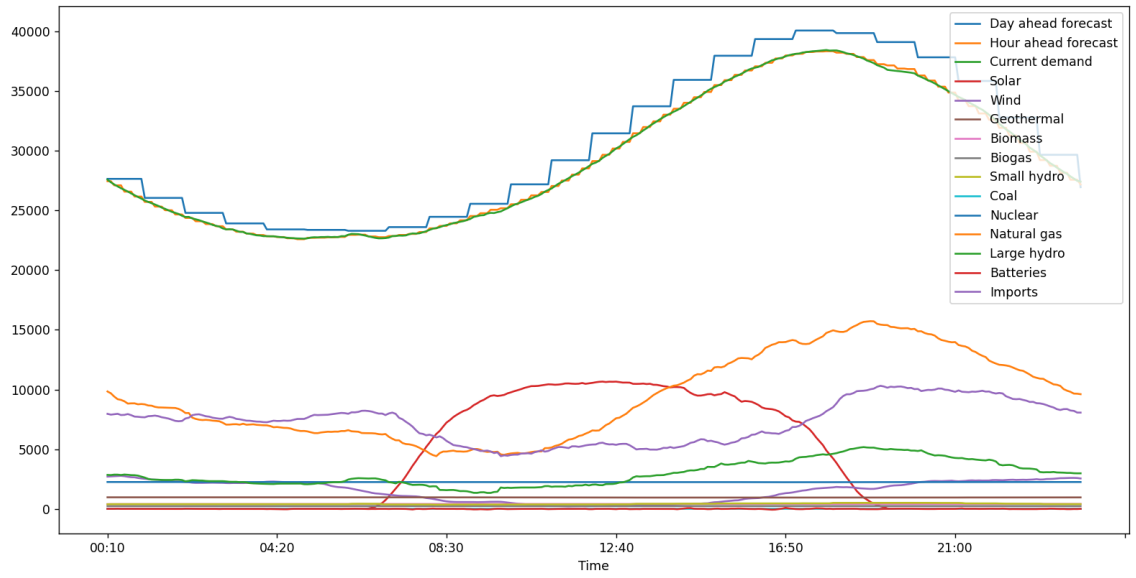
Παρατηρώντας τις συναρτήσεις της πρώτης, της δεύτερης,μίας τυχαίας μέρας και του μέσου όρου μπορούμε να βγάλουμε κάποια συμπεράσματα

Πρώτη και δεύτερη:

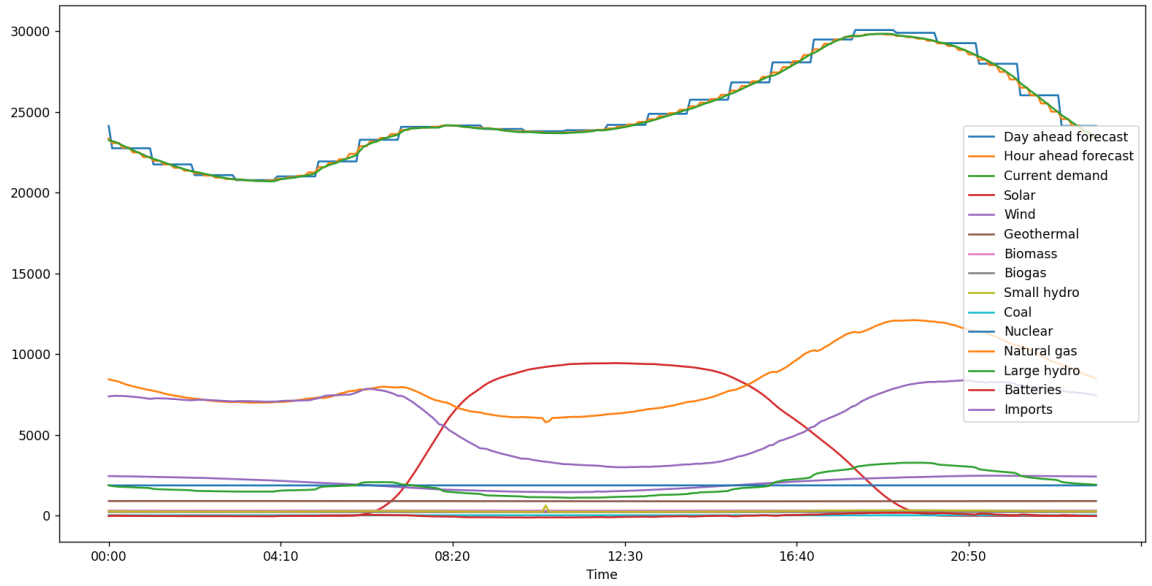




Τυχαία:



Μέσος όρος:

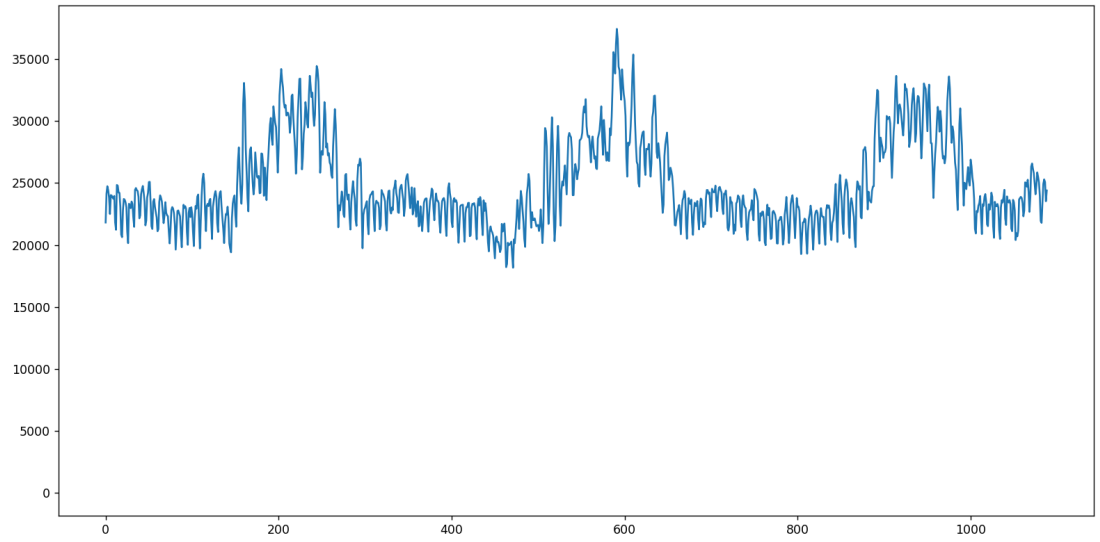


Το hour ahead forecast ακολουθεί με αρκετή ακρίβεια το Current demand, ενώ το day ahead forecast ακολουθεί με αρκετή ακρίβεια το Current demand της επόμενης ημέρας, όπως και είναι αναμενόμενο.

Η τιμή του current demand κορυφώνεται απο περίπου 17:00 έως 20:00

Η ηλιακή ενέργεια έχει μεγάλες τιμές κατα τη διάρκεια της μέρας και μηδενικές κατα την διάρκεια της νύχτας

Με την συνάρτηση της μέσης τιμής της ζήτησης για κάθε μέρας παρατηρούμε πως η ζήτηση αυξάνεται το καλοκαίρι:



B:

Για την συσταδοποίηση και τον εντοπισμό των ημερών outliers επιλέξαμε τον αλγόριθμο dbscan.

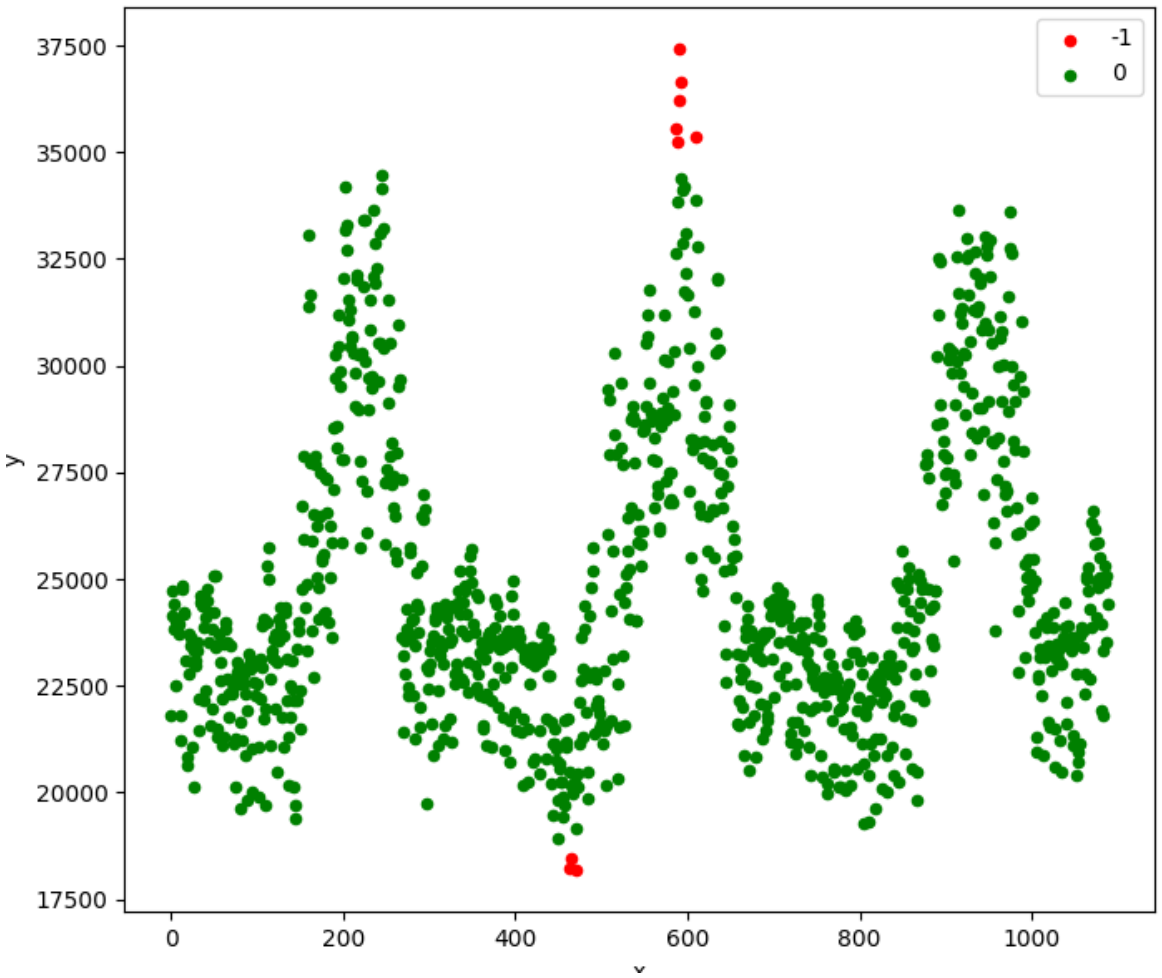
Παίρνουμε τις τιμές της του μέσου όρου της ζήτησης για τον y άξονα και τον αριθμό της ημέρας για τον x άξονα:

```
day_average_csv = pd.read_csv("mega/day_average.csv")
data = []
for j in range(len(day_average_csv)): # Store energy demand of every day in an indexed table
    data.insert(j, [j, day_average_csv.iloc[j, 3]])
data = np.array(data)  # Modify data into a numpy array
x = np.arange(len(data))  # Store values of x axis
```

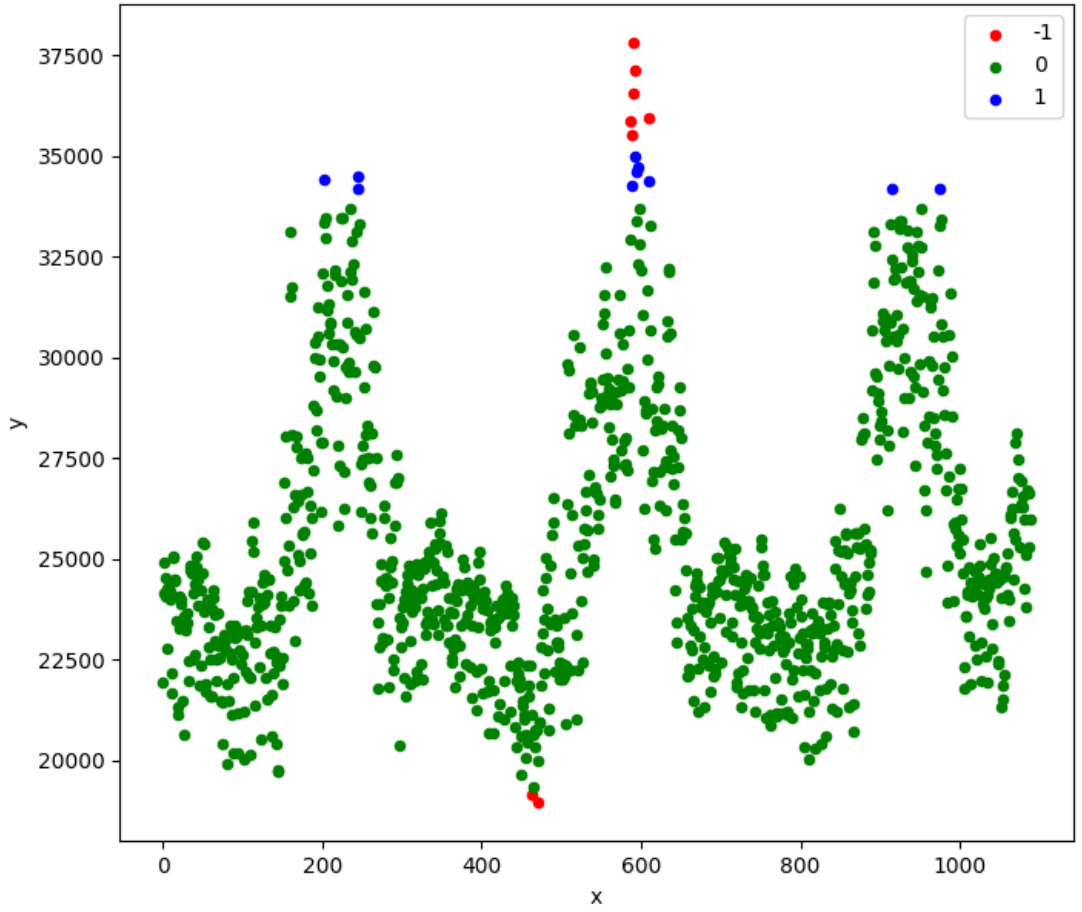
Στην συνέχεια τρέχουμε τον αλγόριθμο με eps = 400 και min_saples = 4:

```
algorithm = DBSCAN(eps=400, min_samples=4).fit(data)  # run DBSCAN with these parameters
```

Και κάνουμε plot τα αποτελέσματα:



Επαναλαμβάνουμε για τις τιμές των διαθέσιμων πηγών (αθροιστικά):



Οι outliers είναι σημειωμένοι με κόκκινο χρώμα

Οι παράμετροι επιλέχθηκαν μετά από πολλαπλές εκτελέσεις με διαφορετικές τιμές

Γ:

Αρχικά, υπολογίζουμε την διαφορά μεταξύ ζήτησης και παραγωγής ενέργειας από ανανεώσιμες πηγές:

```
mega_csv = pd.read_csv("mega/mega.csv")  # Calculate total renewable energy and remove it from current demand
mega_csv['Renewable'] = mega_csv[['Solar', 'Wind', 'Geothermal', 'Biomass', 'Biogas', 'Small hydro', 'Large hydro', 'Batteries']].sum(axis=1)
mega_csv['Non-Renewable demand'] = mega_csv['Current demand'] - mega_csv['Renewable']
mega_csv.loc[mega_csv['Non-Renewable demand'] < 0, 'Non-Renewable demand'] = 0
print(mega_csv)
data = mega_csv.filter(['Non-Renewable demand']).iloc[:]
```

Έτσι μας μένει η ενέργεια που απαιτείται να παραχθεί από μη ανανεώσιμες πηγές ώστε να καλυφθεί η ζήτηση.

Στην συνέχεια κάνουμε scale τα δεδομένα για καλύτερα αποτελέσματα (μετατροπή σε τιμές ανάμεσα στο 0 και 1) και καθορίζουμε το training data set, όπου και με βάση τις 300 πρώτες τιμές, θα υπολογίζεται η 301η

```
training_data_len = 15000  # Set training data length
z = 300

scaler = MinMaxScaler(feature_range=(0, 1))  # Scale data for better results
scaled_data = scaler.fit_transform(dataset)

train_data = scaled_data[0:training_data_len]  # Store the training data
x_train = []
y_train = []
for i in range(z, len(train_data)):
    x_train.append(train_data[i-z:i, 0])
    y_train.append(train_data[i])
```

Έπειτα κάνουμε reshape τα δεδομένα γιατί το LSTM απαιτεί τρισδιάστατα δεδομένα.

```
x_train, y_train = np.array(x_train), np.array(y_train)  # Change into numpy array
print(x_train.shape)
print(x_train)

x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))  # Reshape data
print(x_train.shape)
```

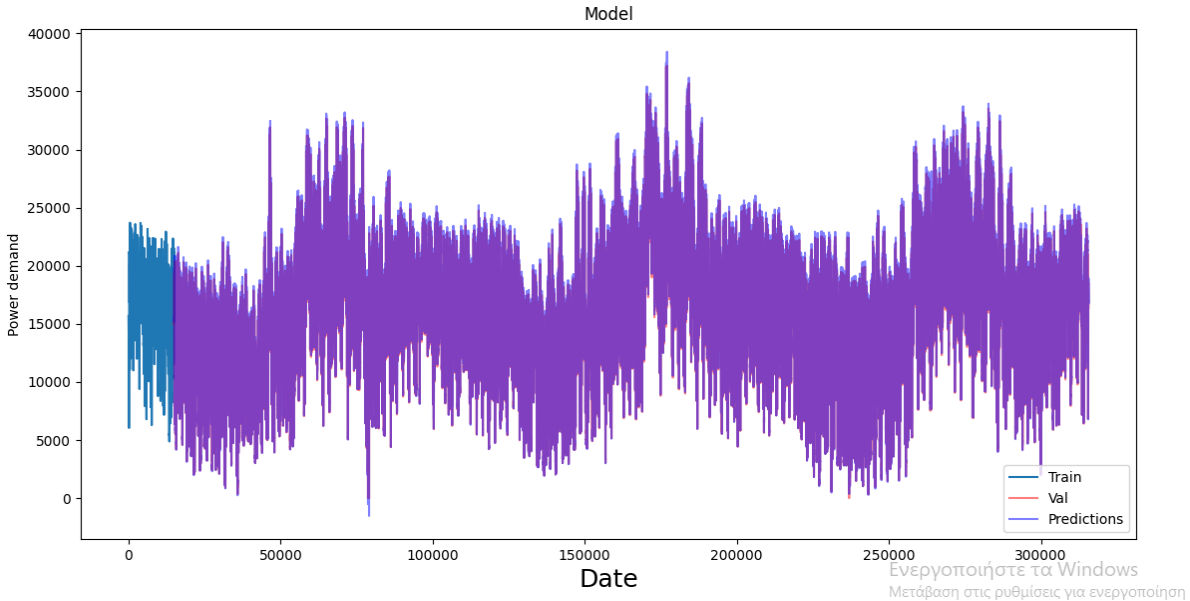
Ορίζουμε το μοντέλο του lstm και το εκπαιδεύουμε:

```
model = Sequential()  # Create LSTM model
model.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1], 1)))  # 50 neurons, first layer
model.add(LSTM(50, return_sequences=False))  # 50 neurons, second layer
model.add(Dense(25))  # Third layer, density layer
model.add(Dense(1))  # Fourth layer, density layer

model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(x_train, y_train, batch_size=1, epochs=1)  # Train the model
```

Καθορίζουμε το test dataset:

```
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(z, len(test_data)):
    x_test.append(test_data[i-z:i, 0])

x_test = np.array(x_test)  # Change into numpy array

x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))  # Reshape data
```

και τέλος, με την χρήση του μοντέλου κάνουμε plot τις εξής προβλέψεις:



με μετρικές αξιολόγησης:

```
mae:289.2069314397337
mse :108870.67831714929
rmse :277.3059158387134
r2:0.9965107415729935
```

Παρατηρώντας τα αποτελέσματα, συμπαιρένουμε πως το μοντέλο προβλέπει τις τιμές σε πολύ ικανοποιητικό βαθμο, το οποίο είναι λογικό, αφού το LSTM είναι ένα RNN με καλή απόδοση στην πρόβλεψη time series δεδομένων, όπως είναι αυτά του προβλήματος μας

ΕΡΩΤΗΜΑ 2

Αρχικά στο αρχείο amazon κάνουμε ένα τυχαίο ανακάτεμα των δεδομένων του και εκτελούμε την συνάρτηση PreProcess που αφαιρεί όσους χαρακτήρες δεν είναι γράμματα, τα stopwords (π.χ. i, the, my,…), μετατρέπει κάθε λέξη σε lower case και μετατρέπει κάθε Text σε μια λίστα από λέξεις :

```
df = pd.read_csv('amazon.csv')

text = []
x = 0


df = df.sample(frac=1).reset_index()    # Randomize the csv file
df = df.drop(columns='index')   # Remove previous index


for review in df['Text']:   # For every review in the csv remove stopwords, convert to lower case and split reviews into words, remove non-letters,
    text += PreProcess(review)
    x += 1
    print(x)
```

```
def PreProcess(review):

    temp = []
    text = re.sub("[^a-zA-Z]", " ", review) # Remove non-letters
    text = text.lower().split() # Convert to words to lower case and split sentences into words

    stops = set(stopwords.words("english")) # Remove stopwords
    text = [w for w in text if not w in stops]

    temp.append(text)

    return temp
```

Έπειτα χρησιμοποιούμε την συνάρτηση Word2vec για να δημιουργήσουμε ένα μοντέλο που θέτει ένα μοναδικό vector σε κάθε λέξη :

```
def Word2vec(text):

    model = word2vec.Word2Vec(text, workers=4, vector_size=100, min_count=40, window=10, sample=1e-3)
    model.save("word2vec.model")
```

Μετά χωρίζουμε το συνολικό Text σε train_text (80%) και test_text(20%) και εκτελούμε τη συνάρτηση Avg_Vec ωστε κάθε review του Text να χαρακτηρίζεται από ένα μοναδικό vector που προκύπτει από το μέσο όρο των vector των λέξεων του εκάστοτε review:

```
x = int(len(text) * 0.8)
y = int(len(text))



for index in range(x):
    train_text.append(text[index])
                                        # Split reviews into train_text
train_text_score = df.iloc[0:x, 1]

train_text_vecs = Avg_Vec(train_text, model) # Find the average vector of each sentence in the train_text
```

```
for index in range(x, y):
    test_text.append(text[index])
                                         # Split reviews into test_text
test_text_score = df.iloc[x:y, 1]

test_text_vecs = Avg_Vec(test_text, model) # Find the average vector of each sentence in the test_text
```

```
def Avg_Vec(text , model):

    review_vecs = np.zeros((len(text), 100), dtype='float32')
    count = 0

    for review in text:
        review_vecs[count] = Get_Avg_Vec(review,model) # Finds the average vector of each review and adds it in the review_vecs list
        count +=1

    return review_vecs
```

```
def Get_Avg_Vec(text,model):

    avg_vec = np.zeros((100,), dtype='float32')
    in2word = set(model.wv.index_to_key)
    wr = 0

    for word in text:
        if word in in2word: # Finds if the words exists in the model of vectors
            wr += 1
            avg_vec = np.add(avg_vec, model.wv[word])   # For each sentence adds the vectors of each word and divides by the number of words

    avg_vec = np.divide(avg_vec, wr)

    return avg_vec # Returns the average vector of each sentence
```

Από τις 2 λίστες με τους μέσους όρους που προκύπτουν την train_text_vecs και την test_text_vecs πρέπει να αφαιρέσουμε τα vectors που περιέχουν NaN και το αντίστοιχο Score καθε Text γιατί θα δημιουργήσουν πρόβλημα κατα την εκτέλεση του Random Forest στη συνέχεια :

```
nan_indices = list({k for k, l in np.argwhere(np.isnan(train_text_vecs))})  # Finds the index where the vector contains NaN
nan_indices.sort(reverse=True)
train_text_vecs = np.delete(train_text_vecs, nan_indices, axis=0) # From the train text vecs remove the vecs that contain NaN values and the corresponding train text score
for x in nan_indices:
    train_text_score.pop(x)
```

```
nan_indices = list({k for k, l in np.argwhere(np.isnan(test_text_vecs))})
nan_indices.sort(reverse=True)
test_text_vecs = np.delete(test_text_vecs, nan_indices, axis=0) # From the test text vecs remove the vecs that contain NaN values and the corresponding test text score
for x in nan_indices:
    test_text_score.pop(x+40000)
```

Εκτελούμε τον αλγόριθμο για το Random Forest χρησιμοποιώντας 1000 δέντρα και με βάση το train_text_vecs και train_text_score παίρνουμε μια πρόβλεψη test_text_score δίνοντας σαν είσοδο το test_text_vecs:

```
forest = RandomForestClassifier(n_estimators=1000)       # Fit a Random Forest to the train text using 1000 trees
forest.fit(train_text_vecs, train_text_score)
prediction = forest.predict(test_text_vecs)     # Find the prediction for the test text
```

Τελος εκτελούμε τις συναρτήσεις f1_score, precision_score και recall_score και μας δίνουν τα αποτελέσματα :

```
F1_Score: 0.26693259904320776
Precision_Score: 0.6198223316953555
Recall_Score: 0.2804509721723629
```