Εξόρυξη Δεδομένων
Υλοποιητικό Project

Βεργίνης Δημήτριος
1066634

2022

1 Άσκηση 1

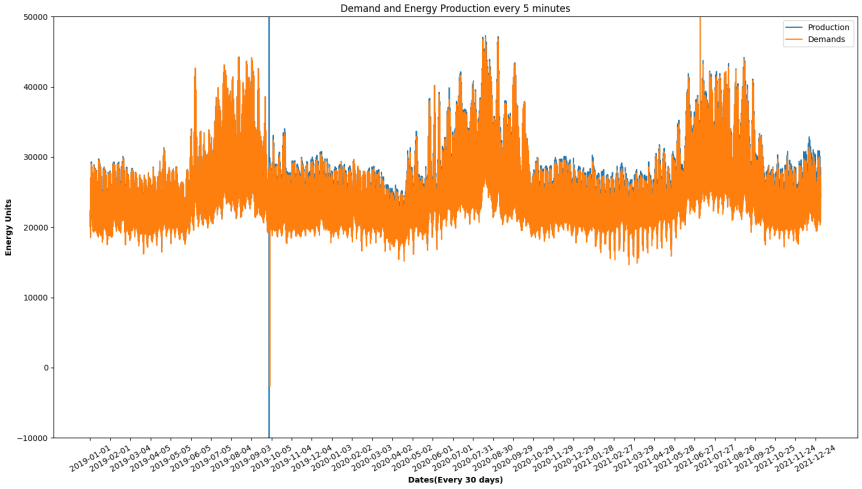
1.1 Α)

Για το συγκεκριμένο υπο-ερώτημα αρχικά βρήκαμε την τοποθεσία όλων των dataset μέσα στους φακέλους τόσο για τα sources όσο και για τα demands. Στη συνέχεια ξεκίνησε η διαδικασία συνένωσης όλων των δεδομένων. Η προ-επεξεργασία περιλάμβανε την αφαίρεση των NaN τιμών, την αφαίρεση των διπλότυπων και στην περίπτωση που κάποιο από τα dataset δεν είχε ακριβώς 288 σειρές που αντιστοιχούν στα πεντάλεπτα εντός ενός 24-ώρου τα αφαιρέσαμε. Ακόμη έγινε έλεγχος για την ύπαρξη των ημερομηνιών που διαβάζαμε κάθε φορά, δηλαδή στην περίπτωση που δεν υπήρχε πραγματική ημερομηνία ολόκληρο το dataset δεν θα διαβαζόταν(δε συναντήσαμε τέτοια περίπτωση).

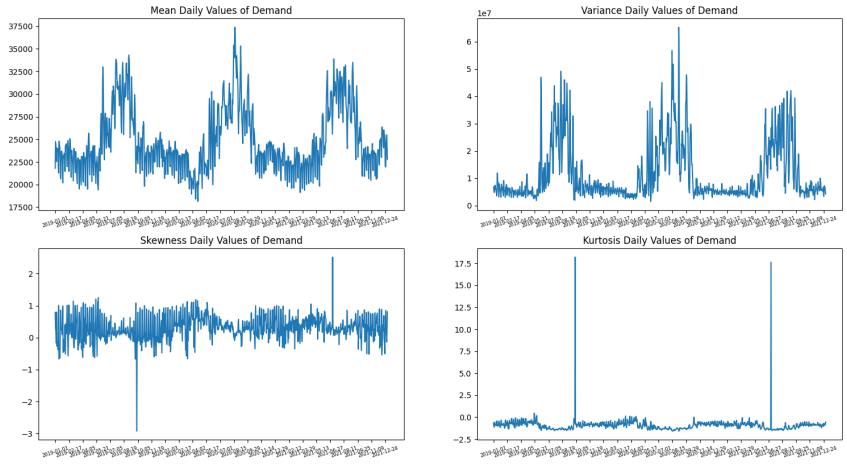
Εφόσον έγινε η συνένωση παρατηρήθηκε ότι δεν τα δυο συνενωμένα dataset δεν είχαν τον ίδιο αριθμό από δεδομένα γιατί κάποια περιείχαν αριθμό στοιχείων διαφορετικό του 288 που προαναφέρθηκε. Έτσι πέρα από τη βασική προ-επεξεργασία έγινε και μια διαδικασία που το βρέθηκαν οι μέρες που δεν κάλυπταν αυτή την απαίτηση και έτσι αφαιρέθηκαν.

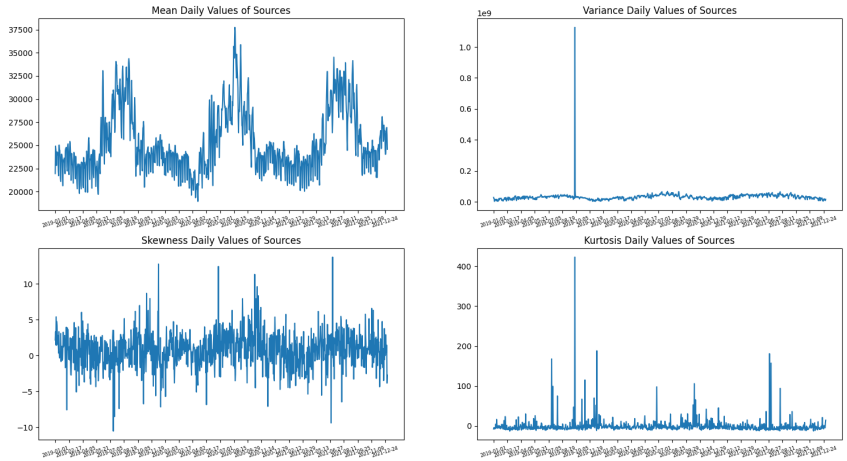
Πλέον τα δεδομένα μας ήταν σε θέση να τα επεξεργαστούμε και για αυτό το λόγο βρήκαμε τις βασικές ροπές και των demands και των sources αρχικά για κάθε μέρα ξεχωριστά. Στη συνέχεια για τα demands βρήκαμε και τις ροπές ανά τρίμηνο, ανά εξάμηνο και ανά χρόνο. Όσον αφορά τα sources δημιουργήσαμε και μια συνάρτηση η οποία υπολογίζει το duck curve. Δηλαδή τη συνολική απαίτηση σε ενέργεια σε ένα 24ωρο καθώς και μια άλλη γραφική που έχει αφαιρεθεί η ηλιακή ενέργεια για να δούμε τις απαιτήσεις σε άλλες μορφές ηλιακής ενέργειας για να καλυφθεί το κενό.

Παρακάτω βλέπουμε τις γραφικές παραστάσεις των ενεργειακών απαιτήσεων μαζί με την παραγωγή ενέργειας από τις διάφορες πηγές στο βάθος των τριών χρόνων. Παρατηρούμε ότι τις περισσότερες φορές οι πηγές υπερκαλύπτουν την απαίτηση, πράγμα που είναι απαραίτητο. Υπάρχουν όμως και μέρες στις οποίες αν παρατηρήσουμε η παραγωγή είναι αρκετά μειωμένη.

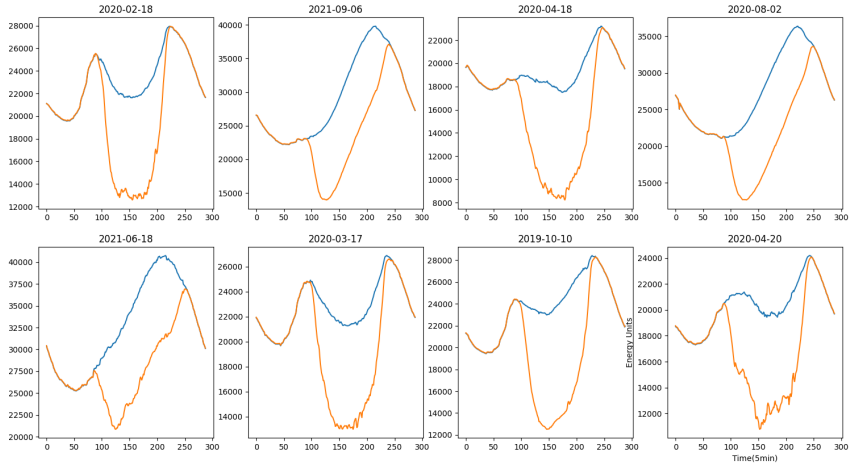


Εδώ βλέπουμε τις μαθηματικές ροπές τόσο για τις πηγές όσο και για τις απαιτήσεις και παρατηρούμε κάποιες ακραίες τιμές που ίσως οφείλεται σε σφάλμα των ακραίων τιμών (outliers).



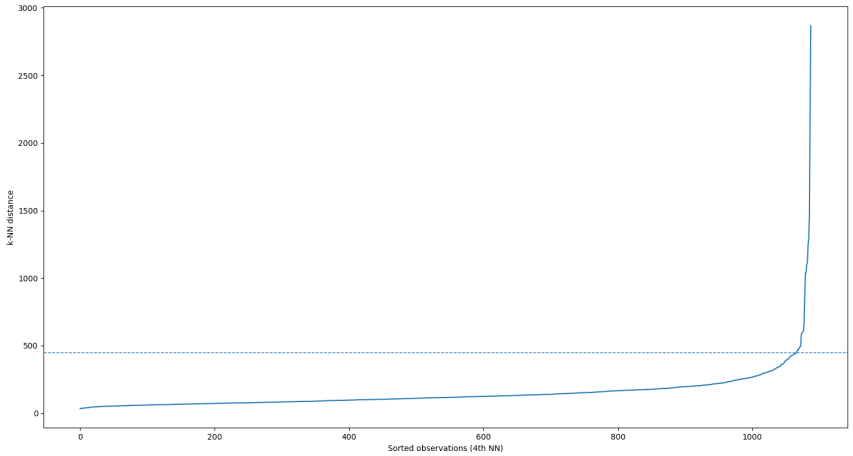


Τέλος ενδεικτικά φαίνονται τα duck curves για 8 ημέρες μέσα στο διάστημα που μας δίνεται.



1.2 Β)

Για την υλοποίηση αυτού του ερωτήματος χρησιμοποιούμε τη μέση τιμή της κάθε ημέρας για τα demands και του αθροίσματος των πηγών ενέργειας ως δεδομένα για τη συσταδοποίηση. Αρχικά εκτελούμε τον αλγόριθμο k-NN για να βρούμε την τιμή της ακτίνας που θα χρησιμοποιήσει ο DBSCAN για τα core points κάνοντας χρήση της elbow method. Παρακάτω φαίνεται η τιμή που έχει επιλεχθεί είναι περίπου ίση 450.

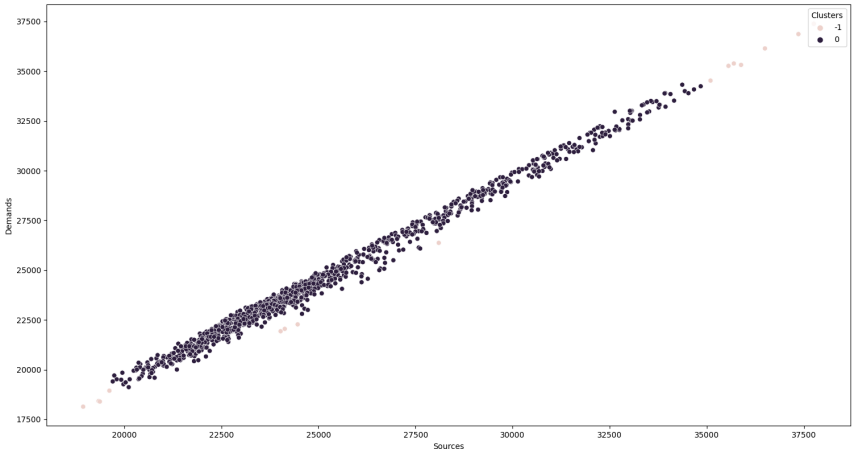


Στη συνέχεια εκτελούμε τον αλγόριθμο DBSCAN με την τιμή που βρήκαμε παραπάνω και ορίζουμε το ελάχιστο αριθμό σημείων (minimum number of points) ίσο με 4, το διπλάσιο δηλαδή όσο ο αριθμός των χαρακτηριστικών μας.

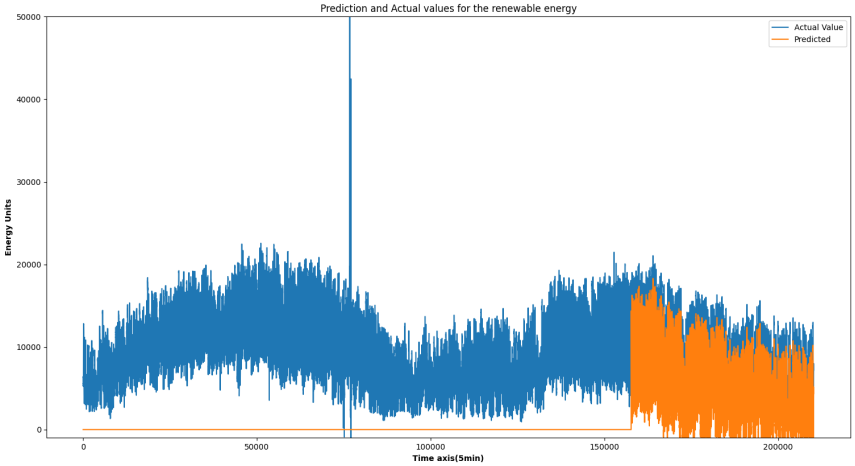
Στη συνέχεια απεικονίζουμε τις συστάδες που έχουν προκύψει καθώς και τα outliers με διαφορετικό χρώμα. Παρακάτω φαίνονται τα αποτελέσματα του clustering.

1.3 Γ)

Τέλος για το τρίτο ερώτημα υλοποιούμε μια πρόβλεψη για το πόση ενέργεια θα παράγεται κάθε στιγμή από τις ανανεώσιμες πηγές ενέργειας αφαιρώντας από το άθροισμα όλης της παραγωγής ενέργειας αυτή των μη ανανεώσιμων. Στη συνέχεια χωρίζουμε τα δεδομένα μας σε train και test. Λόγω υπολογιστικού φόρτου πήραμε μόνο για τα 2 χρόνια. Στη συνέχεια εκτελούμε έναν μετασχηματισμό min - max. Μετά τα δεδομένα τα μετατρέπουμε σε μια χρονοσειρά ανάλογα με το lookback που έχουμε ορίζει(στη συγκεκριμένη περίπτωση = 4) και τροφοδοτούμε τα δεδομένα σε ένα νευρωνικό δίκτυο που αποτελείται από



μια είσοδο όσο ακριβώς είναι κι το lookback και ένα σύνολο από 50 κύτταρα LSTM στο κρυφό επίπεδο. Βάλαμε να γίνει η εκπαίδευση με batchsize 16 και 15 εποχές και τα αποτελέσματα που λάβαμε είναι τα εξής:



Βλέπουμε ότι δεν έχει κάνει την τέλεια πρόβλεψη αλλά είναι πολύ κοντά στις πραγματικές τιμές, πράγμα που σημαίνει ότι με ένα μοντέλο λίγο καλύτερα εκπαιδευμένο(διαφορετικές παράμετροι, lookback) θα είχαμε πολύ καλύτερα αποτελέσματα και θα ήταν αξιόπιστο.

1.4 Βιβλιοθήκες

Οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση αυτής της άσκησης ήταν :

- **Numpy**, για αριθμητικούς υπολογισμούς με πίνακες
- **Matplotlib**, για τις γραφικές παραστάσεις
- **Datetime**, για τους ελέγχους με τις ημερομηνίες
- **Sklearn**, για τα μοντέλα των NearestNeighbors,DBSCAN, MinMaxScaler,TimeSeriesGenerator
- **Seaborn**, για τις γραφικές παραστάσεις
- **KERAS**, για τα νευρωνικά δίκτυα
- **Pandas**, για τον χειρισμό των δεδομένων

1.5 Κώδικας

```
1
2  from operator import index
3  from tokenize import group
4  from unicodedata import numeric
5  import numpy as np
6  import pandas as pd
7  from matplotlib import pyplot as plt
8  import os
9  import re
10 from datetime import datetime
11 from sklearn.neighbors import NearestNeighbors
12 from sklearn.cluster import DBSCAN
13
14 import seaborn as sns
15 from keras.layers import LSTM,Dense
16 from keras import models
17 from keras.models import Sequential
18 from keras.preprocessing.sequence import TimeseriesGenerator
19 from sklearn.preprocessing import MinMaxScaler
20 np.random.seed(1066634)
21
22
23
24
25
```

```
26
27 def getDirectories(path):
28     """
29     It takes a path as an argument and returns a list of all
    the files in that path
30
31     :param path: the path to the directory containing the
    data
32     :return: A list of all the files in the directory
33     """
34     demandDir = []
35
36     for root,dirs,files in os.walk(path):
37         for dataset in files:
38             temp = os.path.join(root,dataset)
39             if os.stat(temp).st_size == 0:
40                 continue
41             demandDir.append(temp)
42
43     return demandDir
44
45
46 def mergeFrames(dirList,cols):
47     """
48     It takes a list of file paths, and a list of column names
    , and returns a dataframe with the data
49     from all the files in the list
50
51     :param dirList: a list of all the files in the directory
52     :param cols: the columns of the dataframe
53     :return: A dataframe with the columns specified in the
    cols list.
54     """
55     temp = pd.DataFrame(columns=cols)
56     for dataset in dirList:
57         df = pd.read_csv(dataset)
58         df.rename(columns=lambda x: x.lower(),inplace=True)
59         df.rename(columns=lambda x: x.title(),inplace=True)
60         df.drop_duplicates(subset=["Time"], keep="first",
    inplace=True)
61         if len(df)!=288:
62             continue
63
64         name = nameREGEX.findall(dataset)[0]
65         try:
66             year = int(name[:4])
67             month = int(name[4:6])
68             day = int(name[6:8])
69             date = datetime(year,month,day)
```

```
            finalDate = date.strftime("%Y-%m-%d")

        except Exception:
            continue
        df["Datetime"] = finalDate
        temp = pd.concat([temp, df])

    return temp

def findDailyMomentsDemands(df):
    temp = pd.DataFrame(columns=["Mean","Variance","Skewness","Kurtosis"])
    meanDemands = df.groupby("Datetime").mean()["Current Demand"]
    varDemands = df.groupby("Datetime").var()["Current Demand"]
    skewDemands = df.groupby("Datetime").apply(pd.DataFrame.skew,numeric_only=True)["Current Demand"]
    kurtDemands = df.groupby("Datetime").apply(pd.DataFrame.kurt,numeric_only=True)["Current Demand"]

    temp["Mean"] = meanDemands
    temp["Variance"] = varDemands
    temp["Skewness"] = skewDemands
    temp["Kurtosis"] = kurtDemands

    return temp.reset_index()

def findMonthMoments(df,dates,months=3):
    """
    It takes a dataframe, a list of dates, and the number of months to be considered. It then creates a
    new dataframe with the mean, variance, skewness, and kurtosis of the dataframe for each date in the
    list

    :param df: The dataframe that contains the data
    :param dates: A list of dates in the format "YYYY-MM"
    :param months: The number of months to consider for each datetime, defaults to 3 (optional)
    :return: A dataframe with the mean, variance, skewness, and kurtosis of the demand for each month.
    """
    counter = 0
    final = pd.DataFrame(columns=["Mean","Variance","Skewness","Kurtosis","Datetime"])
    temp = pd.DataFrame(columns=df.columns)
    for date in dates:
        temp = pd.concat([temp, df.loc[df["Datetime"].str.
```

```
    contains(date)]])
109         counter+=1
110         if counter == months:
111             meanVal = temp["Current Demand"].mean(axis=0)
112             varVal = temp["Current Demand"].var(axis=0)
113             skewVal = temp["Current Demand"].skew(axis=0)
114             kurtVal = temp["Current Demand"].kurtosis(axis=0)
115             final = pd.concat([final,pd.DataFrame({"Mean":[
    meanVal],"Variance":[varVal],"Skewness":[skewVal],"
    Kurtosis":[kurtVal],"Datetime":[date]})], ignore_index=
    True)
116             temp = pd.DataFrame(columns=df.columns)
117             counter =0
118 
119     return final
120 
121 
122 def findDailyMomentsSources(df):
123     """
124     It takes a dataframe and returns the mean, variance,
    skewness, and kurtosis of each column, grouped
125     by the datetime column
126 
127     :param df: dataframe
128     :return: The mean, variance, skew, and kurtosis of the
    dataframe grouped by datetime.
129     """
130 
131     meanSources = df.groupby("Datetime").mean()
132     varSources = df.groupby("Datetime").var()
133     skewSources = df.groupby("Datetime").apply(pd.DataFrame.
    skew,numeric_only=True)
134     kurtSources = df.groupby("Datetime").apply(pd.DataFrame.
    kurt,numeric_only=True)
135 
136     return meanSources,varSources,skewSources,kurtSources
137 
138 
139 
140 def create_dataset(dataset, look_back=1):
141   dataX, dataY = [], []
142   for i in range(len(dataset)-look_back-1):
143     a = dataset[i:(i+look_back), 0]
144     dataX.append(a)
145     dataY.append(dataset[i + look_back, 0])
146   return np.array(dataX), np.array(dataY)
147 
148 def removeNonMatchingDates(demands,sources,dates):
149     """
```

```
    It takes in the demands dataframe, the sources dataframe, and the dates list. It then creates a set
    of the dates in the sources dataframe. It then iterates through the dates list and checks if each
    date is in the set. If it is not, it adds it to a list of excluded dates. Finally, it iterates
    through the excluded dates list and drops the rows in the demands dataframe that have those dates

    :param demands: the dataframe of demands
    :param sources: the dataframe containing the source data
    :param dates: a list of datetime objects
    """
    temp = set(sources["Datetime"])
    excludedDates = []
    for date in dates:
        if date not in temp:
            excludedDates.append(date)
    for ex in excludedDates:
        demands.drop(demands.index[demands["Datetime"]==ex], inplace=True)

def findDuckCurve(demands,sources):
    """
    It takes in the demand dataframe and the sources dataframe, and returns a groupby object of the
    demand dataframe with the solar data subtracted from it

    :param demands: a dataframe of the demand data
    :param sources: a dictionary of dataframes, each containing the data for a source of energy
    :return: A groupby object
    """
    solar = sources["Solar"].reset_index(drop =True).copy()
    curDemand = demands[["Current Demand","Datetime"]].copy()

    temp = curDemand["Current Demand"] - solar
    curDemand["Demand Without Solar"] = temp

    groups = curDemand.groupby("Datetime")

    return groups

def plotDuck(groupedValues, date):
    """
    It takes in a dataframe, and a date, and plots the current demand and demand without solar for that
    date

```

```
    :param groupedValues: the dataframe grouped by date
    :param date: The date you want to plot
    """
    grp = groupedValues.get_group(date).reset_index(drop =
    True)
    plt.plot(grp["Current Demand"])
    plt.plot(grp["Demand Without Solar"])
    plt.title(date)


# create lists with the directories of every dataset in the
    demands
# and sources folders and sorts them
demandPath = "./demand"
sourcesPath = "./sources"
nameREGEX = re.compile(r'[0-9]+')
demandDir = sorted(getDirectories(demandPath))
sourcesPath = sorted(getDirectories(sourcesPath))

# lists with the column names of the merged dataframes to be
    created
colsDemands =["Day Ahead Forecast","Hour Ahead Forecast","
    Current Demand","Datetime"]
colsSources =["Time","Solar","Wind","Geothermal","Biomass","
    Biogas","Small Hydro","Coal","Nuclear","Natural Gas",\
                "Large Hydro","Batteries","Imports","Other","
    Datetime"]

# get the dates of every day in the dataset
datesDemands = [nameREGEX.findall(x)[0] for x in demandDir]
datesDemands = [f'{x[:4]}-{x[4:6]}-{x[6:8]}' for x in
    datesDemands]
# get the monts of every dataset
months = [x[:-3] for x in datesDemands]
months = sorted(list(set(months)))

# merge the dataframes fill all null values with 0 and reset
    the index
# in order to have a common index for every 5 minutes of the
    day
mergedDemands = mergeFrames(demandDir,colsDemands).
    reset_index(drop =True).dropna()
mergedSources = mergeFrames(sourcesPath,colsSources).
    reset_index(drop =True).dropna()

removeNonMatchingDates(mergedDemands,mergedSources,
    datesDemands)
mergedDemands = mergedDemands.reset_index(drop =True)
mergedSources = mergedSources.reset_index(drop =True)
```

```
228
229
230 # store all the csvs for easier manipulation and time saving
231 mergedDemands.to_csv("./MergedDemands.csv")
232 mergedSources.to_csv("./MergedSources.csv")
233
234
235 mergedDemands = pd.read_csv("./MergedDemands.csv", index_col=
        0)
236 mergedSources = pd.read_csv("./MergedSources.csv", index_col=
        0)
237
238 # get all the mathematical moments of sources and demands(
       mean, variance)
239 # skewness and kurtosis. for the demands get the mathematical
        moments
240 # for a 3-month, 6-month and 12-month period
241 dailyMomentsDemands = findDailyMomentsDemands(mergedDemands)
242 quarterMoments = findMonthMoments(mergedDemands,months,3)
243 halfyearMoments = findMonthMoments(mergedDemands,months,6)
244 yearlyMoments = findMonthMoments(mergedDemands,months,12)
245 dailyMomentsSources = findDailyMomentsSources(mergedSources)
246
247 dailyMomentsSources[0].to_csv("./MeanSources.csv")
248 dailyMomentsSources[1].to_csv("./VarSources.csv")
249 dailyMomentsSources[2].to_csv("./SkewSources.csv")
250 dailyMomentsSources[3].to_csv("./KurtSources.csv")
251 dailyMomentsDemands.to_csv("./DemandMoments.csv")
252
253
254
255 dailyMomentsDemands = pd.read_csv("./DemandMoments.csv",
       index_col= 0).reset_index()
256 meanSources = pd.read_csv("./MeanSources.csv", index_col= 0).
       reset_index()
257 varSources = pd.read_csv("./VarSources.csv", index_col= 0).
       reset_index()
258 kurtSources = pd.read_csv("./KurtSources.csv", index_col= 0).
       reset_index()
259 skewSources = pd.read_csv("./SkewSources.csv", index_col= 0).
       reset_index()
260
261 # # get the grouby object of the duckcurve of every day
262 groups=findDuckCurve(mergedDemands,mergedSources)
263
264 #-----------------------------------------COMMENT THIS
       BLOCK IF NOT USING QT BACKEND
       ---------------------------------------#
265 #-----------------------------------------COMMENT THIS
```

```
    BLOCK IF NOT USING QT BACKEND
    ----------------------------------------#
#----------------------------------------COMMENT THIS
    BLOCK IF NOT USING QT BACKEND
    ----------------------------------------#
#----------------------------------------COMMENT THIS
    BLOCK IF NOT USING QT BACKEND
    ----------------------------------------#
#----------------------------------------COMMENT THIS
    BLOCK IF NOT USING QT BACKEND
    ----------------------------------------#
# manager = plt.get_current_fig_manager()
# manager.window.showMaximized()
#
    ------------------------------------------------------------------------------

#
    ------------------------------------------------------------------------------

#
    ------------------------------------------------------------------------------

#
    ------------------------------------------------------------------------------


plt.figure(1)
plt.plot(mergedSources.sum(axis=1, numeric_only=True))
plt.plot(mergedDemands["Current Demand"])

days = 30
labels = sorted(list(set(mergedDemands["Datetime"])))[0::days
    ]
dayTicks = [i*288*days for i in range(len(labels))]
plt.xticks(dayTicks, labels ,rotation=30)
plt.title("Demand and Energy Production every 5 minutes")
plt.xlabel("Dates(Every 30 days)",fontweight="bold")
plt.ylabel("Energy Units",fontweight="bold")
plt.ylim([-10_000,50_000])
plt.legend(["Production","Demands"])


plt.figure(2)
days = 45
labels = dailyMomentsDemands["Datetime"][0::days]
ticks = [x for x in range(0,len(dailyMomentsDemands["Datetime
    "]),days)]

plt.subplot(2,2,1)
```

```
plt.plot(dailyMomentsDemands["Mean"])
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)
plt.title("Mean Daily Values of Demand")
plt.subplot(2,2,2)
plt.plot(dailyMomentsDemands["Variance"])
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)
plt.title("Variance Daily Values of Demand")
plt.subplot(2,2,3)
plt.plot(dailyMomentsDemands["Skewness"])
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)
plt.title("Skewness Daily Values of Demand")
plt.subplot(2,2,4)
plt.plot(dailyMomentsDemands["Kurtosis"])
plt.title("Kurtosis Daily Values of Demand")
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)

plt.figure(3)
days = 45
plt.subplot(2,2,1)
plt.plot(meanSources.sum(axis=1, numeric_only=True))
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)
plt.title("Mean Daily Values of Sources")
plt.subplot(2,2,2)
plt.plot(varSources.sum(axis=1, numeric_only=True))
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)
plt.title("Variance Daily Values of Sources")
plt.subplot(2,2,3)
plt.plot(skewSources.sum(axis=1, numeric_only=True))
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)
plt.title("Skewness Daily Values of Sources")
plt.subplot(2,2,4)
plt.plot(kurtSources.sum(axis=1, numeric_only=True))
plt.title("Kurtosis Daily Values of Sources")
plt.xticks(ticks, labels ,rotation=17,fontsize = 6)

plt.figure(7)
for i in range(8):
    plt.subplot(2,4,i+1)
    plotDuck(groups,np.random.choice(datesDemands))
plt.xlabel("Time(5min)")
plt.ylabel("Energy Units")

# create a dataframe with the mean demands and sources as
    well as datetime
df = pd.DataFrame(columns=["Demands","Sources","Datetime"])
df["Demands"] = dailyMomentsDemands["Mean"]
df["Sources"] = meanSources.loc[:,meanSources.columns!="
    Datetime"].sum(axis =1)
df["Datetime"] = meanSources["Datetime"]
```

```
344
345 clusterFrame = df.loc[:,df.columns!="Datetime"].dropna()
346
347 nrstNeighbors = NearestNeighbors().fit(clusterFrame)
348 neighDistance, neighInd = nrstNeighbors.kneighbors(
    clusterFrame)
349 sortedNeighborDistance = np.sort(neighDistance, axis=0)
350 kDistance = sortedNeighborDistance[:, 4]
351
352 # using the elbow method find the    for the dbscan
353 plt.figure(4)
354 plt.plot(kDistance)
355 plt.axhline(y=450, linewidth=1, linestyle='dashed')
356 plt.ylabel("k-NN distance")
357 plt.xlabel("Sorted observations (4th NN)")
358
359
360 #find the clusters with DBSCAN
361 # we have 2 features so samples 2*2 = 4
362 clusters = DBSCAN(eps=450, min_samples=4).fit(clusterFrame)
363
364 # scatterplot for the sources and demands of each day
365 plt.figure(5)
366 p = sns.scatterplot(data=clusterFrame, x="Sources", y="
    Demands", hue=clusters.labels_, legend="full")
367 sns.move_legend(p, "upper right", title='Clusters')
368
369
370 # find the outlouers and get the dates of the outlier dates
371 outliers = clusterFrame[clusters.labels_ == -1]
372 outlierDates = []
373
374 for i,j in zip(outliers.Demands,outliers.Sources):
375     outlierDates.append(df.loc[(abs(clusterFrame['Demands']-i
    )<10e-5) & (abs(df['Sources']-j)<10e-5)]["Datetime"].
    values[0])
376 # remove the outliers
377 newMergedDemands = mergedDemands[mergedDemands["Datetime"].
    isin(outlierDates) == False]
378 newMergedSources = mergedSources[mergedSources["Datetime"].
    isin(outlierDates) == False]
379
380 # create the dataset conaining only the renewable sources
381 fossilSources = newMergedSources[["Coal","Nuclear","Natural
    Gas", "Batteries", "Imports", "Other"]]
382 fossilSources = fossilSources.sum(axis=1)
383 renewableSources = (newMergedDemands["Current Demand"] -
    fossilSources ).to_frame().dropna()[:2*365*288]
384
```

```
# split the data into train and test set
trainSize = int(len(renewableSources) * 0.75)
testSize = int(len(renewableSources) * 0.25)
trainSet = renewableSources.iloc[:trainSize]
testSet = renewableSources.iloc[trainSize:]

# min max scale transform the data
dataScaler = MinMaxScaler()
dataScaler.fit(trainSet)
scaledTrain = dataScaler.transform(trainSet)
scaledSet = dataScaler.transform(testSet)


# create the lstm newtork
os.environ['KMP_DUPLICATE_LIB_OK'] = 'TRUE'
inputNum = 4
featuresNum = 1
generator = TimeseriesGenerator(scaledTrain, scaledTrain,
    length=inputNum, batch_size=16)

model = Sequential()
model.add(LSTM(50, activation='tanh', input_shape=(inputNum,
    featuresNum)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.summary()
model.fit(generator,epochs=15, batch_size= 16,verbose=2)
model.save("./LSTM_model")

model = models.load_model("./LSTM_model")
testPredictions = model.predict(scaledSet)
truePredictions = dataScaler.inverse_transform(
    testPredictions)
test = np.zeros(len(scaledTrain))
test = np.concatenate((test,truePredictions),axis=None)

plt.figure(6)
plt.plot(renewableSources.sum(axis=1,numeric_only=True).
    reset_index(drop=True))
plt.plot(test)
plt.xlabel("")
plt.title("Prediction and Actual values for the renewable
    energy")
plt.xlabel("Time axis(5min)",fontweight="bold")
plt.ylabel("Energy Units",fontweight="bold")
plt.ylim([-1000,50_000])
plt.legend(["Actual Value","Predicted"])
plt.show()
```

2 Άσκηση 2

Για την άσκηση αυτή η διαδικασία που ακολουθήσαμε είναι σχετικά απλή δεδομένου ότι τα περισσότερα εργαλεία και μοντέλα ήταν έτοιμα από την Python. Αρχικά κάνουμε μια βασική προ-επεξεργασία στα δεδομένα, η οποία περιλαμβάνει την αφαίρεση των λεγόμενων stop-words, δηλαδή λέξεις που χρησιμοποιούνται πολύ συχνά και δεν έχουν ιδιαίτερη σημασία στην εκπαίδευση του δικτύου μας για τα Word-Embedding, την αφαίρεση των αριθμών και των σημείων στίξης από το dataset.

Εφόσον έχει γίνει η προ-επεξεργασία των δεδομένων μας τα τροφοδοτούμε σε ένα μοντέλο Word-Embeddings και έτσι κάθε λέξη των προτάσεων που βρίσκονται στην εκάστοτε κριτική μετατρέπεται σε ένα σύνολο από αριθμούς. Για τη συγκεκριμένη υλοποίηση επιλέχθηκε να γίνεται διάσπαση της λέξης σε ένα διάνυσμα 100 στοιχείων. Στη συνέχεια, παίρνουμε τον μέσο όρο όλων των διανυσμάτων που αποτελούν την πρόταση για την κριτική και τα τοποθετούμε σε ένα πίνακα.

Ύστερα παρατηρήσαμε ότι ο πίνακας μας είχε NaN τιμές εξαιτίας της μετατροπής σε word-embeddings οπότε τα αφαιρέσαμε και δημιουργήσαμε το τελικό dataset το οποίο περιλάμβανε τις τιμές που αναπαριστούσε την κριτική καθώς και την κριτική που δόθηκε. Μετά, χωρίστηκαν τα δεδομένα σε trainset και testset και τα περάσαμε από ένα κατηγοριοποιητή RandomForest με 200 δέντρα αποφάσεων. Στην αρχή δεν έγινε κάποια επεξεργασία στις ετικέτες και λάβαμε τα παρακάτω αποτελέσματα:

$$Accuracy : 0.633$$

$$Recall : 0.633$$

$$F1_Score : 0.633$$

Στη συνέχεια κάναμε μια τροποποίηση στις ετικέτες και από αρχικά πεντε τις μετατρέψαμε σε τρεις χρησιμοποιώντας την εξής παραδοχή: $[1, 2] \longrightarrow 0$,$[3] \longrightarrow 1$, $[4, 5] \longrightarrow 2$. Και λάβαμε τα παρακάτω αποτελέσματα:

$$Accuracy : 0.803$$

$$Recall : 0.803$$

$$F1_Score : 0.803$$

Τα αποτελέσματα την πρώτη φορά που τρέξαμε τον αλγόριθμο δεν είναι ικανοποιητικά. Το 60% στην ακρίβεια για έναν random forest δεν είναι επιθυμητό. Προσπαθήσαμε να τροποποιήσουμε τις τιμές των μοντέλων με πολύ μικρή διαφορά στην απόδοση κάτι που μας έκανε να πιστεύουμε ότι το συγκεκριμένο dataset δεν ήταν τόσο καλό. Τη δεύτερη φορά τα αποτελέσματά

που πήραμε ήταν πολύ καλύτερα αλλά όχι τέλεια. Θα μπορούσαμε για λόγους πληρότητας να υλοποιήσουμε και δυαδική κατηγοριοποίηση, δηλαδή καλή και κακή κριτική αλλά δεν θα είχε και ιδαίτερο νόημα.

2.1 Βιβλιοθήκες

Οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση αυτής της άσκησης ήταν :

- **Numpy**, για αριθμητικούς υπολογισμούς με πίνακες
- **Nltk**, για συναρτήσεις επεξεργασία φυσικής γλώσσας
- **Gensim**, για το μοντέλο του word-embeddings
- **Sklearn**, για το μοντέλο του RandomForest και για τις μετρικές απόδοσης
- **String**, για συναρτήσεις που έχουν να κάνουν με συμβολοσειρές
- **Pandas**, για τον χειρισμό των δεδομένων

2.2 Κώδικας

```
1
2  import numpy as np
3  import pandas as pd
4  from matplotlib import pyplot as plt
5  import string
6  import re
7  from nltk.corpus import stopwords
8  from gensim.models import Word2Vec,utils
9  from sklearn.metrics import precision_score,recall_score,
       f1_score
10 from sklearn.ensemble import RandomForestClassifier
11 from sklearn.model_selection import train_test_split
12 df = pd.read_csv("amazon.csv")
13
14
15
16 def wordCleaning(df):
17     """
18     1. We create a new dataframe with the column "text"
19     2. We remove stopwords from the text
20     3. We remove punctuation from the text
21     4. We remove numbers from the text
```

```
    5. We remove extra spaces from the text
    6. We apply simple preprocessing to the text

    :param df: The dataframe that contains the text column
    :return cleanData: the preprocessed dataset
    """
    cleanData = pd.DataFrame({"text":df["Text"]})
    stop_words = set(stopwords.words('english'))
    punct = re.compile('[%s]' % re.escape(string.punctuation))

    cleanData["text"] = cleanData["text"].apply(lambda x : x.lower())
    cleanData["text"] = cleanData["text"].apply(lambda x : " ".join([w for w in x.split() if not w in stop_words]))
    cleanData["text"] = cleanData["text"].apply(lambda x : re.sub(r'\'[\w]+[^\S]',' ',x))
    cleanData["text"] = cleanData["text"].apply(lambda x : re.sub(r'[0-9]',"",x))
    cleanData["text"] = cleanData["text"].apply(lambda x : punct.sub(' ', x))
    cleanData["text"] = cleanData["text"].apply(lambda x : re.sub(r'\s',' ',x))
    cleanData["text"] = cleanData["text"].apply(lambda x : " ".join(x.split()))
    cleanData["text"] = cleanData["text"].apply(lambda x: utils.simple_preprocess(x))

    return cleanData

def make_feature_vec(words, model, num_features):
    """
    For each word in the review, if the word is in the model's vocabulary, add its feature vector to the
    total. Then, divide the result by the number of words to get the average

    :param words: a list of words
    :param model: the Word2Vec model we're using
    :param num_features: The number of features to be used in the model
    :return: The average of the word vectors for each word in the review.
    """

    feature_vec = np.zeros((num_features,),dtype="float32")
    # pre-initialize (for speed)
    nwords = 0.
    index2word_set = model.wv.key_to_index.keys()  # words
```

```
    known to the model
57
58      for word in words:
59          if word in index2word_set:
60              nwords = nwords + 1.
61              feature_vec = np.add(feature_vec,model.wv[word])
62
63      feature_vec = np.divide(feature_vec, nwords)
64      return feature_vec
65
66  def avg_reviews(df, model, num_features):
67      """
68      It takes a dataframe, a word2vec model, and a number of
    features, and returns a dataframe with a new
69      column called 'vectorised' which contains the word2vec
    representation of each review.
70
71      :param df: the dataframe containing the text
72      :param model: the word2vec model
73      :param num_features: The number of features to be used in
     the model
74      :return: A dataframe with the vectorised text
75      """
76      new = pd.DataFrame({"vectorised":df['text']})
77
78      new['vectorised'] = new['vectorised'].apply(lambda x:
    make_feature_vec(x, model, num_features))
79
80      return new
81
82  # read the dataset and apply cleaning
83  df = pd.read_csv("amazon.csv")
84  cleanDF = wordCleaning(df)
85
86  # create a word embeddings model that will vectorize each
    word into float numbers
87  vec_size = 100
88  model = Word2Vec(vector_size=vec_size,window= 15, min_count=
    2, workers=6)
89  model.build_vocab(cleanDF["text"], progress_per=1000)
90  model.train(cleanDF["text"], total_examples=model.
    corpus_count, epochs=model.epochs)
91
92  # for every word in a sentence get the average and put them
    in a dataframe
93  vectored = avg_reviews(cleanDF,model, vec_size)
94  vectored['vecorised'] = vectored['vectorised'].apply(lambda x
    : x.astype(np.float128))
95
```

```
train_Frame = pd.DataFrame(vectored['vectorised'].tolist())
# remove all the nan indexes
nanIndexes = train_Frame[train_Frame.isna().any(axis=1)].
    index
train_Frame.drop(index=nanIndexes,inplace=True)
df.drop(index=nanIndexes, inplace=True)

#split the data into train test set
X_train, X_test, y_train, y_test = train_test_split(
    train_Frame, df["Score"], test_size=0.2, random_state=1)

#create a random forest algorithm and run the model
randomForest = RandomForestClassifier(n_estimators=200,
    criterion='gini',n_jobs=4)
randomForest.fit(X_train, y_train)

y_pred = randomForest.predict(X_test)
print("--------Results Using The Initial Classes--------")
print('Accuracy: %.3f' % precision_score(y_test, y_pred,
    average='micro'))
print('Recall %.3f' % recall_score(y_test,y_pred, average='
    micro'))
print('F1_Score %.3f' % f1_score(y_test,y_pred, average='
    micro'))


df.loc[df["Score"] == 1,"Score"] = 0
df.loc[df["Score"] == 2,"Score"] = 0
df.loc[df["Score"] == 4,"Score"] = 2
df.loc[df["Score"] == 5,"Score"] = 2
df.loc[df["Score"] == 3,"Score"] = 1

X_train, X_test, y_train, y_test = train_test_split(
    train_Frame, df["Score"], test_size=0.2, random_state=1)

randomForest = RandomForestClassifier(n_estimators=200,
    criterion='gini',n_jobs=4)
randomForest.fit(X_train, y_train)

y_pred = randomForest.predict(X_test)
print("--------Results Using Three Classes--------")
print('Accuracy: %.3f' % precision_score(y_test, y_pred,
    average='micro'))
print('Recall %.3f' % recall_score(y_test,y_pred, average='
    micro'))
print('F1_Score %.3f' % f1_score(y_test,y_pred, average='
    micro'))
```