ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

DEPARTMENT OF INFORMATICS

8/6/2023

# Ανάλυση Κατανάλωσης Ηλεκτρικού Ρεύματος στην Πορτογαλία

*Θέματα Επιστήμης Δεδομένων: Απαλλακτική Εργασία 2023*

*Γεώργιος Μπαλτζάκης*
*ΑΜ: Π18105*
*email: george-baltzakis@hotmail.gr*

# Πίνακας περιεχομένων

# Εισαγωγή

Η συγκεκριμένη εργασία έχει στόχο την ανάλυση της κατανάλωσης ηλεκτρικού ρεύματος στην Πορτογαλία. Η ανάλυση κατανάλωσης ρεύματος αποτελεί ένα σημαντικό εργαλείο για την κατανόηση της χρήσης ενέργειας και την αναγνώριση τρόπων βελτίωσης της ενεργειακής απόδοσης. Μέσω της ανάλυσης κατανάλωσης ρεύματος, μπορούμε να αναλύσουμε τα δεδομένα που συλλέγονται από μετρητές ηλεκτρικής ενέργειας και να προσδιορίσουμε τα μοτίβα κατανάλωσης, τις ανισορροπίες και τις πιθανές αιτίες των ενεργειακών απωλειών. Οι ενότητες που υλοποιήθηκαν είναι:

- Εισαγωγή των δεδομένων
- Προετοιμασία δεδομένων:
    - Καθαρισμός
    - Αρχική Οπτικοποίηση
    - Ανάλυση των τάσεων
- Υλοποίηση Αλγορίθμων Συσταδοποίησης:
    - Μείωση διαστάσεων (Εφαρμογή Αλγορίθμου PCA)
    - Εφαρμογή και αποτελέσματα K-Means
    - Εφαρμογή και αποτελέσματα DBSCAN
- Υλοποίηση Μοντέλων Πρόβλεψης:
    - Επιλογή και Προετοιμασία δεδομένων
    - Ανάλυση των τάσεων
    - Κανονικοποίηση και Detrending
    - Επαύξηση δεδομένων (Overlapping Windows)
    - Διαχωρισμός δεδομένων σε σύνολα:
        - Εκπαίδευσης
        - Επικύρωσης
        - Δοκιμής
    - Ορισμός των μορφών Εισόδου & Εξόδου
    - Υλοποίηση και Αποτελέσματα του μοντέλου Random Forest
    - Υλοποίηση και Αποτελέσματα του μοντέλου LSTM

Για αυτή την ανάλυση χρησιμοποιήθηκε το σύνολο δεδομένων *Electricity Load Diagrams*[1] του UCI, το οποίο αποτελείται από 370 χρονοσειρές κατανάλωσης ηλεκτρικού ρεύματος που συλλέχθηκαν από πελάτες ενός παρόχου ενέργειας στην Πορτογαλία κατά το χρονικό διάστημα 2011-2014.

Τα εργαλεία που πλαισίωσαν αυτό το project περιβάλλονται από την προγραμματιστική γλώσσα *Python*.

[1] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

# Περιγραφή Συνόλου Δεδομένων

Το συγκεκριμένο dataset αποτελείται από 370 χρονοσειρές κατανάλωσης ηλεκτρικού ρεύματος που συλλέχθηκαν από πελάτες ενός παρόχου ενέργειας στην Πορτογαλία κατά το χρονικό διάστημα 2011-2014. Αυτό το ευρύ χρονικό παράθυρο παρέχει μια ολοκληρωμένη εικόνα της κατανάλωσης ρεύματος κατά τη διάρκεια τετραετούς περιόδου.

Κάθε χρονοσειρά αποτελείται από μετρήσεις κατανάλωσης ηλεκτρικού ρεύματος σε συγκεκριμένα χρονικά διαστήματα, πιο συγκεκριμένα, δεκαπέντε λεπτών. Αυτό μας επιτρέπει να εξετάσουμε τις διακυμάνσεις στην κατανάλωση ρεύματος κατά τη διάρκεια της ημέρας, της εβδομάδας και των εποχών του έτους.

Μελετώντας αυτό το dataset, μπορούμε να αναγνωρίσουμε τάσεις και πρότυπα στην κατανάλωση ενέργειας, όπως η αύξηση ή μείωση της κατανάλωσης κατά τη διάρκεια συγκεκριμένων περιόδων ή η επίδραση εξωτερικών παραγόντων, όπως ο καιρός ή η ημέρα της εβδομάδας, στην κατανάλωση. Η ανάλυση αυτού του συνόλου δεδομένων μπορεί να οδηγήσει σε σημαντικές ευκαιρίες για τη βελτίωση της ενεργειακής απόδοσης και την υιοθέτηση βέλτιστων πρακτικών για την εξοικονόμηση ενέργειας.

Αυτό το dataset ανοίγει τον δρόμο για περαιτέρω έρευνα και ανάλυση στον τομέα της κατανάλωσης ρεύματος, και παρέχει ένα πλούσιο υλικό για την κατανόηση των προτύπων και των παραγόντων που επηρεάζουν την κατανάλωση ενέργειας σε μια πολυπληθή πληθυσμιακή ομάδα.

# Εργαλεία Υλοποίησης

Τα προγραμματιστικά εργαλεία παίζουν ένα καθοριστικό ρόλο στην ανάπτυξη και την εκτέλεση αναλύσεων δεδομένων και μοντέλων μηχανικής μάθησης. Ανάμεσα σε αυτά τα εργαλεία είναι:

- Pandas
- NumPy,
- Matplotlib,
- Plotly,
- StatsModels,
- sklearn,
- tslearn,
- SciPy
- TensorFlow.

# Εισαγωγή & Προετοιμασία Δεδομένων

Αρχικά γίνεται Εισαγωγή των Δεδομένων

```
# Import Data
data = pd.read_csv("../Data/LD2011_2014.txt", delimiter=";", low_memory=False)
data.to_csv("../Data/LD2011_2014.csv", index=None)
```

Έπειτα γίνονται ενέργειες έτσι ώστε να «έρθουν» τα δεδομένα σε μία δομημένη μορφή. Αμέσως μετά, διαγράφουμε τις διπλότυπες εγγραφές βάσει του χαρακτηριστικού *datetime*.

```
# Data Preparation
data.rename(columns={'Unnamed: 0':"datetime"}, inplace=True)
# First Column: str -> datatime
data['datetime'] = pd.to_datetime(data['datetime'])

# All other columns: str -> float
data = data.replace(",", ".", regex=True)
data[data.columns[1:]] = data[data.columns[1:]].astype('float')

# Drop Duplicates
data.drop_duplicates(subset=['datetime'], inplace=True)
```

Συνεχίζοντας ψάχνουμε για άκυρες μεταβλητές, δηλαδή για αρνητική κατανάλωση ρεύματος και κενές τιμές. Μετά μετασχηματίζουμε τις τιμές κατανάλωσης: kW → kWh. Τέλος αποθηκεύουμε το dataset.

```
# Search for values < 0
if (True in (data[data.columns[1:]].values < 0)):
    print('There are values less than zero')
else:
    print('Only positive and zero values')

# Search for null values
if (data.isnull().values.any()):
    print('There are null values.')
else:
    print('There are no null values.')

# Convert kW to kWh
data[data.columns[1:]] = data[data.columns[1:]]/4

data.to_csv("../Data/AP_LD2011_2014.csv", index=False)
```

[2] ✓ 0.0s

Αποτέλεσμα:

```
data.tail()
```

[63] ✓ 0.0s

| | datetime | MT_001 | MT_002 | MT_003 | MT_004 | MT_005 | MT_006 | |
|---|---|---|---|---|---|---|---|---|
| 140251 | 2014-12-31 23:00:00 | 0.634518 | 5.512091 | 0.434405 | 37.601626 | 21.341463 | 75.892857 | 2 |
| 140252 | 2014-12-31 23:15:00 | 0.634518 | 5.334282 | 0.434405 | 41.666667 | 20.426829 | 81.101190 | 2 |
| 140253 | 2014-12-31 23:30:00 | 0.634518 | 5.156472 | 0.434405 | 40.650407 | 20.731707 | 79.613095 | 2 |
| 140254 | 2014-12-31 23:45:00 | 0.317259 | 5.334282 | 0.434405 | 41.666667 | 21.341463 | 71.428571 | 2 |
| 140255 | 2015-01-01 00:00:00 | 0.634518 | 4.978663 | 0.434405 | 44.715447 | 21.036585 | 69.940476 | 2 |

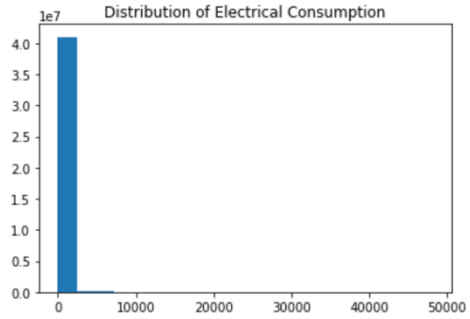
5 rows × 371 columns

## Οπτικοποίηση και Κατανόηση των Δεδομένων

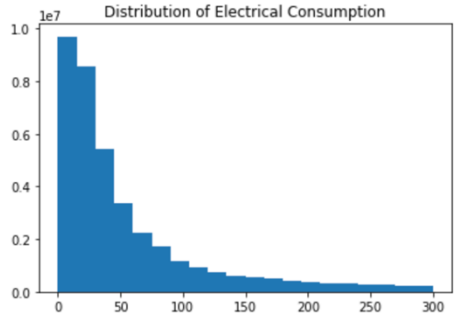
Στα παρόν διαγράμματα παρουσιάζεται η κατανομή των τιμών κατανάλωσης όπου φαίνεται ότι ακολουθεί την κατανομή:

$$\frac{1}{x} \text{ ,όπου } x > 0$$

```
Entire Dataset
```

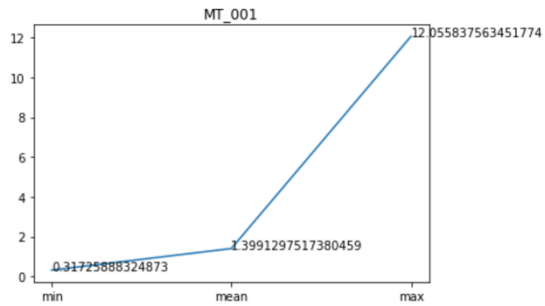


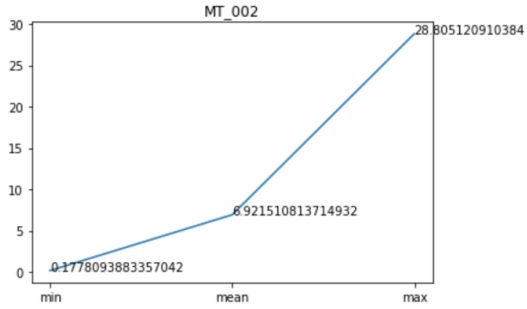

```
Values between 0 and 300
```

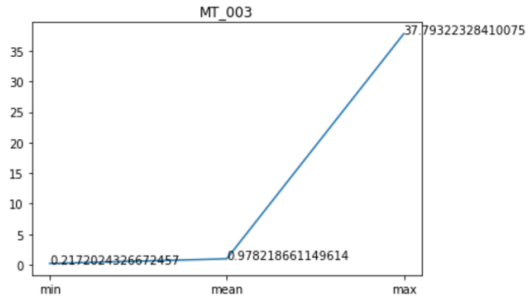



```
So schematically we understand that the distribution of values follows the function 1÷x, where x > 0
```

Ηλεκτρική κατανάλωση ανά πελάτη (kWh): min | mean | max

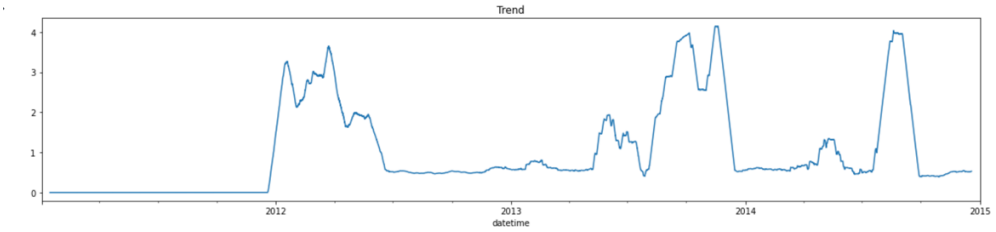


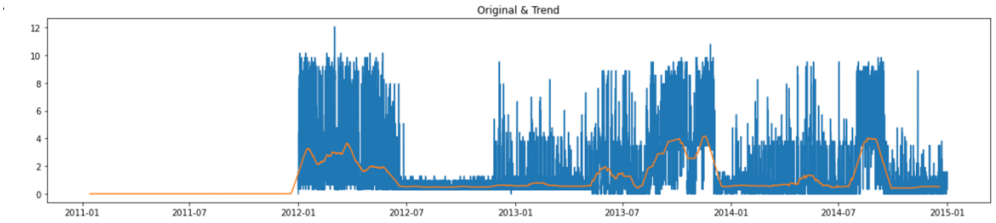



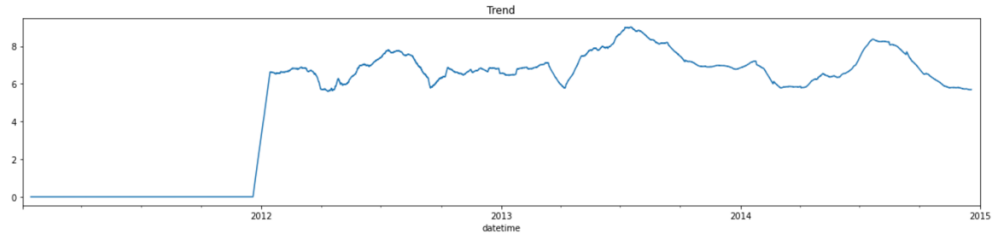
## Ανάλυση τάσεων κατανάλωσης ανά πελάτη

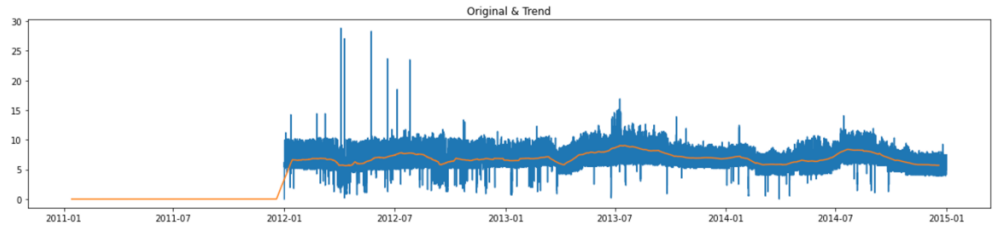
MT_001



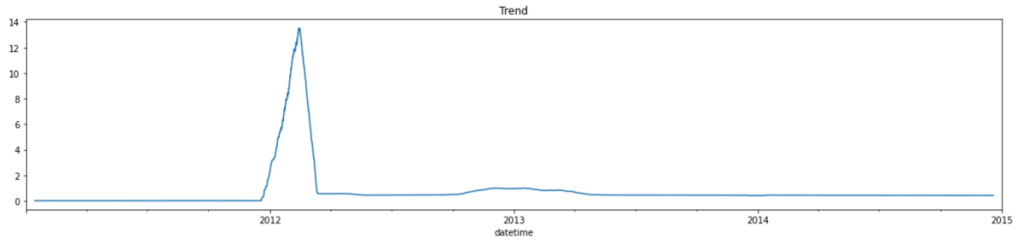



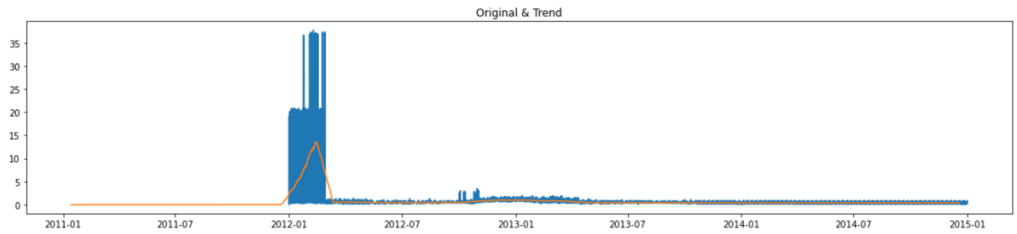

MT_002




MT_003

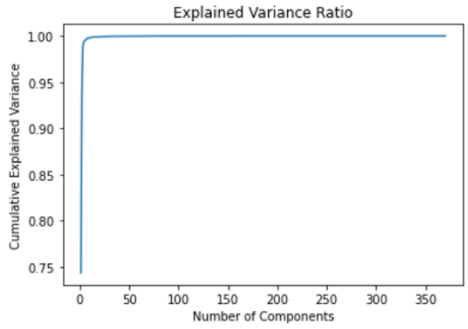


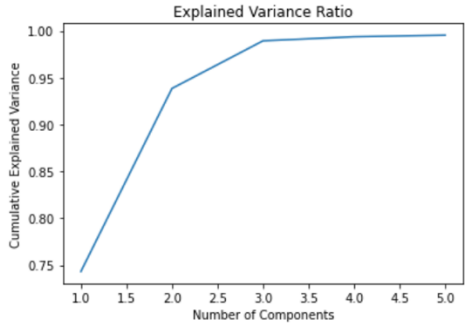
# Συσταδοποίηση

## Αλγόριθμος μείωσης διαστάσεων (PCA)

Αρχικά εξετάζουμε ποιό είναι το ιδανικό πλήθος διαστάσεων. Και καταλήγουμε ότι είναι οι τρεις διαστάσεις.




Closer Look




Αποτέλεσμα

```
pca_result.head()
```

✓ 0.0s

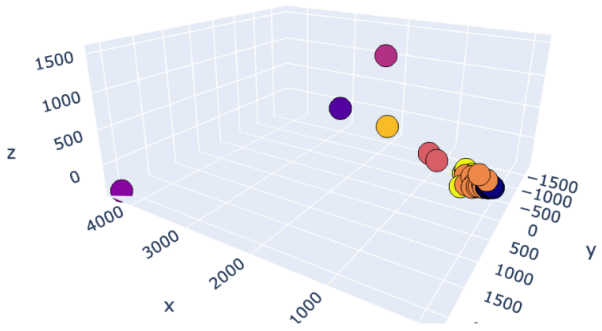
| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | -78.241293 | 14.899625 | -4.968907 |
| 1 | -75.507412 | 15.445048 | -4.167743 |
| 2 | -78.290995 | 14.864254 | -5.414635 |
| 3 | -65.479330 | 18.151775 | -1.126192 |
| 4 | -72.537530 | 16.235943 | -3.276934 |

## Εφαρμογή Αλγορίθμου K-Means με k = 8

# K-Means (with k = 8)

```
[10]  kmeans_8 = TimeSeriesKMeans(n_clusters=8, metric="dtw", max_iter=10)
      kmeans_8.fit(pca_result)
```

✓ 4.3s

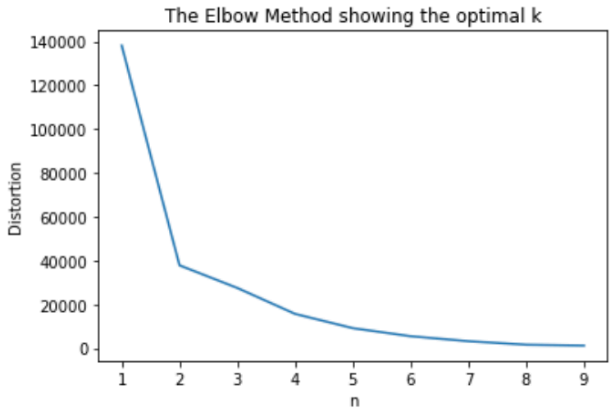


## Elbow Method

### K-Means: Elbow Method

```
distortions = []
n = range(1, 10)
for k in n:
    kmeans_elbow = TimeSeriesKMeans(n_clusters=k, metric="dtw", max_iter=10)
    kmeans_elbow.fit(pca_result)
    distortions.append(kmeans_elbow.inertia_)

plt.plot(n, distortions)
plt.xlabel('n')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

[12] ✓ 12.3s



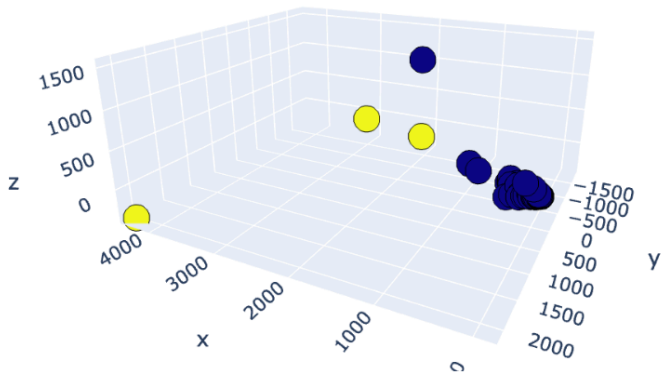

## Εφαρμογή Αλγορίθμου K-Means με k = 2

### K-Means with optimal number of clusters (k = 2)

```
kmeans_2 = TimeSeriesKMeans(n_clusters=2, metric="dtw", max_iter=10)
kmeans_2.fit(pca_result)
```

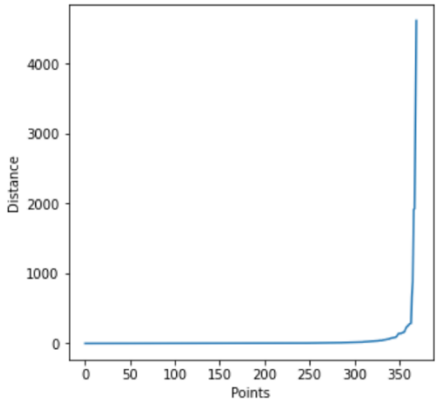
[3] ✓ 0.7s

```
TimeSeriesKMeans(max_iter=10, metric='dtw', n_clusters=2)
```

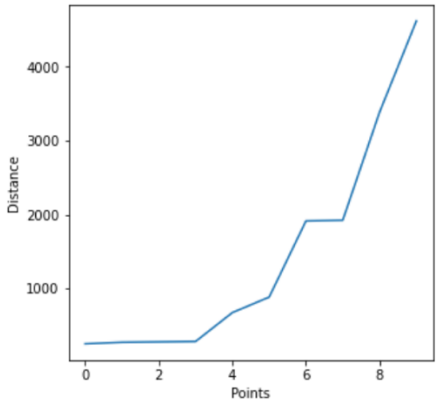


## Εφαρμογή Αλγορίθμου DBSCAN

Εύρεση κατάλληλης τιμής για την παράμετρο $\varepsilon$.




```
Closer Look
```



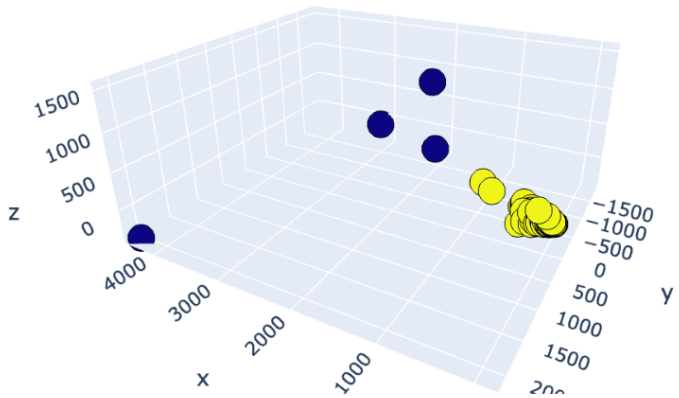

```
Optimal value of e: distances[365] -> 886
```

Εφαρμογή

```
db = DBSCAN(eps=886, min_samples=8)
db.fit(pca_result.values)
```

[16] ✓ 0.0s

```
DBSCAN(eps=886, min_samples=8)
```




## Υλοποίηση Μοντέλων Πρόβλεψης

Επιλογή της συστάδας με τους λιγότερους πελάτες

Which Cluster of the K-Means (k = 2)?

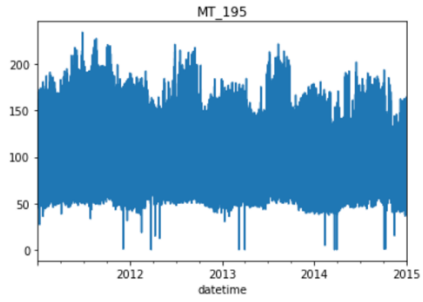
```
print('Unique values: ')
print(set(kmeans_2.labels_))
print('\nCluster 0: ')
print(list(kmeans_2.labels_).count(0), 'clients'
print('\nCluster 1: ')
print(list(kmeans_2.labels_).count(1), 'clients'
print('\nSo, pick Cluster 1')
```

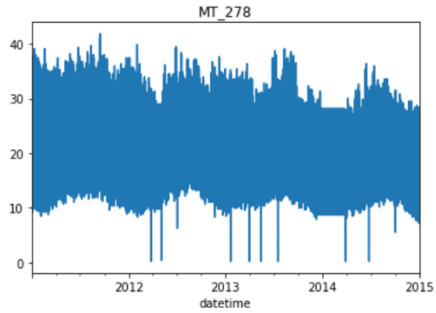
[18]

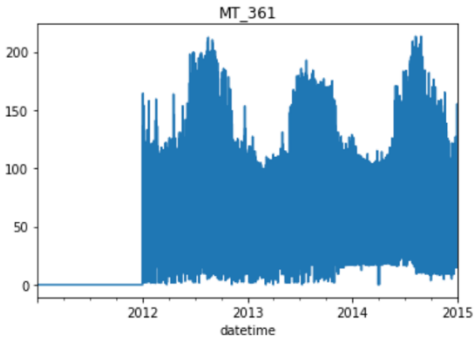
```
Unique values:
{0, 1}

Cluster 0:
367 clients

Cluster 1:
3 clients

So, pick Cluster 1
```

| | MT_195 | MT_278 | MT_361 |
|---|---|---|---|
| **datetime** | | | |
| 2011-01-01 00:15:00 | 54.568528 | 12.349154 | 0.0 |
| 2011-01-01 00:30:00 | 55.837563 | 12.717071 | 0.0 |
| 2011-01-01 00:45:00 | 55.837563 | 12.349154 | 0.0 |
| 2011-01-01 01:00:00 | 56.260575 | 11.981236 | 0.0 |
| 2011-01-01 01:15:00 | 56.260575 | 11.797277 | 0.0 |

## Οπτικοποίηση Τιμών Κατανάλωσης



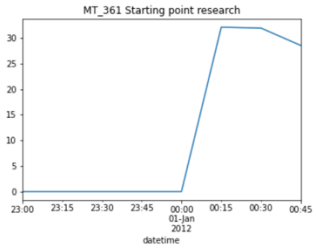

## Επεξεργασία Δεδομένων

Διαγραφή όλων των τιμών πριν από 00:15:00 01-01-2012, διότι τότε ο πελάτης «MT_361» έχει μηδενικές τιμές.

Removing the datetimes before MT_361 start the consumption

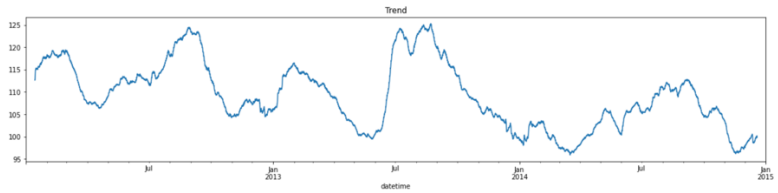
```
# Starting point research of MT_361 customer
unique_customers['MT_361']['2011-12-31 23:00:00':'2012-01-01 00:45:00'].plot()
plt.title('MT_361 Starting point research')
plt.show()

# Select prices after 2012-01-01 00:00:00
unique_customers = unique_customers['2012-01-01 00:15:00':]

# Plot Customer Consumption
for i in unique_customers.columns.values:
    unique_customers[i].plot()
    plt.title(i)
    plt.show()
```

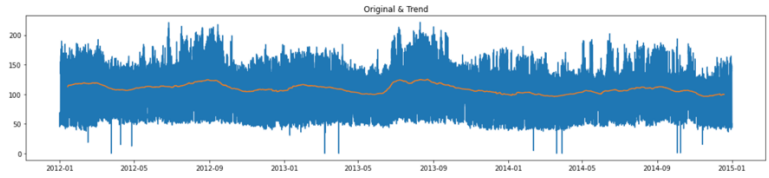
[21]



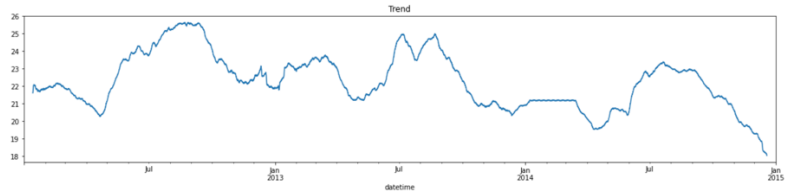
## Ανάλυση Τάσεων

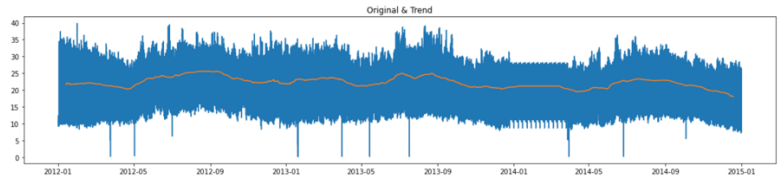
MT_195



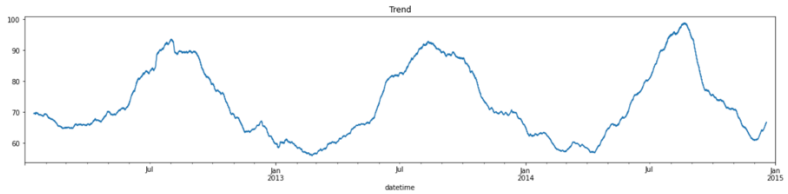



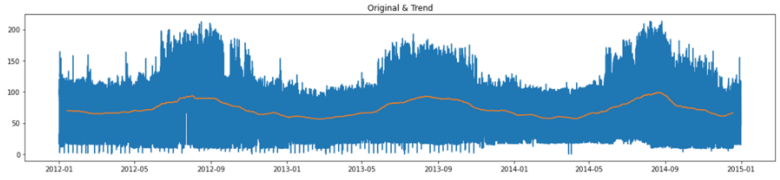

MT_278




MT_361



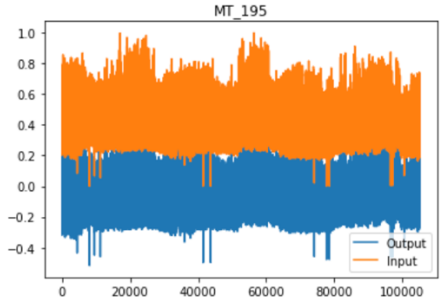



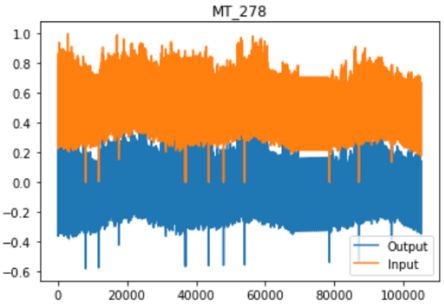

## Κανονικοποίηση Δεδομένων (Min-Max Scaler)

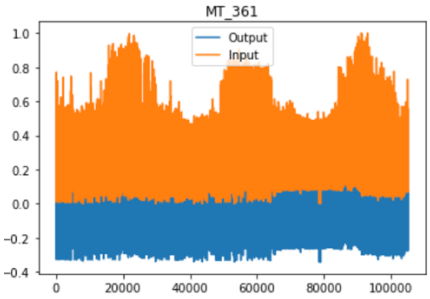
Normalization (min-max)

```
[23]
unique_customers_max = unique_customers.max()
unique_customers_min = unique_customers.min()

norm_unique_customers = (unique_customers-unique_customers.min())/(unique_customers.max()-unique_customers.min())
```

## Detrending




## Επαύξηση Δεδομένων (Αλγόριθμος → Overlapping Windows)

### Time Series Augmentation | Overlapping Windows

```
data_for_augmentation = detrend_unique_customers.copy()
data_for_augmentation.reset_index(inplace=True)

def data_augmentation(data_aug, window_size, overlap):
    num_windows = (len(data_aug) - window_size) // overlap + 1

    # Create a list to store the windows
    windows = []

    # Generate overlapping windows
    for i in range(num_windows):
        start_index = i * overlap
        end_index = start_index + window_size
        window = data_aug.iloc[start_index:end_index]
        windows.append(window)

    output=pd.DataFrame()
    # Concat the generated windows
    for window in windows:
        output = pd.concat([output, window], ignore_index=True)

    output.sort_values("datetime", inplace=True)

    return output

data_after_augmentation = data_augmentation(data_for_augmentation, 960, 96)
data_after_augmentation.set_index("datetime", inplace=True)

print("Data before Augmentation:", detrend_unique_customers.shape)
print("Data after Augmentation:", data_after_augmentation.shape)

data_for_prediction = data_after_augmentation.copy()
```

[25]

```
Data before Augmentation: (105216, 3)
Data after Augmentation: (1043520, 3)
```

## Διαχωρισμός Δεδομένων

### Split Data

- Train 70%
- Validation 15%
- Test 15%

```
[26]
# Split the data into train, validation, and test sets
train_size = 0.7
val_size = 0.15
test_size = 0.15

# Calculate the number of samples for each set based on the ratios
num_samples = data_for_prediction.shape[0]
num_train = int(num_samples * train_size)
num_val = int(num_samples * val_size)
num_test = int(num_samples * test_size)

# Split the data based on the shuffled indices
train_data = data_for_prediction.iloc[:num_train]
val_data = data_for_prediction.iloc[num_train:num_train + num_val]
test_data = data_for_prediction.iloc [num_train + num_val:]

print('Total Rows:\t\t', num_samples, '\n')
print('Train Data:\t\t', train_data.shape)
print('Validation Data:\t', val_data.shape)
print('Test Data shape:\t', test_data.shape)
```

```
Total Rows:             1043520

Train Data:             (730464, 3)
Validation Data:        (156528, 3)
Test Data shape:        (156528, 3)
```

# Ορισμός Εισόδου & Εξόδου

## Define Input & Output

- Input: 1h data (4 * 15m)
- Output: the next 15m

```
# Define the number of time steps and features
n_steps = 4  # Number of time steps to consider

# Generate sequences of input data
def generate_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        X.append(data.iloc[i:i + n_steps].values.T)
        y.append(data.iloc[i + n_steps].values)
    return np.array(X).astype(np.float32), np.array(y).astype(np.float32)

X_train, y_train = generate_sequences(train_data, n_steps)
X_val, y_val = generate_sequences(val_data, n_steps)
X_test, y_test = generate_sequences(test_data, n_steps)

# 3D to 2D for Random Forest
X_train_rm = X_train.reshape(X_train.shape[0], -1)
X_val_rm = X_val.reshape(X_val.shape[0], -1)
X_test_rm = X_test.reshape(X_test.shape[0], -1)
```

```
print('Input:', X_train_rm.shape)
print(X_train_rm[:3])
print('\n----------------------------------------\n')
print('Output:', y_train.shape)
print(y_train[:3])
```

```
Input: (730460, 12)
[[-0.28862578 -0.27908322 -0.26572388 -0.2924409  -0.336081   -0.3128936
  -0.32680508 -0.33144182 -0.18598679 -0.18682368 -0.20272335 -0.2052339 ]
 [-0.27908322 -0.26572388 -0.2924409  -0.24091361 -0.3128936  -0.32680508
  -0.33144182 -0.33607858 -0.18682368 -0.20272335 -0.2052339  -0.20439716]
 [-0.26572388 -0.2924409  -0.24091361 -0.29816496 -0.32680508 -0.33144182
  -0.33607858 -0.33144063 -0.20272335 -0.2052339  -0.20439716 -0.20439725]]

----------------------------------------

Output: (730460, 3)
[[-0.24091361 -0.33607858 -0.20439716]
 [-0.29816496 -0.33144063 -0.20439725]
 [-0.28480563 -0.33144003 -0.20439732]]
```

# Υλοποίηση Μοντέλου Random Forest

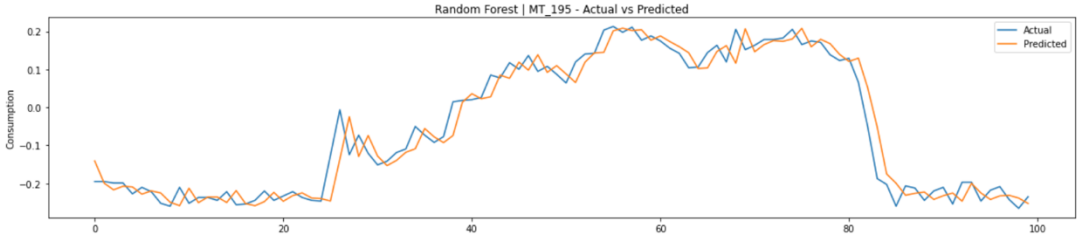
## Random Forest

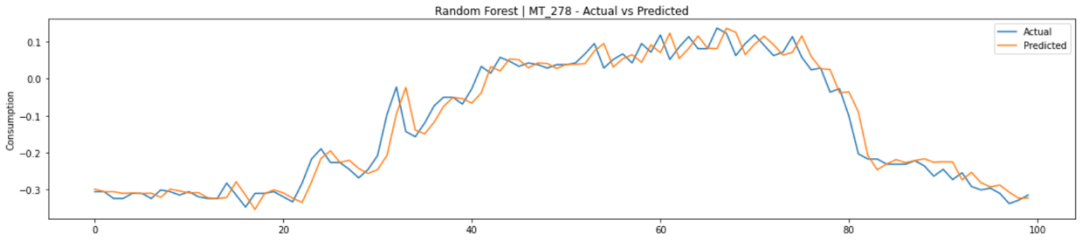
```
# Import the model_rm
model_rm = joblib.load('../Data/model_rm.pkl')
```
[29]

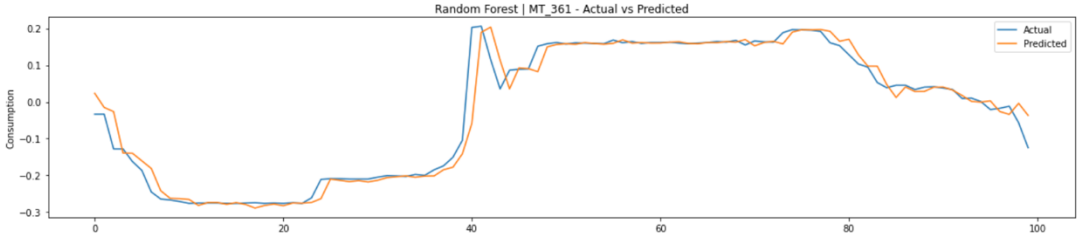
```
# Create and train the Random Forest model
model_rm = RandomForestRegressor(n_estimators=100, random_state=42)
model_rm.fit(X_train_rm, y_train)
```
[30]

```
# Make predictions on the validation set
y_val_pred = model_rm.predict(X_val_rm)

# Evaluate the model
mse = mean_squared_error(y_val, y_val_pred)
print(f"Mean Squared Error on the validation set:", mse)

# Make predictions on the test set
y_test_pred = model_rm.predict(X_test_rm)

# Evaluate the model on the test set
mse_test = mean_squared_error(y_test, y_test_pred)
print(f"Mean Squared Error on the test set:", mse_test)
```
[31]

```
Mean Squared Error on the validation set: 0.00017425707790254358
Mean Squared Error on the test set: 0.0002320805657098409
```



## Προετοιμασία για το μοντέλο LSTM

### LSTM

```
[36]
y_train_lstm = y_train.reshape((y_train.shape[0], 3, 1))
y_val_lstm = y_val.reshape((y_val.shape[0], 3, 1))
y_test_lstm = y_test.reshape((y_test.shape[0], 3, 1))
```

```
[37]
print('Input:', X_train.shape)
print(X_train[:3])
print('\n-----------------------------------------\n')
print('Output:', y_train_lstm.shape)
print(y_train_lstm[:3])
```

```
Input: (730460, 3, 4)
[[[-0.28862578 -0.27908322 -0.26572388 -0.2924409 ]
  [-0.336081   -0.3128936  -0.32680508 -0.33144182]
  [-0.18598679 -0.18682368 -0.20272335 -0.2052339 ]]

 [[-0.27908322 -0.26572388 -0.2924409  -0.24091361]
  [-0.3128936  -0.32680508 -0.33144182 -0.33607858]
  [-0.18682368 -0.20272335 -0.2052339  -0.20439716]]

 [[-0.26572388 -0.2924409  -0.24091361 -0.29816496]
  [-0.32680508 -0.33144182 -0.33607858 -0.33144063]
  [-0.20272335 -0.2052339  -0.20439716 -0.20439725]]]

-----------------------------------------

Output: (730460, 3, 1)
[[[-0.24091361]
  [-0.33607858]
  [-0.20439716]]

 [[-0.29816496]
  [-0.33144063]
  [-0.20439725]]

 [[-0.28480563]
  [-0.33144003]
  [-0.20439732]]]
```

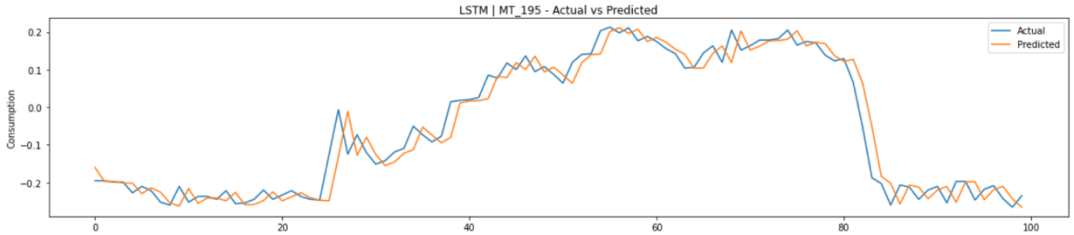
# Υλοποίηση Μοντέλου LSTM

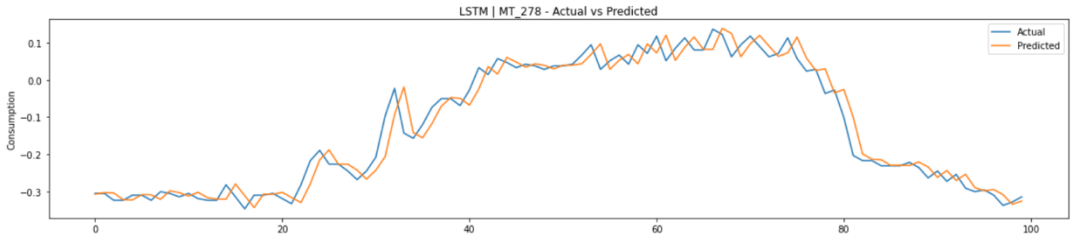
```
# Build the LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(16, activation='sigmoid', input_shape=(3, 4)))
model_lstm.add(Dense(3))
model_lstm.compile(optimizer='adam', loss='mse')
```
[38]

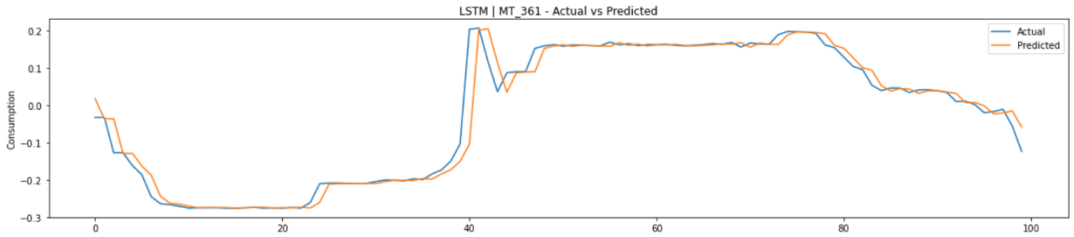
```
history = model_lstm.fit(X_train, y_train_lstm, epochs=20, batch_size=500,
                         validation_data=(X_val, y_val_lstm))
```
[39]

```
# Make predictions
predictions = model_lstm.predict(X_test)

lstm_mse = mean_squared_error(y_test_lstm.reshape((y_test_lstm.shape[0], -1)), predictions.reshape((predictions.shape[0], -1)))
# Print the mean squared error
print('Mean Squared Error:', lstm_mse)
```
[40]

```
2023-06-06 15:57:02.326805: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are e
Mean Squared Error: 0.00019785274
```



# Συμπεράσματα

Η παρόν υλοποίηση κατέληξε να αποδώσει ένα αρκετά μικρό μέσο τετραγωνικό σφάλμα ειδικά για το LSTM μοντέλο. Ωστόσο η είσοδος και η έξοδος που έχουν αυτά τα δύο μοντέλα είναι αρκετά βραχυπρόθεσμα. Θα είχε νόημα να γίνουν περισσότερες υλοποιήσεις με στόχο την αποδοτικότερη αλλά και χρήσιμη ανάπτυξη αυτών των προβλέψεων, βάσει των μορφών εισόδου και εξόδου.

Για παράδειγμα, παραπάνω, η είσοδος είναι οι τιμές κατανάλωσης μίας ώρας και η έξοδος τα επόμενα δεκαπέντε λεπτά αυτής. Θα μπορούσε να δοκιμαστεί ως είσοδος ένα εύρος χρονοσειρών του ενός τριμήνου και να προβλεφθούν οι τιμές κατανάλωσης του επόμενου μήνα. Αυτές οι υλοποιήσεις θα είναι πιο χρήσιμες σε μία τέτοια ανάλυση, ωστόσο απαιτούν και πολύ παραπάνω υπολογιστική ισχύ.