ΟΝΟΜΑΤΕΠΩΝΥΜΟ Χρήστος Κρατημένος

ΑΜ 1089892

ΕΤΟΣ ΦΟΙΤΗΣΗΣ 1ο

Ανάλυση δεδομένων κατανάλωσης φορτίου Ελληνικού Δικτύου Ηλεκτρικής Ενέργειας

Περίληψη: Στην εργασία με θέμα «Ανάλυση δεδομένων κατανάλωσης φορτίου Ελληνικού Δικτύου Ηλεκτρικής Ενέργειας» ανέπτυξα ένα πρόγραμμα σε γλώσσα προγραμματισμού python. Στο εν λόγω πρόγραμμα ο χρήστης εισάγει μία ή πολλές ημερομηνίες και το πρόγραμμα υπολογίζει τα στατιστικά στοιχεία του ηλεκτρικού φορτίου της Ελλάδας και τα παρουσιάζει στον χρήστη γραφικά. Επίσης δίνεται η δυνατότητα στον χρήστη να συγκρίνει το ηλεκτρικό φορτίο/ηλεκτρική κατανάλωση της Ελλάδας με μια άλλη ευρωπαϊκή χώρα για μια δεδομένη μέρα. Επίσης, στην εργασία έχω συντάξει μια έκθεση που εξηγεί την ανάγκη και αξία σχετικά με την σωστή πρόβλεψη της ισχύος του ηλεκτρικού δικτύου.

I. ΕΙΣΑΓΩΓΗ

Η πρόβλεψη του ηλεκτρικού φορτίου είναι μια τεχνική που χρησιμοποιείται από εταιρίες διανομής ηλεκτρικής ενέργειας ώστε να προβλέψουν την ισχύ/ενέργεια που χρειάζεται το δίκτυο ενέργειας, ώστε να εξισορροπηθεί η ζήτηση με την προσφορά. Η πρόβλεψη αυτή συνιστά τον ακρογωνιαίο λίθο του συστήματος ηλεκτρικής ενέργειας. Οι εταιρείες διανομής ηλεκτρικής ενέργειας έχουν ιδιαίτερη ανάγκη αυτήν την πρόβλεψη καθώς σύμφωνα με τα αποτελέσματα της λαμβάνουν αποφάσεις τόσο για την καθημερινή τους λειτουργία, αλλά και για τον μακροχρόνιο σχεδιασμό του δικτύου.

II. ΑΝΑΠΤΥΞΗ ΘΕΜΑΤΟΣ

Η εξαιρετική δυσκολία αποθήκευσης ηλεκτρικής ενέργειας σε μεγάλη κλίμακα καθιστά την πρόβλεψη του φορτίου αναγκαία. Η προβλέψεις του φορτίου χωρίζονται σε 3 κατηγορίες [1] :

1. **Βραχυπρόθεσμες προβλέψεις ηλεκτρικού φορτίου**: αυτή η μέθοδος προβλέψεων είναι περιόδου από μία ώρα έως μια εβδομάδα. Προβλέψεις τέτοιου είδους χρησιμοποιούνται για την προσέγγιση της ροής του φορτίου για μια δεδομένη στιγμή και προσφέρουν σε πάροχους ηλεκτρικής ενέργειας την δυνατότητα να ελέγχουν την προσφορά της ενέργειας δεδομένου της προσδοκώμενης ζήτησης. Οι μέθοδοι με τους οποίους γίνονται προβλέψεις είναι ανάλυση παλινδρόμησης , fuzzy logic , έμπειρα συστήματα, αλγόριθμοι μηχανικής μάθησης, νευρωνικά δίκτυα και προσεγγίσεις παρόμοιας ημέρας[2].
2. **Μεσοπρόθεσμες προβλέψεις ηλεκτρικού φορτίου**: αυτή η μέθοδος προβλέψεων είναι περιόδου από μια εβδομάδα μέχρι ένα χρόνο. Προβλέψεις τέτοιου είδους έχουν σκοπό την διαχείριση και τον προγραμματισμό της χρήσης καυσίμων και των υποδομών της μεταφοράς ενέργειας. Τα μοντέλα που χρησιμοποιούνται για αυτή την πρόβλεψη ARIMA, Wavelet-ARIMA and Machine Learning, Inter Taylor Model Algorithm και πολλά άλλα. Επίσης οικονομετρικά μοντέλα αξιοποιούνται για την πρόβλεψη της ζήτησης ενέργειας. Παρόμοια μοντέλα αξιοποιούνται και για το επόμενο είδος προβλέψεων[1][3].
3. **Μακροπρόθεσμες προβλέψεις ηλεκτρικού φορτίου**: αυτή η μέθοδος προβλέψεων αναφέρεται σε προβλέψεις μεγαλύτερες από ένα χρόνο. Η κύρια χρήσης τέτοιας πρόβλεψης για τις εταιρείες διανομής ηλεκτρικής ενέργειας είναι να δοθεί μια ακριβή πρόβλεψη για τις ανάγκες επέκτασης του δικτύου, για την αγορά και εγκατάσταση εξοπλισμού (π.χ. μετασχηματιστές, εφεδρικά συστήματα παραγωγής ενέργειας, μηχανισμοί προστασίας) καθώς και της παραγωγής.

Τα πλεονεκτήματα ενός συστήματος ακριβούς πρόβλεψης είναι κυρίως η δυνατότητα όσο το δυνατόν να αποφευχθεί η υπερβολική παραγωγή αλλά και υποπαραγωγή ηλεκτρικής ενέργειας[1]. Κατά συνέπεια, οι φυσικοί μας πόροι χρησιμοποιούνται πιο αποτελεσματικά περιορίζοντας το περιβαλλοντικό μας αντίκτυπο. Η πρόβλεψη φορτίου βοηθά στον μακροχρόνιο σχεδιασμό ως προς το μέγεθος, την τοποθεσία και το είδος των μελλοντικών σταθμών διανομής (και παραγωγής) ενέργειας. Με την ταυτοποίηση περιοχών με μεγάλης ή αυξανόμενης ζήτησης οι πάροχοι ηλεκτρισμού παράγουν την ενέργεια κοντά στους πόλους κατανάλωσης. Η διαδικασία αυτή ελαχιστοποιεί το μέγεθος της υποδομής μεταφοράς και διανομής της ενέργειας με μεγάλα κέρδη τόσο ενεργειακά όσο και οικονομικά. Επιπροσθέτως, τα μοντέλα πρόβλεψης βοηθάνε στον προγραμματισμό της συντήρησης των συστημάτων ενέργειας. Έχοντας καλύτερη επίγνωση της ζήτησης, η εταιρεία διανομής ενέργειας λαμβάνει ενημερωμένες αποφάσεις για την διεξαγωγή έργων συντήρησης, περιορίζοντας τις επιπτώσεις στην καθημερινότητα των καταναλωτών[1]. Τέλος, η πρόβλεψη του ηλεκτρικού φορτίου είναι αδήριτης σημασίας για την σωστή και αποτελεσματική διαχείριση και ενσωμάτωση ανανεώσιμων πηγών ενέργειας, όπως ηλιακή ή αιολική. Η αποτελεσματικότητα αυτών των πηγών ενέργειας βασίζεται σε μεγάλο βαθμό στις περιβαλλοντικές συνθήκες, κάτι που τα μοντέλα προβλέψεων ενσωματώνουν.[4]

A. Ο ENSOE-E και η Α.Δ.Μ.Η.Ε.

Ο ευρωπαϊκός σύνδεσμος **'European Network of Transmission System Operators for Electricity'**, (ENTSO-E) αντιπροσωπεύει 42 Διαχειριστές Συστημάτων Μεταφοράς από 35 χώρες της ΕΕ, και η έκταση του υπερβαίνει τα σύνορα της Ε.Ε.. Η αποστολή του είναι η προώθηση της καλύτερης συνεργασίας μεταξύ των διαφορετικών διαχειριστών ηλεκτρικής ενέργειας της Ευρώπης για την εξασφάλιση ενός πιο αποδοτικού συστήματος, ασφαλούς και αξιόπιστου συστήματος εφοδιασμού ηλεκτρικής ενέργειας της ηπείρου[5].

Ο **Ανεξάρτητος Διαχειριστής Μεταφοράς Ηλεκτρικής Ενέργειας (ΑΔΜΗΕ)** συστάθηκε το 2011 με σκοπό να αναλάβει τα καθήκοντα Διαχειριστή του Ελληνικού Συστήματος Μεταφοράς Ηλεκτρικής Ενέργειας (ΕΣΜΗΕ). Στο πλαίσιο αυτό, σκοπός της Εταιρείας είναι η λειτουργία, ο έλεγχος, η συντήρηση και η ανάπτυξη του ΕΣΜΗΕ, ώστε να διασφαλίζεται ο εφοδιασμός της χώρας με ηλεκτρική ενέργεια, με τρόπο επαρκή, ασφαλή, αποδοτικό και αξιόπιστο καθώς και η λειτουργία της αγοράς ενέργειας και του διασυνοριακού εμπορίου σύμφωνα με τις αρχές της διαφάνειας, της ισότητας και του ελεύθερου ανταγωνισμού[6].

B. Το πρόγραμμα

Το πρόγραμμα είναι γραμμένο σε γλώσσα python 3 καθώς μας προσφέρει πρόσβαση σε πληθώρα βιβλιοθηκών τόσο για γραφική αναπαράσταση δεδομένων(matplotlib) όσο και για data scraping (BeautifulSoup4), άλλωστε το γεγονός ότι είναι υψηλού επιπέδου γλώσσα την καθιστά κατάλληλη για ανάλυση δεδομένων. Το πρόγραμμα επιτρέπει στον χρήστη να αναλύσει μια ή πολλές ημερομηνίες δημιουργώντας γραφικές παραστάσεις και υπολογίζοντας στατιστικά στοιχεία (τυπική απόκλιση, μέση τιμή κ.τ.λ.). Επίσης, το πρόγραμμα επιτρέπει στον χρήστη να συγκρίνει την κατανάλωση της Ελλάδας με άλλες ευρωπαϊκές χώρες για μια δεδομένη μέρα. Συγκεκριμένα, το πρόγραμμά ζητά από τον χρήστη, ποια λειτουργία θέλει να εκτελέσει (πατώντας 1,2,3). Ο χρήστης εισάγει την ημερομηνία σε μορφή **DD.MM.YYYY** και οι πληροφορίες για το φορτίο αυτής της μέρας εισάγονται αυτόματα από την ιστοσελίδα του E.N.T.S.O.E., με χρήση των βιβλιοθηκών BeautifulSoup και lmxl. Κατόπιν, εκτυπώνονται οι τιμές του ηλεκτρικού φορτίου ανά μια ώρα και παρουσιάζονται επιλογές στον χρήστη για οπτικοποιήση και ανάλυση των δεδομένων του φορτίου. Αυτά τα χαρακτηριστικά είναι κοινά για όλες τις λειτουργίες του προγράμματος. Επίσης, είναι αξιοσημείωτο να αναφερθεί ότι όλες οι γραφικές αναπαραστάσεις έγιναν με την βοήθεια της εξωτερικής βιβλιοθήκης matplotlib και οι στατιστικοί υπολογισμοί έγιναν με τις βιβλιοθήκες statistics και sklearn.

- Αν ο χρήστης πατήσει 1 τότε επιλέγει την λειτουργία της εκτενούς ανάλυσης μιας συγκεκριμένης μέρας. Οι επιλογές του είναι:

1) Αν ο χρήστης πατήσει 1 τότε μία γραφική παράσταση της πραγματικής τιμής φορτίου

και της πρόβλεψης της επόμενης μέρας του φορτίου θα εμφανιστεί. Επίσης, αναπαριστώνται και οι μέσες τιμές τόσο της πραγματικής τιμής φορτίου όσο και της πρόβλεψης της επόμενης μέρας του φορτίου.

2) Αν ο χρήστης πατήσει 2 τότε ένα ραβδόγραμμα της πραγματικής τιμής φορτίου και της πρόβλεψης της επόμενης μέρας του φορτίου θα εμφανιστεί, με τις μέσες τιμές τους.
3) Αν ο χρήστης πατήσει 3 τότε ένα ραβδόγραμμα της πραγματικής τιμής φορτίου θα εμφανιστεί. Επίσης, αναπαρίσταται και η μέση τιμή της πραγματικής τιμής φορτίου.
4) Αν ο χρήστης πατήσει 4 τότε ένα ραβδόγραμμα της πραγματικής τιμής φορτίου θα εμφανιστεί. Επίσης, αναπαρίσταται και η μέση τιμή της πραγματικής τιμής φορτίου.
5) Αν ο χρήστης πατήσει 5 τότε μια γραφική παράσταση του ποσοστού του σφάλματος της πρόβλεψης ανά ώρα θα εμφανιστεί.
6) Αν ο χρήστης πατήσει 6 τότε μια γραφική αναπαράσταση του ποσοστού της ακρίβειας της πρόβλεψης ανά ώρα θα εμφανιστεί.
7) Αν ο χρήστης πατήσει 7 τότε μια γραφική παράσταση του φορτίου ως ποσοστό του μέγιστου φορτίου ανά ώρα θα εμφανιστεί.
8) Αν ο χρήστης πατήσει 8 τότε θα εκτυπωθεί το μέσο απόλυτο σφάλμα της ημέρας.
9) Αν ο χρήστης πατήσει 9 τότε θα εκτυπωθεί το μέσο απόλυτο ποσοστιαίο σφάλμα της ημέρας.
10) Αν ο χρήστης πατήσει 10 τότε θα εκτυπωθεί η τυπική απόκλιση της μέρας
11) Αν ο χρήστης πατήσει 11 τότε θα εκτυπωθεί το μέσο τετραγωνικό σφάλμα της ημέρας.
12) Αν ο χρήστης πατήσει 12 τότε θα εκτυπωθεί η μέση τιμή του πραγματικού φορτίου.
13) Αν ο χρήστης πατήσει 13 τότε θα εκτυπωθεί η μέση τιμή της πρόβλεψης της επόμενης μέρας του φορτίου.
14) Αν ο χρήστης πατήσει 14 τότε θα εκτυπωθεί η διάμεση τιμή του πραγματικού φορτίου.
15) Αν ο χρήστης πατήσει 15 τότε θα εκτυπωθεί η διάμεση τιμή της πρόβλεψης της επόμενης μέρας του φορτίου.
16) Αν ο χρήστης πατήσει 16 τότε θα εμφανιστεί η γραφική παράσταση της κανονικής κατανομής του πραγματικού φορτίου.

- Αν ο χρήστης πατήσει 2 τότε επιλέγει την λειτουργία σύγκρισης πολλών ημερομηνιών. Συγκεκριμένα οι επιλογές είναι:

1) Αν ο χρήστης πατήσει 1 τότε εκτυπώνονται σε αύξουσα σειρά όλες οι μέσες τιμές των πραγματικών φορτίων.
2) Αν ο χρήστης πατήσει 2 τότε εκτυπώνονται σε αύξουσα σειρά όλες οι τιμές του μέσου απόλυτου σφάλματος των φορτίων.
3) Αν ο χρήστης πατήσει 3 τότε εκτυπώνονται σε αύξουσα σειρά όλες οι τιμές του μέσου απόλυτου ποσοστιαίου σφάλματος των πραγματικών φορτίων.
4) Αν ο χρήστης πατήσει 4 τότε εκτυπώνονται σε αύξουσα σειρά όλες οι τιμές του της τυπικής απόκλισης των πραγματικών φορτίων των ημερών.
5) Αν ο χρήστης πατήσει 5 τότε εκτυπώνονται σε αύξουσα σειρά όλες οι τιμές του μέσου τετραγωνικού σφάλματος των πραγματικών φορτίων των ημερών.
6) Αν ο χρήστης πατήσει 6 τότε παρουσιάζεται μια γραφική παράσταση των πραγματικών φορτίων για πολλές μέρες.
7) Αν ο χρήστης πατήσει 7 τότε παρουσιάζεται μια γραφική παράσταση του ποσοστού του σφάλματος του φορτίου για πολλές μέρες.
8) Αν ο χρήστης πατήσει 8 τότε παρουσιάζεται μια γραφική παράσταση του ποσοστού της ακρίβειας του φορτίου για πολλές μέρες.
9) Αν ο χρήστης πατήσει 9 τότε παρουσιάζεται μια γραφική παράσταση του ποσοστού του πραγματικού φορτίου ως ποσοστό του μέγιστου φορτίου της μέρας για όλες τις μέρες.

- Αν ο χρήστης πατήσει 3, τότε επιλέγει την λειτουργία σύγκρισης της κατανάλωσης με άλλες ευρωπαϊκές χώρες. Οι επιλογές για χώρες είναι Γαλλία, Φινλανδία, Τσεχία, Πολωνία, Πορτογαλία, Ισπανία .Ο χρήστης έχει τις εξής επιλογές:

1) Αν ο χρήστης πατήσει 1, τότε θα εμφανιστεί μία γραφική παράσταση της πραγματικής τιμής φορτίου της Ελλάδας και της χώρας που επιλέχθηκε για σύγκριση. Επίσης, αναπαριστώνται και οι μέσες τιμές της πραγματικής τιμής φορτίου των 2 χωρών
2) Αν ο χρήστης πατήσει 2, τότε θα εμφανιστεί ένα ραβδόγραμμα που αναπαριστά την πραγματική τιμή φορτίου της Ελλάδας και

της χώρας που επιλέχθηκε για σύγκριση και τις μέσες τιμές των φορτίων.

3) Αν ο χρήστης πατήσει 3 τότε θα εμφανιστεί μία γραφική παράσταση του ποσοστού του σφάλματος της πρόβλεψης της επόμενης μέρας της Ελλάδας και της χώρας που επιλέχθηκε για σύγκριση.
4) Αν ο χρήστης πατήσει 4 τότε θα εμφανιστεί μία γραφική παράσταση του ποσοστού της ακρίβειας της πρόβλεψης της επόμενης μέρας της Ελλάδας και της χώρας που επιλέχθηκε για σύγκριση.
5) Αν ο χρήστης πατήσει 5 τότε θα εμφανιστεί μία γραφική παράσταση του φορτίου ως ποσοστό του μέγιστου ηλεκτρικού φορτίου της πρόβλεψης της επόμενης μέρας τόσο της Ελλάδας όσο και της χώρας που επιλέχθηκε για σύγκριση.
6) Αν ο χρήστης πατήσει 6 τότε εκτυπώνεται το μέσο απόλυτο σφάλμα των πραγματικών φορτίων, τόσο της Ελλάδας, όσο και της χώρα που επιλέχθηκε για σύγκρισή.
7) Αν ο χρήστης πατήσει 7 τότε εκτυπώνεται το μέσο απόλυτο ποσοστιαίο σφάλμα των πραγματικών φορτίων, τόσο της Ελλάδας, όσο και της χώρα που επιλέχθηκε για σύγκρισή.
8) Αν ο χρήστης πατήσει 8 τότε εκτυπώνεται η τυπική απόκλιση των πραγματικών φορτίων, τόσο της Ελλάδας, όσο και της χώρα που επιλέχθηκε για σύγκρισή.
9) Αν ο χρήστης πατήσει 9 τότε εκτυπώνονται οι τιμές του μέσου τετραγωνικού σφάλματος των πραγματικών φορτίων, τόσο της Ελλάδας, όσο και της χώρα που επιλέχθηκε για σύγκρισή.
10) Αν ο χρήστης πατήσει 10 τότε εκτυπώνεται η μέση τιμή, τόσο της Ελλάδας, όσο και της χώρα που επιλέχθηκε για σύγκρισή.
11) Αν ο χρήστης πατήσει 11, τότε θα εμφανιστεί μία γραφική αναπαράσταση του κατά κεφαλήν φορτίου της Ελλάδας και της χώρας που επιλέχθηκε για σύγκριση καθώς και οι μέσες τιμές των φορτίων των 2 χωρών
12) Αν ο χρήστης πατήσει 12, τότε θα εμφανιστεί μία γραφική αναπαράσταση του φορτίου προς το Ακαθάριστο Εγχώριο Προϊόν της Ελλάδας και της χώρας που επιλέχθηκε για σύγκριση με τις μέσες τιμές τους. Επίσης, το Α.Ε.Π μεταβάλλεται για κάθε χρονιά βάσει μιας πρόβλεψη ανάπτυξης της κάθε χώρας.

- Τέλος, για την ορθή λειτουργία του προγράμματος ο χρήστης πρέπει να έχει εγκαταστήσει στον υπολογιστή του την python 3.9 και τις εξωτερικές βιβλιοθήκες beautifulsoup4, matplotlib, lmxl, Cycler, requests, numpy, scikit-learn.

III. ΣΥΝΟΨΗ

Ένα από τα πιο σημαντικά μέρη του σύγχρονου συστήματος διανομής ηλεκτρικής ενέργειας είναι η πρόβλεψη. Στο πρόγραμμα της εργασίας αναλύεται οι προβλέψεις του φορτίου ηλεκτρικής ενέργειας της Ελλάδας

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Anwar, Tahreem & Sharma, Bhaskar & Chakraborty, Koushik & Sirohia, Himanshu. (2018). Introduction to Load Forecasting. International Journal of Pure and Applied Mathematics. 119. 1527-1538.

[2] S. Singh, S. Hussain and M. A. Bazaz, "Short term load forecasting using artificial neural network," *2017 Fourth International Conference on Image Information Processing (ICIIP)*, 2017, pp. 1-5, doi: 10.1109/ICIIP.2017.8313703.

[3] A. Gupta and A. Kumar, "Mid Term Daily Load Forecasting using ARIMA, Wavelet-ARIMA and Machine Learning," *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, 2020, pp. 1-5, doi: 10.1109/EEEIC/ICPSEurope49358.2020.9160563.

[4] P. S. Sauter, P. Karg, M. Kluwe and S. Hohmann, "Load Forecasting in Distribution Grids with High Renewable Energy Penetration for Predictive Energy Management Systems," *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, 2018, pp. 1-6, doi: 10.1109/ISGTEurope.2018.8571524.

[5]https://www.entsoe.eu/about/inside-entsoe/objectives/

[6]https://www.admie.gr/i-etaireia/me-mia-matia

IV. Appendix: Ο Κώδικας του Προγράμματος:

```
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
import matplotlib.ticker as plticker
from datetime import date
from collections import OrderedDict
from operator import itemgetter
from matplotlib import use
import statistics
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_absolute_percentage_error as err
from sklearn.metrics import mean_squared_error
import requests
import numpy as np
import random
from itertools import cycle
import sys


def main():
    use("TkAgg")
    print("Hello, this is a small program to analyze the Energy Load of the
greek power system for a given date, as well as the Forecast of the expected
Load.")
    while(True):
        print("\nPlease press 1 if you want to thoroughly analyze a given
date.")
        print("Please press 2 if you want to analyze a number of dates.")
        print("Please press 3 if you want to compare Greece with another
country for a given date.")
        print("Please enter exit if you want to terminate the program.")
        scan=input("Please enter your choice: ")
        if(scan=='1'):
            First()
        elif(scan=='2'):
            number=input("Please enter how  many dates you want to analyze:
")
            try :
                Second(int(number))
            except:
                print("Something went wrong")
                pass
        elif(scan=='3'):
            Third()
        elif(scan.lower()=='exit'):
            break
        else:
            print("\n\nSomething went wrong\n")
            pass

#General Functions for all:

def scanner():
    while(True):
        dates = input("Please enter date after in DD.MM.YYYY format: ")
```

```
        list = load(dates)
        try:
            a=len(list)
        except:
            list=load(dates)
        if a==48 and list[-1].isdigit() and list[-2].isdigit():
            break
    return list,dates

def load(date):
    a=f'https://transparency.entsoe.eu/load-
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|EET|DAY&biddingZone.values=CTY|1
0YGR-HTSO-----Y!BZN|10YGR-HTSO-----
Y&dateTime.timezone=EET_EEST&dateTime.timezone_input=EET+(UTC+2)+/+EEST+(UTC+
3)'
    html_text=requests.get(a).text
    soup= BeautifulSoup(html_text,'lxml')
    data=soup.find_all('td',class_="dv-value-cell")
    list_1=[]
    for i in data:
        list_1.append(i.text)
    return list_1

def print_values(list):
    print("TIME     SPECULATION   REAL")
    for o in range(len(list)):
        if(o%2==0):
            if((int(o / 2) + 1) == 10):
                print(f'{int(o / 2)}-{int(o / 2 + 1)}:       {list[o]}     ',
format(list[o + 1]))
            elif ((int(o / 2) + 1) > 9):
                print(f'{int(o / 2)}-{int(o / 2 + 1)}:      {list[o]}     ',
format(list[o + 1]))

            else:
                print(f'{int(o/2)}-{int(o/2+1)}:        {list[o]}
',format(list[o+1]))

def get_real(list):
    real=[]
    for i in range(1,len(list),2):
        try:
            real.append(int(list[i]))
        except ValueError:
            sys.exit('Incomplete Data')
    return real

def get_spec(list):
    spec=[]
    for i in range(0,len(list),2):
        try:
            spec.append(int(list[i]))
        except ValueError:
            sys.exit('Incomplete Data')
    return spec
```

```
def find_date(dates):
    day, month, year = dates.split('.')
    day_name = date(int(year), int(month), int(day))
    return day_name.strftime("%A")

def percent_error(real,spec):
    error=[]
    for i in range(len(real)):
        error.append(round(((-real[i]+spec[i])/real[i])*100,2))
    return error

class First():
    def __init__(self):
        # Import the date
        list,dates=scanner()
        day, month, year = dates.split('.')
        day_name = date(int(year), int(month), int(day))
        print(f'The day is a {day_name.strftime("%A")}')

        print_values(list)
        #Find the Actual Load
        real=get_real(list)
        # Find the Forecast Load
        spec=get_spec(list)
        #Find the Mean
        mean_real=[statistics.mean(real)]*len(real)
        mean_spec=[statistics.mean(spec)]*len(spec)
        error = percent_error(real,spec)
        while(True):
            print("\nPlease enter a number from 1 to 16:\n1: A line graph of
both Actual and Forecast of the load values.")
            print("2: A bar chart of both Actual and Day-ahead Total Forecast
of the load values.")
            print("3: A bar chart of the Actual load value.")
            print("4: A bar chart of the Day-ahead Total Load Forecast.")
            print("5: A line graph of the % of error of our Forecast.")
            print("6: A line graph of the % of accuracy of our Forecast.")
            print("7: A line graph of load per hour as a percentage of
maximum daily energy load.")
            print("8: Print the Mean Absolute Error.")
            print("9: Print the Mean Absolute Percentage Error")
            print("10: Print the Standard Deviation of the Actual Load.")
            print("11: Print the Mean Squared Error")
            print("12: Print the Mean value of the Actual Load.")
            print("13: Print the Mean value of the Day-ahead Total Load
Forecast.")
            print("14: Print the Median value of the Actual Load.")
            print("15: Print the Median value of the Day-ahead Total Load
Forecast.")
            print("16: A line graph of the normal distribution of the Actual
Total Load.")
            print("To end the program just enter 'End' or 'Telos'.")
            scan = input("Please enter your choice: ")
            print('\n')
            if(scan=='1'):
                self.graph(real, spec,mean_real,mean_spec)
            elif(scan=='2'):
```

```
                self.bar_chart_both(real, spec, mean_real, mean_spec)
            elif(scan=='3'):
                self.bar_chart_real(real,mean_real)
            elif(scan=='4'):
                self.bar_chart_forecast(spec,mean_spec)
            elif (scan=='5'):
                self.percent_error_show(error)
            elif (scan=='6'):
                self.percent_accuracy_show(error)
            elif(scan=='7'):
                self.peak_energy_load(real)
            elif(scan=='8'):
                print(f'The Mean Absolute error is: {round(mae(real,spec),3)}
MW')
            elif(scan=='9'):
                print(f'The Mean Absolute percentage % Error is:
{round(err(real, spec) * 100, 3)}%')
            elif(scan=='10'):
                print(f'The Standard Deviation of the Actual Load is:
{round(statistics.stdev(real), 3)}')
            elif(scan=='11'):
                print(f'The Mean Squared Error is:
{round(mean_squared_error(real, spec),3)}')
            elif(scan=='12'):
                print(f'The Mean value of the Actual Load is:
{round(mean_real[0],3)} MW')
            elif(scan=='13'):
                print(f'The Mean value of the Day-ahead Total Load Forecast
is: {round(mean_spec[0],3)} MW')
            elif(scan=='14'):
                print(f'The Median value of the Actual Load is:
{round(statistics.median(real),3)} MW')
            elif(scan=='15'):
                print(f'The Median value of the Day-ahead Total Load Forecast
is: {round(statistics.median(spec),3)} MW')
            elif(scan=='16'):
                self.normal_distribution(real, mean_real[0])
            elif(scan.lower()=="end" or scan.lower()=="telos"):
                break
            else:
                print("Something went wrong\n")
                pass

    # A graph of the percentage of Error of the Actual Load
    def percent_error_show(self,error):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
        plt.figure(figsize=(10,4))

plt.plot(time,error,label='Error',color='red',linewidth='2',marker='.',marker
size='10',markeredgecolor='black',markerfacecolor='red')
        plt.title("How off was the Forecast? [The % of error between the
Forecast Load and the Actual Load]",fontdict={'fontname' : 'Times New Roman',
'fontsize': 16})
        plt.ylabel('Error %',fontdict={'fontname' : 'Times New Roman'})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New Roman'})
        plt.xticks([x for x in range(24)])
```

```
        plt.ylim(-20, 20)

        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)

        plt.tight_layout()
        plt.legend()
        plt.show()

    # A graph of the percentage of Accuracy of the Actual Load
    def percent_accuracy_show(self,error):
        accuracy=[]
        for i in error:
            accuracy.append(100-abs(i))
        time=[x for x in range(24)]
        loc = plticker.MultipleLocator(base=1.0)
        plt.figure(figsize=(10,4))

plt.plot(time,accuracy,label='Accuracy',color='red',linewidth='2',marker='.',
markersize='10',markeredgecolor='black',markerfacecolor='red')
        plt.title("How close was the Forecast? [Forecast as a % of the real
value]",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
        plt.ylabel('ERROR',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New Roman'})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    # A graph of the Actual Load and the Day-Ahead Forecast
    def graph(self,real, spec,mean_real,mean_spec):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
        plt.figure(figsize=(10,4))
        plt.plot(time, mean_real, label='Mean of Actual Total Load',
linestyle='dashdot',color='yellow')
        plt.plot(time, mean_spec, label='Mean of Day-ahead Total Load
Forecast', linestyle='--',color='green')
        plt.plot(time,spec,label='Total Load
Forecast',color='blue',linewidth='2',marker='.',markersize='10',markeredgecol
or='black',markerfacecolor='blue')
        plt.plot(time,real,label='Actual Total Load
',color='red',linewidth='2',marker='.',markersize='10',markeredgecolor='black
',markerfacecolor='red')
        plt.title("Load: Day-ahead [Total Load Forecast] vs [Actual Total
Load]",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load [MW]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        ax = plt.gca()
```

```
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()

        plt.show()

    # A bar chart of the Actual Load
    def bar_chart_real(self,real,mean_real):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
        plt.figure(figsize=(10,4))
        bars= plt.bar(time,real,edgecolor='black')
        plt.plot(time,real,label='Actual Total
Load',color='black',linewidth='1',marker='.',markersize='10',markeredgecolor=
'black',markerfacecolor='black')
        plt.plot(time, mean_real, label='Mean', linestyle='--
',color='purple')

        plt.title("Load [Actual Total Load]",fontdict={'fontname' : 'Times
New Roman', 'fontsize': 20})
        plt.xticks([x for x in range(24)])
        plt.ylabel('Load [MW]',fontdict={'fontname' : 'Times New Roman',
'fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New Roman',
'fontsize': 10})
        max_value=max(real)
        max_index =  real.index(max_value)
        bars[max_index].set_color('r')
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)

        min_value=min(real)
        min_index =  real.index(min_value)
        bars[min_index].set_color('green')
        plt.tight_layout()
        plt.legend()
        plt.show()

    # A bar chart of the Day-Ahead Forecast Load
    def bar_chart_forecast(self,spec,mean_spec):
        loc = plticker.MultipleLocator(base=1.0)

        time=[x for x in range(24)]
        plt.figure(figsize=(10,4))
        bars= plt.bar(time,spec,edgecolor='black')
        plt.plot(time,spec,label='Total Load
Forecast',color='black',linewidth='1',marker='.',markersize='10',markeredgeco
lor='black',markerfacecolor='black')
        plt.plot(time, mean_spec, label='Mean', linestyle='--
',color='purple')

        plt.title("Load [Total Load Forecast]",fontdict={'fontname' : 'Times
New Roman', 'fontsize': 20})
```

```
        plt.xticks([x for x in range(24)])
        plt.ylabel('Load [MW]',fontdict={'fontname' : 'Times New Roman',
'fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New Roman',
'fontsize': 10})

        max_value=max(spec)
        max_index =  spec.index(max_value)
        bars[max_index].set_color('r')
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)

        min_value=min(spec)
        min_index =  spec.index(min_value)
        bars[min_index].set_color('green')

        plt.tight_layout()
        plt.legend()

        plt.show()

    # A graph of the the Actual Load as a a percentage of the peak Actual
Load
    def peak_energy_load(self,real):
        loc = plticker.MultipleLocator(base=1.0)
        maximum =  real.index(max(real))
        list=[]
        for i in real:
            list.append(round(i/real[maximum]*100,3))
        time=[x for x in range(24)]
        plt.figure(figsize=(10,4))
        plt.plot(time,list,label='Value percent
%',color='red',linewidth='2',marker='.',markersize='10',markeredgecolor='blac
k',markerfacecolor='red')

        plt.title("Load per hour as a percentage of maximum daily energy
load",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load %',fontdict={'fontname' : 'Times New Roman'})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New Roman'})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    # A bar chart of the Actual Load and the Day-Ahead Forecast Load
    def bar_chart_both(self,real,spec,mean_real,mean_spec):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
        width=np.min(np.diff(time))/3
        fig=plt.figure(figsize=(10,4))
        ax=fig.add_subplot(111)
        plt.plot(time, mean_real, label='Mean of Actual Total Load',
```

```
linestyle='dashdot',color='yellow')
        plt.plot(time, mean_spec, label='Mean of Day-ahead Total Load
Forecast', linestyle='--',color='green')

        ax.bar(time - width, real, width, color='b', label='Actual Total
Load', align='edge')
        ax.bar(time, spec, width, color='r', label='Day-ahead Total Load
Forecast', align='edge')
        ax.xaxis.set_major_locator(loc)
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')

        plt.tight_layout()
        plt.legend()
        plt.show()

    # A graph of the normal distribution of the Actual Load
    def normal_distribution(self,real,mean_real):
        import scipy.stats
        value=real[:]
        value.sort()
        x_min = 0
        x_max = value[-1]+value[0]
        mean = mean_real
        std = statistics.stdev(real)
        x = np.linspace(x_min, x_max,)

        y = scipy.stats.norm.pdf(x, mean, std)

        plt.plot(x, y, color='black',linewidth='2')

        # ----------------------------------------------------------------
----------------------#
        # fill area 1

        pt1 = mean + std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
color='black')

        pt2 = mean - std
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
color='black')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)

        plt.fill_between(ptx, pty, color='#0b559f', alpha=1)

        # ----------------------------------------------------------------
----------------------#
        # fill area 2

        pt1 = mean + std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
linestyle='dashdot',color='red')

        pt2 = mean + 2.0 * std
```

```
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
linestyle='dashdot',color='red')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)

        plt.fill_between(ptx, pty, color='#2b7bba', alpha=1)

        # ---------------------------------------------------------------
---------------------#
        # fill area 3

        pt1 = mean - std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
linestyle='dashdot',color='red')

        pt2 = mean - 2.0 * std
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
linestyle='dashdot',color='red')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)

        plt.fill_between(ptx, pty, color='#2b7bba', alpha=1)

        # ---------------------------------------------------------------
---------------------#
        # fill area 4

        pt1 = mean + 2.0 * std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
linestyle='dashdot',color='red')

        pt2 = mean + 3.0 * std
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
linestyle='dashdot',color='red')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)

        plt.fill_between(ptx, pty, color='#539ecd', alpha=1)

        # ---------------------------------------------------------------
---------------------#
        # fill area 5

        pt1 = mean - 2.0 * std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
linestyle='dashdot',color='red')

        pt2 = mean - 3.0 * std
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
linestyle='dashdot',color='red')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)
```

```
        plt.fill_between(ptx, pty, color='#539ecd', alpha=1)

        # ------------------------------------------------------------------
--------------------#
        # fill area 6

        pt1 = mean + 3.0 * std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
linestyle='dashdot',color='red')

        pt2 = mean + 10.0 * std
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
linestyle='dashdot',color='red')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)

        plt.fill_between(ptx, pty, color='#89bedc', alpha=1)

        # ------------------------------------------------------------------
--------------------#
        # fill area 7

        pt1 = mean - 3.0 * std
        plt.plot([pt1, pt1], [0.0, scipy.stats.norm.pdf(pt1, mean, std)],
linestyle='dashdot',color='red')

        pt2 = mean - 10.0 * std
        plt.plot([pt2, pt2], [0.0, scipy.stats.norm.pdf(pt2, mean, std)],
linestyle='dashdot',color='red')

        ptx = np.linspace(pt1, pt2, 10)
        pty = scipy.stats.norm.pdf(ptx, mean, std)

        plt.fill_between(ptx, pty, color='#89bedc', alpha=1)

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

        plt.grid()


        plt.title('Normal distribution of Actual Load', fontdict={'fontname'
: 'Times New Roman', 'fontsize': 20})

        plt.xlabel('LOAD [MW]',fontdict={'fontname' : 'Times New Roman',
'fontsize': 10})
        plt.ylabel('Normal Distribution',fontdict={'fontname' : 'Times New
Roman', 'fontsize': 10})
        plt.show()

class Second():

    def __init__(self,number):
        total=[]
        self.num=number
        #import all the dates and separate the Actual from the forecast
```

```
        for i in range(self.num):
            #import a date
            list,dates=scanner()
            a=find_date(dates)
            print(f'The day is a {a}')
            print_values(list)
            #add the name
            total.append(dates +' which is a ' + a)
            # add the Real Actual Load
            total.append(get_real(list))
            # add the Day-Ahead Forecast Actual Load
            total.append(get_spec(list))
        while(True):
            print("Please enter a number from 1 to 5:\n1: The Mean value of
the Actual Total Load.")
            print("2: The Mean Absolute Error of all the dates in order.")
            print("3: The Mean Absolute Percentage % Error of all the dates
in order.")
            print("4: The Standard Deviation of all the dates in order.")
            print("5: The Mean Squared Error of all the dates in order.")
            print("6: A Graph of all the Actual Total Load of all the
dates.")
            print("7: A Graph of the Percentage % of Error of all the
dates.")
            print("8: A Graph of the Percentage % of Accuracy of all the
dates")
            print("9: A Graph of Load per hour as a percentage % of maximum
daily energy load ")
            print("To end the program just enter 'End' or 'Telos'.")
            scan = input("Please enter your choice: ")
            print('\n')
            if scan=='1':
                self.mean_value_2(total)
            elif(scan=='2'):
                self.MAE(total)
            elif(scan=='3'):
                self.MAPE(total)
            elif(scan=='4'):
                self.standard_deviation(total)
            elif(scan=='5'):
                self.meansquarederror(total)
            elif(scan=='6'):
                self.graph_2(total)
            elif(scan=='7'):
                self.percent_error_2(total)
            elif(scan=='8'):
                self.percent_accuracy_2(total)
            elif(scan=='9'):
                self.percent_peak_energy_2(total)
            elif(scan.lower()=="end" or scan.lower()=="telos"):
                break
            else:
                print("Something went wrong\n")
                pass

    # Find the Mean Average Percentage Error of all the dates in ascending
order
```

```
    def mean_value_2(self,total):
        dict = OrderedDict()
        for i in range(0, len(total), 3):
            a = round(statistics.mean(total[i + 1]),3)
            dict[total[i]] = a

        d = OrderedDict(sorted(dict.items(), key=itemgetter(1)))
        print("\nThe Mean Actual Load in ascending order:")

        for key, value in d.items():
            print(f'\tOn {key} the Mean Total Load is {value} MW')
        print('\n')

    #Find the Mean Average Percentage Error of all the dates in ascending
order
    def MAPE(self,total):
        dict = OrderedDict()
        for i in range(0, len(total), 3):
            a = round(err(total[i+1],total[i+2])*100,3)
            dict[total[i]] = a

        d = OrderedDict(sorted(dict.items(), key=itemgetter(1)))
        print("\nThe Mean Absolute percentage % Error in ascending order:")

        for key, value in d.items():
            print(f'\tOn {key} the Mean Absolute percentage Error is
{value}%')
        print('\n')

    #Find the Mean Average Error of all the dates in ascending order
    def MAE(self,total):
        dict = OrderedDict()

        for i in range(0, len(total), 3):
            a = round(mae(total[i+1],total[i+2]),3)
            dict[total[i]] = a

        d = OrderedDict(sorted(dict.items(), key=itemgetter(1)))
        print("\nThe Mean Absolute Error in ascending order:")

        for key, value in d.items():
            print(f'\tOn {key} the Mean Absolute Error is {value} MW')
        print('\n')

    #Find the Standard Deviation of all the dates in ascending order
    def standard_deviation(self,total):
        dict = OrderedDict()

        for i in range(0, len(total), 3):
            a = round(statistics.stdev(total[i+1]),3)
            dict[total[i]] = a

        d = OrderedDict(sorted(dict.items(), key=itemgetter(1)))
        print("\nThe Standard Deviation in ascending order:")

        for key, value in d.items():
            print(f'\tOn {key} the Standard Deviation Error is {value}')
```

```
        print('\n')

    #Find the Mean Squared Error of all the dates in ascending order
    def meansquarederror(self,total):
        dict = OrderedDict()

        for i in range(0, len(total), 3):
            a = round(mean_squared_error(total[i+1],total[i+2]), 3)
            dict[total[i]] = a

        d = OrderedDict(sorted(dict.items(), key=itemgetter(1)))
        print("\nThe Mean Squared Error in ascending order:")

        for key, value in d.items():
            print(f'\tOn {key} the Mean Squared Error is {value}')
        print('\n')

    #Graph the Actual Load of all the dates
    def graph_2(self,total):
        loc = plticker.MultipleLocator(base=1.0)
        values=['-', '--', '-.', ':', ' ', '', 'solid', 'dashed', 'dashdot',
'dotted']
        cycol=cycle(values)
        time=[x for x in range(24)]
        plt.figure(figsize=(10, 4))
        list_1=[]
        for i in range(0, len(total), 3):
            a = round(statistics.mean(total[i + 1]),3)
            list_1.append([a]*len(total[1]))
        j=0
        for i in range(0,len(total),3):

plt.plot(time,total[i+1],label=total[i],c=(random.random(),random.random(),ra
ndom.random()),linewidth='2',marker='.',markersize='10',markeredgecolor='blac
k')
            plt.plot(time, list_1[j], label="The mean of "+ total[i],
linestyle=next(cycol), c=(random.random(),random.random(),random.random()))
            j+=1

        plt.title("Graph of all the Actual Loads",fontdict={'fontname' :
'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load [MW]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    # A graph of the percentage of error of all the Actual Loads
    def percent_error_2(self,total):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
```

```
        plt.figure(figsize=(10, 4))
        list_1=[]
        for i in range(0, len(total), 3):
            a= percent_error(total[i+1],total[i+2])
            list_1.append(a)
        j=0
        for i in range(0,len(total),3):

plt.plot(time,list_1[j],label=total[i],c=(random.random(),random.random(),ran
dom.random()),linewidth='2',marker='.',markersize='10',markeredgecolor='black
')
            j+=1
        plt.title("The percentage % of Error of all the
dates",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Error %',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    # A graph of the percentage of accuracy of all the Actual Loads
    def percent_accuracy_2(self,total):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
        plt.figure(figsize=(10, 4))
        list_1=[]
        for i in range(0, len(total), 3):
            a= [100-abs(x) for x in percent_error(total[i+1],total[i+2])]
            list_1.append(a)

        j=0
        for i in range(0,len(total),3):

plt.plot(time,list_1[j],label=total[i],c=(random.random(),random.random(),ran
dom.random()),linewidth='2',marker='.',markersize='10',markeredgecolor='black
')
            j+=1
        plt.title("The percentage % of Accuracy of all the
dates",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Accuracy %',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()
```

```
    # A graph of the Actual Loads as a percentage of max Load of the Given
Day
    def percent_peak_energy_2(self,total):
        loc = plticker.MultipleLocator(base=1.0)
        time=[x for x in range(24)]
        plt.figure(figsize=(10, 4))
        list_1=[]
        for i in range(0, len(total), 3):
            b=[(x/max(total[i+1]))*100 for x in total[i+1]]
            list_1.append(b)
        j=0
        for i in range(0,len(total),3):

plt.plot(time,list_1[j],label=total[i],c=(random.random(),random.random(),ran
dom.random()),linewidth='2',marker='.',markersize='10',markeredgecolor='black
')
            j+=1
        plt.title("Load per hour as a percentage % of maximum daily energy
load",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load as a %',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

class Third():
    def __init__(self):
        #Add the date
        list, dates = scanner()
        day, month, year = dates.split('.')
        day_name = date(int(year), int(month), int(day))
        print(f'The day is a {day_name.strftime("%A")}')
        print("The Load of GREECE is:")
        #import for Greece
        print_values(list)
        real_gre = get_real(list)
        spec_gre = get_spec(list)
        mean_real_gre = [statistics.mean(real_gre)] * len(real_gre)
        error_gre = percent_error(real_gre, spec_gre)
        # Choose a foreign country and Import the Actual and Forecast values
        while (True):
            print("Please choose one of the following countries:
France,Finland,Czechia,Poland,Portugal,Spain")
            a=input("Choice: ")
            try:
                country= a.upper()
            except:
                country='FALSE'
            if(country=="FRANCE"):
                list_for=self.load_fra(dates)
                population=65408602
```

```
                gdp_2019=2715518
                gdp= self.gdp_adjustor(year,gdp_2019,1.7)
                break
            elif(country=="FINLAND"):
                list_for=self.load_fin(dates)
                population=5548732
                gdp_2019=269296
                gdp= self.gdp_adjustor(year,gdp_2019,2.1)
                break
            elif(country=="CZECHIA"):
                list_for=self.load_cze(dates)
                population=10727551
                gdp_2019=250681
                gdp= self.gdp_adjustor(year,gdp_2019,3.2)
                break
            elif(country=="POLAND"):
                list_for=self.load_pol(dates)
                population=37808065
                gdp_2019=595858
                print(gdp_2019)
                gdp= self.gdp_adjustor(year,gdp_2019,4.5)
                print(gdp)
                break
            elif(country=="PORTUGAL"):
                list_for=self.load_por(dates)
                population=10169149
                gdp_2019=238785
                gdp= self.gdp_adjustor(year,gdp_2019,2.65)
                break
            elif(country=="SPAIN"):
                list_for=self.load_spa(dates)
                population=46771662
                gdp_2019=1393491
                gdp= self.gdp_adjustor(year,gdp_2019,2.6)
                break
            else:
                print("Please enter the country's name properly.")
        print(f'\nThe Load of {country} is:')
        print_values(list_for)
        # find the Actual Load
        real_for = get_real(list_for)
        # find the Day-Ahead Forecast Load
        spec_for = get_spec(list_for)
        # find the Mean Value of the Actual Load
        mean_real_for = [statistics.mean(real_for)] * len(real_for)
        error_for = percent_error(real_for, spec_for)
        while(True):
            print(f"\nPlease enter a number from 1 to 10:\n1: A line graph
the Actual load Values of GREECE and {country}.")
            print(f"2: A bar chart the Actual load Values of GREECE and
{country}.")
            print(f"3: A line graph of the % of error of GREECE and
{country}.")
            print(f"4: A line graph of the % of accuracy of GREECE and
{country}.")
            print(f"5: A line graph of load per hour as a percentage of
maximum daily energy load of GREECE and {country}.")
```

```
            print(f"6: Print the Mean Absolute Error of GREECE and
{country}.")
            print(f"7: Print the Mean Absolute Percentage Error of GREECE and
{country}.")
            print(f"8: Print the Standard Deviation of the Actual Load of
GREECE and {country}")
            print(f"9: Print the Mean Squared Error of GREECE and {country}")
            print(f"10: Print the Mean value of the Actual Load of GREECE and
{country}.")
            print(f"11: A line graph of the per capita power
consumption/Actual Load of GREECE and {country}")
            print(f"12: A line graph of the Actual Load per GDP(Nominal)
GREECE and {country}")

            print("To end the program just enter 'End' or 'Telos'.")
            choice=input("Please enter your choice: ")
            print("\n")
            if(choice=='1'):
                self.graph_power_compare(real_gre, real_for, mean_real_gre,
mean_real_for,country)
            elif(choice=='2'):
                self.bar_chart_both_compare(real_gre, real_for,
mean_real_gre, mean_real_for,country)
            elif(choice=='3'):
                self.percent_error_compare(error_gre,error_for,country)
            elif (choice == '4'):
                self.percent_accuracy_compare(error_gre,error_for,country)
            elif (choice == '5'):
                self.peak_energy_load_compare(real_gre,real_for,country)
            elif (choice == '6'):
                print(f'The Mean Absolute Error of GREECE is:
{round(mae(real_gre, spec_gre), 3)} MW')
                print(f'The Mean Absolute Error of {country} is:
{round(mae(real_for, spec_for), 3)} MW')
            elif (choice == '7'):
                print(f'The Mean Absolute Percentage % Error of GREECE is:
{round(err(real_gre, spec_gre) * 100, 3)}%')
                print(f'The Mean Absolute Percentage % Error of {country} is:
{round(err(real_for, spec_for) * 100, 3)}%')
            elif (choice == '8'):
                print(f'The Standard Deviation of the Actual Load of GREECE
is: {round(statistics.stdev(real_gre), 3)}')
                print(f'The Standard Deviation of the Actual Load of
{country} is: {round(statistics.stdev(real_for), 3)}')
            elif (choice == '9'):
                print(f'The Mean Squared Error of GREECE is:
{round(mean_squared_error(real_gre, spec_gre), 3)}')
                print(f'The Mean Squared Error of {country} is:
{round(mean_squared_error(real_for, spec_for), 3)}')
            elif (choice == '10'):
                print(f'The Mean value of the Actual Load of GREECE is:
{round(statistics.mean(real_gre), 3)} MW')
                print(f'The Mean value of the Actual Load of {country} is:
{round(statistics.mean(real_for), 3)} MW')
            elif(choice=='11'):
                self.per_capita_power(real_gre, mean_real_gre, real_for,
mean_real_for, country, population)
```

```
            elif(choice=='12'):
                self.load_per_gdp(real_gre, mean_real_gre, real_for,
mean_real_for, country, gdp, year)
            elif (choice.lower() == "end" or choice.lower() == "telos"):
                break
            else:
                print("Something went wrong\n")
                pass

    #A bar chart of the load as a percentage of the Peak Load for both
countries
    def bar_chart_both_compare(self,real_gre, real_for, mean_real_gre,
mean_real_for,country):
        loc = plticker.MultipleLocator(base=1.0)
        time = [x for x in range(24)]
        width = np.min(np.diff(time)) / 3
        fig = plt.figure(figsize=(10, 4))
        ax = fig.add_subplot(111)

        plt.plot(time, mean_real_gre, label='Mean of Actual Total Load of
GREECE', linestyle='dashdot', color='yellow')
        plt.plot(time, mean_real_for, label=f'Mean of Actual Total Load of
{country}', linestyle='--', color='green')
        plt.title(f"Actual Load Greece vs {country}",fontdict={'fontname':
'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load [MW]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})
        plt.xlabel('Time [HOURS]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})

        ax.bar(time - width, real_gre, width, color='b', label='Actual Total
Load of GREECE', align='edge')
        ax.bar(time, real_for, width, color='r', label=f'Actual Total Load of
{country}', align='edge')
        ax.xaxis.set_major_locator(loc)
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')

        plt.tight_layout()
        plt.savefig('Forecast and Real bar.png')
        plt.legend()
        plt.show()

    # A graph of the Load of both countries
    def graph_power_compare(self,real_gre, real_for, mean_real_gre,
mean_real_for,country):
        loc = plticker.MultipleLocator(base=1.0)
        time = [x for x in range(24)]
        plt.figure(figsize=(10, 4))
        plt.plot(time, mean_real_gre, label='Mean of Actual Total Load of
GREECE', linestyle='dashdot', color='yellow')
        plt.plot(time, mean_real_for, label=f'Mean of Actual Total Load of
{country}', linestyle='--', color='green')
        plt.plot(time, real_gre, label='Actual Total Load of Greece',
color='blue', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='blue')
        plt.plot(time, real_for, label=f'Mean of Actual Total Load of
```

```
{country} ', color='red', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='red')
        plt.title(f"Actual Total Load [GREECE] vs
[{country}]",fontdict={'fontname': 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load [MW]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})
        plt.xlabel('Time [HOURS]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    # A graph of the load as a percentage of the Peak Load for both countries
    def peak_energy_load_compare(self,real_gre,real_for,country):
        loc = plticker.MultipleLocator(base=1.0)

        maximum = real_gre.index(max(real_gre))
        list_gre = []
        for i in real_gre:
            list_gre.append(round((i / real_gre[maximum]) * 100, 3))
        list_for = []
        maximum = real_for.index(max(real_for))

        for i in real_for:
            list_for.append(round((i / real_for[maximum]) * 100, 3))

        time = [x for x in range(24)]
        plt.figure(figsize=(10, 4))
        plt.plot(time, list_gre, label='Value percent % of GREECE',
color='blue', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='blue')
        plt.plot(time, list_for, label=f'Value percent % of {country}',
color='Red', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='red')

        plt.title("Load per hour as a percentage of maximum daily energy load
",fontdict={'fontname': 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load %', fontdict={'fontname': 'Times New Roman'})
        plt.xlabel('Time [HOURS]', fontdict={'fontname': 'Times New Roman'})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    # Load the dates for the different countries
    def load_fra(self,date):
        a=f'https://transparency.entsoe.eu/load-
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|CET|DAY&biddingZone.values=CTY|1
0YFR-RTE------C!BZN|10YFR-RTE------
```

```
C&dateTime.timezone=CET_CEST&dateTime.timezone_input=CET+(UTC+1)+/+CEST+(UTC+
2)'
        html_text=requests.get(a).text
        soup= BeautifulSoup(html_text,'lxml')
        data=soup.find_all('td',class_="dv-value-cell")
        list_1=[]
        for i in data:
            list_1.append(i.text)
        return list_1

    def load_fin(self,date):
        a=f'https://transparency.entsoe.eu/load-
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|EET|DAY&biddingZone.values=CTY|1
0YFI-1--------U!BZN|10YFI-1--------
U&dateTime.timezone=EET_EEST&dateTime.timezone_input=EET+(UTC+2)+/+EEST+(UTC+
3)'
        html_text=requests.get(a).text
        soup= BeautifulSoup(html_text,'lxml')
        data=soup.find_all('td',class_="dv-value-cell")
        list_1=[]
        for i in data:
            list_1.append(i.text)
        return list_1

    def load_pol(self,date):
        a=f'https://transparency.entsoe.eu/load-
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|CET|DAY&biddingZone.values=CTY|1
0YPL-AREA-----S!BZN|10YPL-AREA-----
S&dateTime.timezone=CET_CEST&dateTime.timezone_input=CET+(UTC+1)+/+CEST+(UTC+
2)'
        html_text=requests.get(a).text
        soup= BeautifulSoup(html_text,'lxml')
        data=soup.find_all('td',class_="dv-value-cell")
        list_1=[]
        for i in data:
            list_1.append(i.text)
        return list_1

    def load_cze(self,date):
        a=f'https://transparency.entsoe.eu/load-
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|CET|DAY&biddingZone.values=CTY|1
0YCZ-CEPS-----N!BZN|10YCZ-CEPS-----
N&dateTime.timezone=CET_CEST&dateTime.timezone_input=CET+(UTC+1)+/+CEST+(UTC+
2)'
        html_text=requests.get(a).text
        soup= BeautifulSoup(html_text,'lxml')
        data=soup.find_all('td',class_="dv-value-cell")
        list_1=[]
        for i in data:
            list_1.append(i.text)
        return list_1

    def load_por(self,date):
        a=f'https://transparency.entsoe.eu/load-
```

```
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|WET|DAY&biddingZone.values=CTY|1
0YPT-REN------W!BZN|10YPT-REN------
W&dateTime.timezone=WET_WEST&dateTime.timezone_input=WET+(UTC)+/+WEST+(UTC+1)
'
        html_text=requests.get(a).text
        soup= BeautifulSoup(html_text,'lxml')
        data=soup.find_all('td',class_="dv-value-cell")
        list_1=[]
        for i in data:
            list_1.append(i.text)
        return list_1

    def load_spa(self,date):
        a=f'https://transparency.entsoe.eu/load-
domain/r2/totalLoadR2/show?name=&defaultValue=false&viewType=TABLE&areaType=B
ZN&atch=false&dateTime.dateTime={date}+00:00|CET|DAY&biddingZone.values=CTY|1
0YES-REE------0!BZN|10YES-REE------
0&dateTime.timezone=CET_CEST&dateTime.timezone_input=CET+(UTC+1)+/+CEST+(UTC+
2)'
        html_text=requests.get(a).text
        soup= BeautifulSoup(html_text,'lxml')
        data=soup.find_all('td',class_="dv-value-cell")
        list_1=[]
        for i in data:
            list_1.append(i.text)
        return list_1

    # A graph of the percentage of Error for the 2 countries
    def percent_error_compare(self,error_gre,error_for,country):
            loc = plticker.MultipleLocator(base=1.0)
            time=[x for x in range(24)]
            plt.figure(figsize=(10,4))
            plt.plot(time,error_gre,label='Error of
Greece',color='blue',linewidth='2',marker='.',markersize='10',markeredgecolor
='black',markerfacecolor='blue')
            plt.plot(time,error_for,label=f'Error of
{country}',color='red',linewidth='2',marker='.',markersize='10',markeredgecol
or='black',markerfacecolor='red')
            plt.title(f"How off was the Forecast? [GREECE] VS
[{country}]",fontdict={'fontname' : 'Times New Roman', 'fontsize': 16})
            plt.ylabel('Error %',fontdict={'fontname' : 'Times New Roman'})
            plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman'})
            plt.xticks([x for x in range(24)])
            plt.ylim(-20, 20)

            ax = plt.gca()
            ax.tick_params(axis='x', colors='blue')
            ax.tick_params(axis='y', colors='red')
            ax.xaxis.set_major_locator(loc)

            plt.tight_layout()
            plt.legend()
            plt.show()

    # A graph of the percent accuracy for the 2 countries
```

```
    def percent_accuracy_compare(self,error_gre,error_for,country):
            accuracy_gre=[]
            for i in error_gre:
                accuracy_gre.append(100-abs(i))
            accuracy_for = []
            for i in error_for:
                accuracy_for.append(100 - abs(i))

            time=[x for x in range(24)]
            loc = plticker.MultipleLocator(base=1.0)
            plt.figure(figsize=(10,4))
            plt.plot(time,accuracy_gre,label='Accuracy of the Forecest in
GREECE',color='blue',linewidth='2',marker='.',markersize='10',markeredgecolor
='black',markerfacecolor='blue')
            plt.plot(time,accuracy_for,label=f'Accuracy of the Forecest in
{country}',color='red',linewidth='2',marker='.',markersize='10',markeredgecol
or='black',markerfacecolor='red')

            plt.title(f"How close was the Forecast? [GREECE] vs
[{country}]",fontdict={'fontname' : 'Times New Roman', 'fontsize': 20})
            plt.ylabel('ACCURACY %',fontdict={'fontname' : 'Times New
Roman','fontsize': 10})
            plt.xlabel('Time [HOURS]',fontdict={'fontname' : 'Times New
Roman'})
            plt.xticks([x for x in range(24)])

            ax = plt.gca()
            ax.tick_params(axis='x', colors='blue')
            ax.tick_params(axis='y', colors='red')
            ax.xaxis.set_major_locator(loc)

            plt.tight_layout()
            plt.legend()
            plt.show()

    # A graph of the per capita Load of the 2 countries
    def
per_capita_power(self,real_gre,mean_real_gre,real_for,mean_real_for,country,p
op):
        loc = plticker.MultipleLocator(base=1.0)
        time = [x for x in range(24)]
        list_gre=[]
        for i in real_gre:
            list_gre.append(1000000*(i/10375594))
        list_for=[]
        for i in real_for:
            list_for.append(1000000*(i/pop))
        list_mean_gre=[1000000*(x/10375594) for x in mean_real_gre]
        list_mean_for=[1000000*(x/pop) for x in mean_real_for]
        plt.figure(figsize=(10, 4))
        plt.plot(time, list_mean_gre, label='Mean Load per capita GREECE',
linestyle='dashdot', color='yellow')
        plt.plot(time, list_mean_for, label=f'Mean Load per capita
{country}', linestyle='--', color='green')
        plt.plot(time, list_gre, label='Power per capita GREECE',
color='blue', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='blue')
```

```
        plt.plot(time, list_for, label=f'Power per capita {country}',
color='red', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='red')
        plt.title(f"Actual Total Load per capita [GREECE] vs
[{country}]",fontdict={'fontname': 'Times New Roman', 'fontsize': 20})
        plt.ylabel('Load [W]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})
        plt.xlabel('Time [HOURS]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()

    #A graph of the Load as a percentage of GDP
    def
load_per_gdp(self,real_gre,mean_real_gre,real_for,mean_real_for,country,gdp,y
ear):
        loc = plticker.MultipleLocator(base=1.0)
        time = [x for x in range(24)]
        gdp_gre= self.gdp_adjustor(int(year),209853,2)
        list_gre=[]
        for i in real_gre:
            list_gre.append(i/gdp_gre)
        list_for=[]
        for i in real_for:
            list_for.append((i/gdp))
        list_mean_gre=[(x/gdp_gre) for x in mean_real_gre]
        list_mean_for=[(x/gdp) for x in mean_real_for]
        plt.figure(figsize=(10, 4))
        plt.plot(time, list_mean_gre, label='Mean Load per G.D.P. GREECE',
linestyle='dashdot', color='yellow')
        plt.plot(time, list_mean_for, label=f'Mean Load per G.D.P.
{country}', linestyle='--', color='green')
        plt.plot(time, list_gre, label='Load per G.D.P. GREECE',
color='blue', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='blue')
        plt.plot(time, list_for, label=f'Load per G.D.P. {country}',
color='red', linewidth='2', marker='.',
markersize='10',markeredgecolor='black', markerfacecolor='red')
        plt.title(f"Actual Load per Nominal G.D.P.(World Bank) in USD
[GREECE] vs [{country}]",fontdict={'fontname': 'Times New Roman', 'fontsize':
20})
        plt.ylabel('Load/G.D.P [MW/$]', fontdict={'fontname': 'Times New
Roman', 'fontsize': 10})
        plt.xlabel('Time [HOURS]', fontdict={'fontname': 'Times New Roman',
'fontsize': 10})
        ax = plt.gca()
        ax.tick_params(axis='x', colors='blue')
        ax.tick_params(axis='y', colors='red')
        ax.xaxis.set_major_locator(loc)
        plt.tight_layout()
        plt.legend()
        plt.show()
```

```
    # Adjust the 2019 GDP World Bank Estimates by the growth
    def gdp_adjustor(self,year,gdp_2019,growth):
        time=2019-int(year)
        if(time<0):
            gdp = gdp_2019 * (((1 + ((growth / 100.0))) ** abs(time)))
        elif(time==0):
            return gdp_2019
        else:
            gdp = gdp_2019 * (((1 - ((growth / 100.0) )) ** abs(time)))
        return gdp

if __name__ == '__main__':
    main()
```