Τμήμα Ψηφιακών Συστημάτων

Ακαδημαϊκό Έτος: 2020-2021
Μάθημα: Ανάλυση Δεδομένων

Πανώριος Μιχαήλ Ε18127
Email: michaelpanorios@gmail.com

Περιεχόμενα

Σκοπός και λεπτομέρειες εργασίας

Σκοπός της εργασίας είναι η ανάλυση χρονοσειρών κρουσμάτων COVID-19 για διαφορετικές γεωγραφικές περιοχές ενός κράτους. Συγκεκριμένα, δίνεται σύνολο δεδομένων στο οποίο καταγράφεται ο αριθμός κρουσμάτων COVID-19 σε ημερήσια βάση για διαφορετικές περιοχές της Γερμανίας για περίοδο ενός έτους. Ουσιαστικά πρόκειται για πολλές χρονοσειρές – μία για κάθε περιοχή. Ο αριθμός κρουσμάτων είναι αθροιστικός, δηλαδή ο αριθμός που δίνεται στο αρχείο για μια ημερομηνία αφορά το σύνολο των κρουσμάτων μέχρι εκείνη την ημερομηνία. Ζητείται να εφαρμόσουμε τεχνικές ανάλυσης χρονοσειρών για τη μελέτη του συνόλου. Συγκεκριμένα ζητείται η κατασκευή ενός προβλεπτικού μοντέλου, η αξιολόγηση της ακρίβειας κ.ά.

Εργαλεία για την κατασκευή των χρονοσειρών

Για την εκπόνηση της εργασίας χρησιμοποίησα την Python η οποία είναι η καλύτερη γλώσσα για την ανάλυση δεδομένων, καθώς διαθέτει ένα πολύ εύχρηστο περιβάλλον και πληθώρα πακέτων και βιβλιοθηκών που επιταχύνουν την διαδικασία της ανάλυσης σε μεγάλα αρχεία, που εν τέλει απλοποιούν την μεθοδολογία και εξοικονομεί σημαντικό χρόνο σε σύγκριση με άλλες γλώσσες προγραμματισμού. Τέλος, οι βιβλιοθήκες που χρησιμοποίησα στο project είναι οι ακόλουθες:

- Matplotlib.pyplot: για την κατασκευή των γραφημάτων.
- Sklearn.metrics: για τον υπολογισμό των MSE και RMSE.
- Pandas: για την αναγνώριση του txt αρχείου.

Πλάνο εργασίας, προβλήματα και αντιμετώπιση τους

Για την εκπόνηση της εργασίας χρειάστηκε να δαπανήσω τις πρώτες ώρες διαβάζοντας τις σχετικές διαφάνειες καθώς και διάφορες άλλες πηγές του διαδικτύου, προκειμένου να κατανοήσω την μελέτη των χρονοσειρών αλλά και την δημιουργία των προβλεπτικών μοντέλων. Αφού ολοκλήρωσα τα παραπάνω προχώρησα στην κατασκευή του κώδικα. Αρχικά, προσπάθησα να κάνω parse το δοσμένο αρχείο txt και στην συνέχεια προσπάθησα να κάνω αφαίρεση όλων των ημερών με την χθεσινή του έτσι ώστε να εντοπίσω τα ημερήσια κρούσματα και όχι τα αθροιστικά τους. Στην συνέχεια, δημιούργησα τις μεθόδους ονομαστικά timeseries, mse_and_rmse και forecast. Θεώρησα πως επειδή το πλήθος των πόλεων είναι μεγάλο, μια καλή πρακτική ήταν να ανατρέχω το αρχείο στην main με βάση το id της κάθε πόλης προκειμένου η διαδικασία να αυτοματοποιηθεί για κάθε πόλη. Η λεπτομερής επεξήγηση βρίσκεται στην συνέχεια της τεχνικής αναφοράς. Τέλος, τα προβλήματα που αντιμετώπισα ήταν στην κατασκευή του προβλεπτικού μοντέλου καθώς χρησιμοποιήσα διάφορες τεχνικές και βιβλιοθήκες οι οποίες δεν έφερναν σωστά αποτελέσματα. Ωστόσο, η τεχνική που εν τέλει χρησιμοποιήσα μου έλυσε το πρόβλημα που αντιμετώπισα.

Ανάλυση του κώδικα και λεπτομερής επεξήγηση.

Ανοίγοντας το αρχείο υπάρχουν στις πρώτες γραμμές οι βιβλιοθήκες και τα πακέτα που χρησιμοποίησα για την κατασκευή του κώδικα.

```
import warnings
import matplotlib.pyplot as plt
import np as np
from sklearn.metrics import mean_squared_error
from pandas.tseries.offsets import DateOffset
from statsmodels.tsa.seasonal import seasonal_decompose
from tqdm.contrib import itertools
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
from pandas.plotting import autocorrelation_plot
import statsmodels.api as sm
import matplotlib
from pylab import rcParams
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
```

Παρακάτω με χρήση της βιβλιοθήκης matplotlib δίνω διάφορους παραμέτρους για την κατασκευή των γραφημάτων όπως πχ το μέγεθος των γραμμών x,y και το χρώμα της γραμμής.

```
# Settings for matplotlib.
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'
rcParams['figure.figsize'] = (18, 8)
```

Function main

```
def main(file):
    names = ['BAYERN', 'BADEN-WÄSTRTTEMBERG', 'BRANDENBURG', 'HAMBURG', 'BREMEN', 'MECKLENBURG-VORPOMMERN', 'HESSEN',
             'NIEDERSACHSEN', 'RHEINLAND-PFALZ', 'NORDRHEIN-WESTFALEN', 'SACHSEN', 'SAARLAND', 'SCHLESWIG-HOLSTEIN',
             'SACHSEN-ANHALT', 'THASTRINGEN']# Creating an array with all city names of text file
    df = pd.read_csv(file, header='infer', parse_dates=['date'])# Reading the data.txt using pandas read_csv method
    df["confirmedInfections"] = df.groupby('ID')['confirmedInfections'].diff(periods=1).fillna(df.confirmedInfections).astype(int)
    names_parser = 0 # Counter to parse correct city name for the methods
    for i in df.ID.unique():
        df_temp = df.groupby('ID').get_group(i)  # Group the dataframe of each ID
        timeseries(df_temp, names[names_parser])# Calling timeseries function
        names_parser += 1 # Counter to place the index to the correct city name
```

Η συνάρτηση main αποτελεί την καρδιά του κώδικα καθώς μέσω αυτής χωρίζω τα dataframes σε ομάδες ξεχωρίστα για κάθε πόλη με βάση το ID τους. Αρχικά δημιουργώ έναν πίνακα με όλα τα ονόματα των πόλεων που υπάρχουν στο αρχείο data.txt. Στην συνέχεια δημιουργώ μια μεταβλητή df στην οποία ορίζω με χρήση του pandas και συγκεκριμένα της μεθόδου read_csv τα στοιχεία του αρχείου με βάση το column date προκειμένου να μην χαθεί καμία ημερομηνία. Έπειτα, με την παράμετρο df[“confirmedInfections”] επιλέγω την στήλη confirmedInfections και όπου υπάρχει ίδιο ID αφαιρώ τα κρούσματα του επιλεγμένου date με τα χθεσινά (diff(periods=1)) και μετατρέπω τα στοιχεία του confirmedInfections από αθροιστικά σε ημερήσια ακέραιου τύπου (astype(int)). Η μεταβλητή names_counter υπάρχει προκειμένου να ανατρέχει των πίνακα names στην for επανάληψη. Δημιουργώ την for loop που ανατρέχει το αρχείο data.txt κατασκεύαζοντας ένα dataframe για κάθε ένα ξεχωριστό ID που υπάρχει. Μέσα στην επανάληψη στην μεταβλητή df_temp ορίζω το dataframe της κάθε πόλης ξεχωριστά και στην συνέχεια καλώ την function timeseries στην οποία δίνω ως παραμέτρους το dataframe της πόλης και το όνομα της με βάση των πίνακα names και τον μετρητή names_parser τον οποίο αυξάνω κατά 1 στην τελευταία γραμμή της main.

Function timeseries

```
def timeseries(dataframe, city_name):
    cols = ['ID', 'name']  # Creating columns to be dropped
    dataframe.drop(cols, axis=1, inplace=True)  # Dropping columns that I don't need
    dataframe.columns = ["date", "Covid cases"] # Renaming the columns for better handling
    dataframe.describe()
    dataframe.set_index('date', inplace=True) # Setting and choosing the date column as index
    dataframe.plot(figsize=(15, 6))  # Setting figure size

    # Plotting typical timeseries graph
    plt.title(city_name)  # Giving a title to graph based on city
    plt.ylabel("Covid cases")  # Giving a name to y-axis
    plt.xlabel("Date")  # Giving a name to x-axis
    plt.show()  # Plotting the graph


    #Autocorrelation plot
    autocorrelation_plot(dataframe['Covid cases'])  # Creating an autocorrelation plot of covid cases for e
    plt.show() # Showing the plot

    # Plotting trend, seasonality and resid
    # Period needs to be in a specific number based on the number of data we have in each frame
    decomposition = sm.tsa.seasonal_decompose(dataframe, model='additive', period=int(len(dataframe) / 2))
    decomposition.plot() # Applying decomposition formula to axes
    plt.show() # Showing the plot


    forecast(dataframe) # Calling the forecast function
```

Στην παραπάνω φωτογραφία φαίνεται η συνάρτηση timeseries η οποία παίρνει ως παραμέτρους την το dataframe της κάθε πόλης ξεχωριστά και το όνομα της πόλης που πρόκειται να αναλύσει. Για να δημιουργήσω τα γραφήματα των χρονοσειρών αρκεί να κρατήσω τις απαραίτητες στήλες. Για αυτό τον λόγο δημιουργώ έναν πίνακα cols με τις ονομασίες των στηλών που δεν χρειάζομαι και στην συνέχεια τις κάνω drop προκειμένου να έχω τις στήλες confirmedInfections και date. Έπειτα μετονομάζω τις στήλες date σε date και την confirmedInfections σε Covid cases για ευκολότερη χρήση. Προκειμένου να φέρω τα δεδομένα σε διαχειρίσιμη μορφή μετατρέπω την στήλη date σε index τύπου DateTimeIndex. Έπειτα περνάω παραμέτρους για το γράφημα όπως ονομασίες των αξόνων και μέγεθος γραφήματος και έπειτα το παρουσιάζω με την εντολή plt.show(). Έπειτα εμφανίζω ένα διαγνωστικό γράφημα autocolleration χρησιμοποιώντας ως παράμετρο την στήλη covid cases του dataframe. Τέλος, για την ανάλυση της χρονοσειράς χρησιμοποιώ την βιβλιοθήκη sm.tsa.seasonal_decompose στην οποία εμφανίζονται τα γραφήματα της τάσης, εποχικότητας και του θορύβου μαζί με την αρχική χρονοσειρά της πόλης συγκεντρωτικά. Κατά την χρήση της βιβλιοθήκης περνάω ως παραμέτρους:

- Dataframe: προκειμένου να πάρει τα στοιχεία ημερομηνίες και τα αντίστοιχα ημερήσια κρούσματα.
- Model='additive': για να εμφανίσει συγκεντρωτικά τα γραφήματα
- Period=int(len(dataframe)/2): ορίζω την μέγιστη περίοδο για τα γραφήματα ανάλογα το πλήθος εγγραφών που περιέχει το γράφημα προκειμένου να αποφύγω τυχόν ValueError καθώς και για αποδοτικότερη αναπαράσταση των γραφημάτων.

Τέλος, εφαρμόζω τα διαγνωστικά αποτελέσματα στους άξονες και κάνω plot το τελικό γράφημα και καλώ την συνάρτηση forecast.

Function forecast

Μία από τις πιο κοινές μεθόδους που χρησιμοποιούνται στην πρόβλεψη χρονοσειρών είναι γνωστή ως το μοντέλο ARIMA, που σημαίνει AutoregRessive Integrated Moving Average. Το ARIMA είναι ένα μοντέλο που μπορεί να προσαρμοστεί σε δεδομένα χρονοσειρών προκειμένου να κατανοήσει καλύτερα ή να προβλέψει μελλοντικά σημεία της σειράς. Υπάρχουν τρεις ξεχωριστοί ακέραιοι αριθμοί (p, d, q) που χρησιμοποιούνται για τα μοντέλα ARIMA. Εξαιτίας αυτού, τα μοντέλα ARIMA συμβολίζονται με τη σημείωση ARIMA (p, d, q). Αυτές οι τρεις παράμετροι μαζί αντιπροσωπεύουν την εποχικότητα, την τάση και τον θόρυβο σε σύνολα δεδομένων: Το p είναι το αυτόματο παλινδρομικό μέρος του μοντέλου. Μας επιτρέπει να ενσωματώσουμε την επίδραση των προηγούμενων τιμών στο μοντέλο μας. Το d είναι το ενσωματωμένο μέρος του μοντέλου. Αυτό περιλαμβάνει όρους στο μοντέλο που ενσωματώνουν το ποσό των διαφορών (δηλαδή τον αριθμό των προηγούμενων χρονικών σημείων προς αφαίρεση από την τρέχουσα τιμή) που ισχύουν για τις χρονοσειρές. Το q είναι ο κινητός μέσος όρος του μοντέλου. Αυτό μας επιτρέπει να ορίσουμε το σφάλμα του μοντέλου μας ως γραμμικό συνδυασμό των τιμών σφάλματος που παρατηρήθηκαν σε προηγούμενα χρονικά σημεία στο παρελθόν. Όταν αντιμετωπίζουμε τιμές όπως πχ ετήσιες, χρησιμοποιούμε το seasonal ARIMA, το οποίο χαρακτηρίζεται ως ARIMA (p, d, q) (P, D, Q, S). Εδώ, (p, d, q) είναι οι μη εποχιακές παράμετροι που περιγράφονται παραπάνω, ενώ (P, D, Q) ακολουθούν τον ίδιο ορισμό, αλλά εφαρμόζονται στο εποχιακό στοιχείο της χρονικής σειράς. Ο όρος S είναι η περιοδικότητα των χρονοσειρών (4 για τριμηνιαίες περιόδους, 12 για ετήσιες περιόδους, κ.λπ.). Όταν θέλουμε να ταιριάξουμε δεδομένα χρονοσειρών με ένα εποχιακό μοντέλο ARIMA, ο πρώτος μας στόχος είναι να βρούμε τις τιμές των ARIMA (p, d, q) (P, D, Q) s που βελτιστοποιούν μια μέτρηση. Υπάρχουν πολλές οδηγίες και βέλτιστες πρακτικές για την επίτευξη αυτού του στόχου, ωστόσο η σωστή παραμετροποίηση των μοντέλων ARIMA είναι αυτή που ακολούθησα για την πρόβλεψη. Για να εντοπίσω λοιπόν όλους τους πιθανούς συνδυασμούς παραμέτρων περιορίζω το διάστημα των p,d,q σε 0 ή 1. Στην συνέχεια υπολογίζω τους συνδυασμούς των τριάδων σε μία λίστα. Έπειτα, βρίσκω όλους τους πιθανούς συνδυασμούς του seasonal P,D,Q τους συγκεντρώνω και τους εκτυπώνω.

```
def forecast(dataframe):
    # Define the p, d and q parameters to take any value between 0 and 2
    p = d = q = range(0, 2)

    # Generate all different combinations of p, q and q triplets
    pdq = list(itertools.product(p, d, q))

    # Generate all different combinations of seasonal p, q and q triplets
    seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

    # Summarizing and printing the possible combinations for ARIMA and seasonal_roder
    print('Examples of parameter combinations for Seasonal ARIMA...')
    print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
    print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
    print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
    print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

Μπορούμε τώρα να χρησιμοποιήσουμε τις τριάδες παραμέτρων που ορίστηκαν παραπάνω για να αυτοματοποιήσουμε τη διαδικασία εκπαίδευσης και αξιολόγησης μοντέλων ARIMA σε διαφορετικούς συνδυασμούς. Στην στατιστική και το machine learning, αυτή η διαδικασία είναι γνωστή ως grid search, για την επιλογή μοντέλου. Κατά την αξιολόγηση και τη σύγκριση στατιστικών μοντέλων με διαφορετικές παραμέτρους, το καθένα μπορεί να κατατάσσεται το ένα στο άλλο με βάση το πόσο ταιριάζει τα δεδομένα ή την ικανότητά του να προβλέπει με ακρίβεια τα μελλοντικά σημεία δεδομένων. Θα χρησιμοποιήσουμε την τιμή AIC (Akaike Information Criterion), που συνδυάζεται με μοντέλα ARIMA και την βιβλιοθήκη statsmodels. Το AIC μετρά πόσο καλά ένα μοντέλο ταιριάζει στα δεδομένα, λαμβάνοντας υπόψη τη συνολική πολυπλοκότητα του μοντέλου. Σε ένα μοντέλο που ταιριάζει πολύ καλά στα δεδομένα, ενώ χρησιμοποιεί πολλές δυνατότητες, θα έχει μεγαλύτερη βαθμολογία AIC από ένα μοντέλο που χρησιμοποιεί λιγότερα χαρακτηριστικά για να επιτύχει την ίδια ευελιξία. Επομένως, μας ενδιαφέρει να βρούμε το μοντέλο που αποδίδει τη χαμηλότερη τιμή AIC. Το κομμάτι κώδικα που ακολουθεί επαναλαμβάνεται μέσω συνδυασμών παραμέτρων και χρησιμοποιεί τη συνάρτηση SARIMAX από statsmodels για να ταιριάζει στο αντίστοιχο εποχιακό μοντέλο ARIMA. Εδώ, το όρισμα order καθορίζει τις παραμέτρους (p, d, q), ενώ το όρισμα seasonal_order καθορίζει το εποχιακό στοιχείο (P, D, Q, S) του εποχιακού μοντέλου ARIMA. Μετά την τοποθέτηση κάθε μοντέλου SARIMAX (), ο κώδικας εκτυπώνει την αντίστοιχη βαθμολογία AIC.

```
AIC_list = pd.DataFrame({}, columns=['param', 'param_seasonal', 'AIC'])

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            # Searching for all order and seasonal order combinations
            mod = sm.tsa.statespace.SARIMAX(dataframe['Covid cases'],
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)


            results = mod.fit()
            # Printing each combination that for loop finds
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
            # Creating a temp an applying to that (x,y,z)|(a,b,c,d)|AIC:(float number) combination
            temp = pd.DataFrame([[param, param_seasonal, results.aic]], columns=['param', 'param_seasonal', 'AIC'])
            AIC_list = AIC_list.append(temp, ignore_index=True)
            del temp

        except:
            continue
```

Προκειμένου να κρατάω όλους τους συνδυασμούς του order x seasonal_order x AIC δημιουργώ μια λίστα με τις αντίστοιχες στήλες και την ονομασία τους. Με δύο for loops που διατρέχουν τις πιθανές τριάδες pdq και PDQ υπολογίζω την τιμή AIC σε κάθε περίπτωση με την βοήθεια του εργαλείου SARIMAX. Αποθηκεύω τα αποτελέσματα και τα εμφανίζω σε κάθε περίπτωση που τελειώνει ένα τμήμα του dataframe της εκάστοτε πόλης. Τέλος αποθηκεύω τους συνδυασμούς σε μια μεταβλητή temp και την εισάγω στην λίστα που δημιούργησα.

```
# Find minimum value in AIC
m = np.amin(AIC_list['AIC'].values)
# Find index number for lowest AIC
l = AIC_list['AIC'].tolist().index(m)
# Presenting the order and seasonal order parameters based on minimum AIC
Min_AIC_list = AIC_list.iloc[l, :]

# Applying the combination of the minimum AIC we found to order and seasonal order
mod = sm.tsa.statespace.SARIMAX(dataframe['Covid cases'],
                                order=Min_AIC_list['param'],
                                seasonal_order=Min_AIC_list['param_seasonal'],
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()
```

Από την στιγμή που έχω αποθηκέυσει όλους τους πιθανούς συνδυασμούς στην AIC_list όπως ανέφερα και πριν πρέπει να εντοπίσω τον μικρότερο αριθμό AIC που θα μου δώσει το πιο ακριβή συνδυασμό παραμέτρων σε σχέση με τους υπόλοιπους. Με την εντολή np.amin(AIC_list[AIC].values()) σε συνδυασμό με την μεταβλητή l που προσδιορίζει το index που ο μικρότερος AIC εντοπίζεται πετυχαίνω τον σκοπό μου. Στην συνέχεια εμφανίζω τον συνδυασμό των παραμέτρων order x seasonal_order x AIC και το αποθηκεύω σε μια νέα λίστα Min_AIC_list. Τέλος, χωρίζω τις στήλες που περιέχουν τις παραμέτρους και τις αξιοποιώ στο υπολογιστικό μοντέλο. Πλέον μπορώ να προχωρήσω στην ανάλυση αυτού του συγκεκριμένου μοντέλου σε βάθος. Το σύνολο πληροφοριών που προκύπτει από την έξοδο του SARIMAX επιστρέφει σημαντικό αριθμό πληροφοριών:

```
CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1          0.4413      0.038     11.758      0.000       0.368       0.515
ar.S.L12      -0.1893      0.031     -6.092      0.000      -0.250      -0.128
ma.S.L12      -1.0000      0.041    -24.405      0.000      -1.080      -0.920
sigma2       2.22e+05   1.85e-07    1.2e+12      0.000    2.22e+05    2.22e+05
==============================================================================
### Min_AIC_list ###
param                (0, 1, 1)
param_seasonal    (1, 1, 1, 12)
AIC                5266.209899
```

Η στήλη coef δείχνει το βάρος (δηλαδή τη σημασία) κάθε δυνατότητας και τον τρόπο με τον οποίο το καθένα επηρεάζει τις χρονοσειρές. Το P>|z| Η στήλη μας ενημερώνει για τη σημασία κάθε βάρους χαρακτηριστικών. Εδώ, κάθε βάρος έχει τιμή p χαμηλότερη ή κοντά στο 0,05, επομένως είναι λογικό να διατηρούνται όλα αυτά στο μοντέλο μας. Κατά την τοποθέτηση εποχιακών μοντέλων ARIMA, είναι σημαντικό να εκτελέσω διαγνωστικά μοντέλα για να βεβαιωθώ ότι δεν έχει παραβιαστεί καμία από τις υποθέσεις του μοντέλου. Το αντικείμενο plot_diagnostics μας επιτρέπει να δημιουργήσουμε γρήγορα τα διαγνωστικά και να διερευνήσουμε τυχόν ασυνήθιστη συμπεριφορά. Αυτό επιτυγχάνεται με την εντολή

```
# Plotting diagnostics to check if our parameters will be precise
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

Έχουμε αποκτήσει ένα μοντέλο για τις χρονολογικές μας σειρές που μπορούν πλέον να χρησιμοποιηθούν για την παραγωγή προβλέψεων. Ξεκινάμε συγκρίνοντας τις προβλεπόμενες τιμές με τις πραγματικές τιμές των χρονοσειρών, οι οποίες θα μας βοηθήσουν να κατανοήσουμε την ακρίβεια των προβλέψεών μας. Τα χαρακτηριστικά get_prediction () και conf_int () μας επιτρέπουν να λάβουμε τις τιμές και τα σχετικά διαστήματα εμπιστοσύνης για τις προβλέψεις των χρονοσειρών.

```
# Setting the time we decide to start forecasting
pred = results.get_prediction(start=pd.to_datetime('2020-11-24'), dynamic=False)
pred_ci = pred.conf_int()
ax = dataframe['2020':].plot(label='observed')
# Apply the orange line to see how applies before forecast
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Covid cases')
plt.legend()
plt.show()
```

Με τον παραπάνω κώδικα ορίζω τις προβλέψεις να ξεκινούν από τον 24 Νοέμβρη. Το dynamic=False διασφαλίζει ότι παράγουμε προβλέψεις ενός βήματος μπροστά, πράγμα που σημαίνει ότι οι προβλέψεις σε κάθε σημείο δημιουργούνται χρησιμοποιώντας το πλήρες ιστορικό έως εκείνο το σημείο. Από εκεί και έπειτα ορίζω κάποιες παραμέτρους για το γράφημα και στην συνέχεια περνάω την φόρμα στο γράφημα με το plt.legend() και στην συνέχεια το κάνω show().

```
# Applying truth and forecasted values to variables
y_forecasted = pred.predicted_mean
y_truth = dataframe['2020-11-24':]

# Calling mse_and_rmse function
mse_and_rmse(y_truth, y_forecasted)
```

Εδώ αποθηκεύω τις τιμές που έγιναν πρόβλεψη στην μεταβλητή y_forecasted και στην συνέχεια αποθηκεύω στην y_truth τις πραγματικές τιμές πριν το forecast. Τέλος καλώ την function mse_and_rmse που θα επεξηγήσω παρακάτω.

Το αντικείμενο get_forecast μπορεί να υπολογίσει τις προβλεπόμενες τιμές για έναν καθορισμένο αριθμό βημάτων μπροστά.

```
# Get forecast 30 steps (one month) ahead in future
pred_uc = results.get_forecast(steps=30)

# Get confidence intervals of forecasts
pred_ci = pred_uc.conf_int()
```

Τόσο οι προβλέψεις όσο και το σχετικό διάστημα εμπιστοσύνης που έχουμε δημιουργήσει μπορούν τώρα να χρησιμοποιηθούν για να κατανοήσουμε περαιτέρω τις χρονοσειρές και να προβλέψουμε τι να περιμένουμε. Οι προβλέψεις μας δείχνουν ότι οι χρονοσειρές αναμένεται να συνεχίσουν με σταθερό ρυθμό σε χαμηλά επίπεδα. Καθώς προβλέπουμε περαιτέρω στο μέλλον, είναι φυσικό να είμαστε λιγότερο σίγουροι για την ακρίβεια. Αυτό αντικατοπτρίζεται από τα διαστήματα εμπιστοσύνης που δημιουργούνται από το μοντέλο μας, τα οποία μεγαλώνουν καθώς προχωράμε περισσότερο στο μέλλον.

```
# Final settings before the forecast
ax = dataframe.plot(label='observed', figsize=(20, 15))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Covid cases')

plt.legend()
plt.show()
```

Πριν εμφανίσω το τελευταίο γράφημα ορίζω τους παραμέτρους των γραφημάτων όπως και σε όλα τα πρηγούμενα γραφήματα. Δίνω τίτλους τόσο στους άξονες όσο και στο γράφημα “Forecast”. Τέλος, συνέχεια περνάω την φόρμα στο γράφημα με το plt.legend() και στην συνέχεια το κάνω show().

Function mse_and_rmse

Με την συνάρτηση mse_and_rmse υπολογίζω τα μετρικά MSE & RMSE. Ως ορίσματα δέχονται το y_actual και y_predicted που περιλαμβάνουν τις πραγματικές τιμές και τις προβλεπόμενες αντίστοιχα.

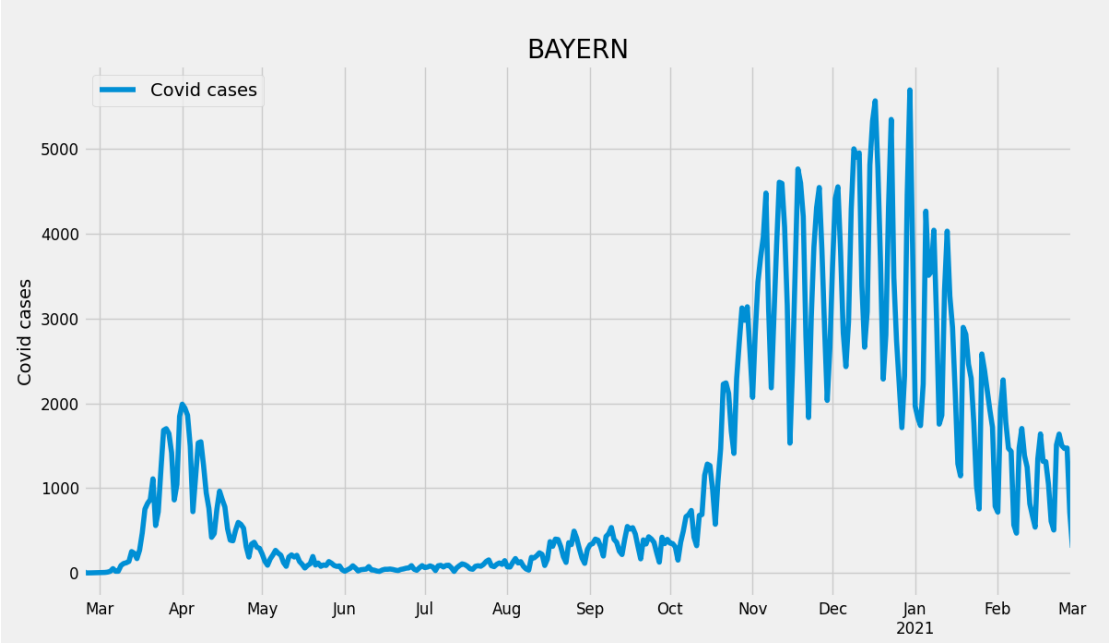
```
def mse_and_rmse(y_actual,y_predicted):
    # Calculating mse and rmse with sklearn.metrics library
    mse = mean_squared_error (y_actual, y_predicted, squared=True)
    print("MSE:", mse)
    rmse = mean_squared_error(y_actual, y_predicted, squared=False)
    print("RMSE:", rmse)
```

Τον υπολογισμό των μετρικών τον υλοποιώ με την βοήθεια της βιβλιοθήκης sklearn.metrics. Το μόνο που αλλάζει είναι η παράμετρος squared από True σε False ανάλογα τον τύπο και το αν είναι υψωμένος στο τετράγωνο. Τέλος, μετά τον κάθε υπολογισμό εμφανίζω στην κονσόλα τα υπολογισμένα MSE και RMSE αντίστοιχα.

Ανάλυση της πόλης Bayern

Αφού ανέλυσα τον κώδικα χρειάζεται να δούμε και τα αποτελέσματα τα οποία προκύπτουν. Ο κώδικας λειτουργεί για όλο το πλήθος των πόλεων ωστόσο επειδή ο αριθμός είναι μεγάλος θα επεξηγηθούν τα διαγράμματα της πόλης Bayern. Κάνοντας run το αρχείο covidcases.py ο κώδικας αρχικά κάνει τα πρώτα βήματα προκειμένου να παρουσιάσει την απλή χρονοσειρά κρουσμάτων η οποία έχει στον άξονα χ τις ημερομηνίες και στον άξονα y τα κρούσματα.

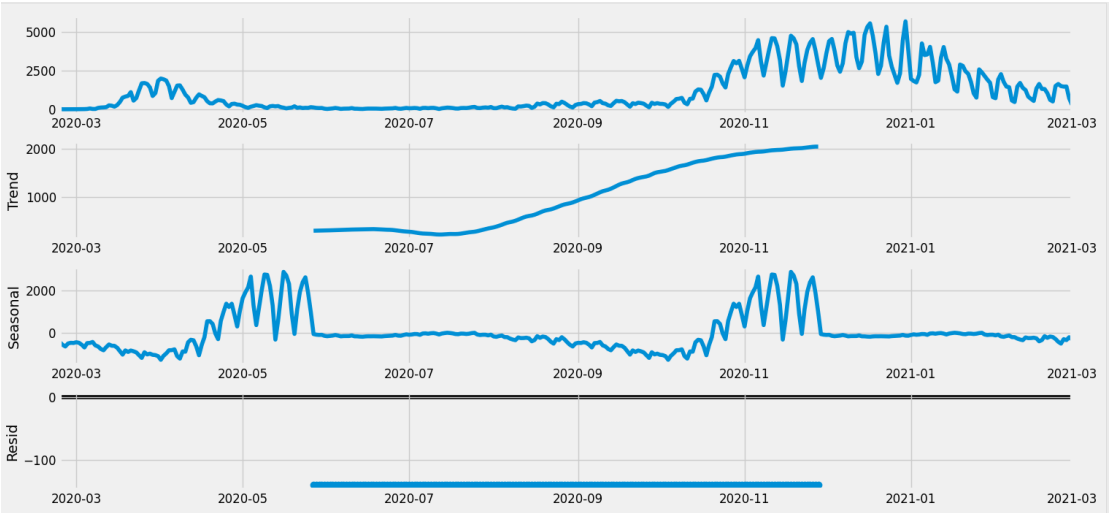
Χρονοσειρά της πόλης



Στο παραπάνω διάγραμμα βλέπουμε την απεικόνιση των ημερήσιων κρουσμάτων. Παρατηρούμε μια μικρή αύξηση από τα μέσα Μαρτίου εώς μέσα Απριλίου. Στην συνέχεια

τα κρούσματα φαίνεται να έχουν μια ισορροπία χωρίς μεγάλες αυξομειώσεις. Όμως, παρατηρείται μια απότομη άνοδος των κρουσμάτων από τον Οκτώβρη παίρνοντας μέγιστη τιμή τον Ιανουάριο του 2021. Στην συνέχεια τα κρούσματα μειώνονται ωστόσο συνεχίζουν τα παρατηρούνται αρκετές αυξομειώσεις.

Τάση, εποχικότητα και resid

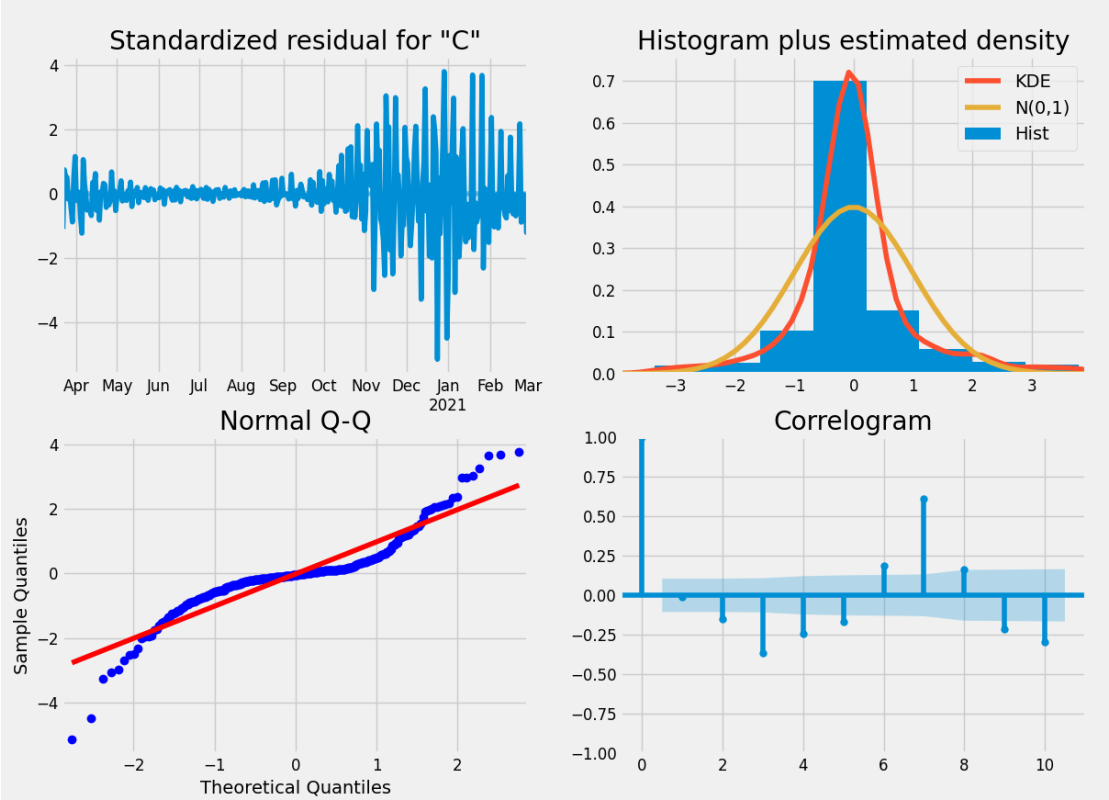


Στο παραπάνω γράφημα εμφανίζεται η χρονοσειρά, η τάση, η εποχικότητα και το resid κατά σειρά της πόλης Bayern. Παρατηρούμε ότι η τάση είναι γραμμική καθώς τα κρούσματα αυξάνονται με σταθερό ρυθμό και συγκεκριμένα η καμπή της ξεκινά από το σημείο όπου τον 9ο μήνα τα κρούσματα αρχίζουν και ξεπερνούν τα 1000 ημερήσια. Τα δεδομένα εμφανίζουν βραχυπρόθεσμη και μακροπρόθεσμη εποχικότητα. Το resid φαίνεται να παίρνει αρνητικές τιμές. Δεν μπόρεσα να βρω κάποιον τρόπο ώστε να το βελτιώσω. Όμως αναζήτησα την σχέση του resid με τον θόρυβο και το σφάλμα.

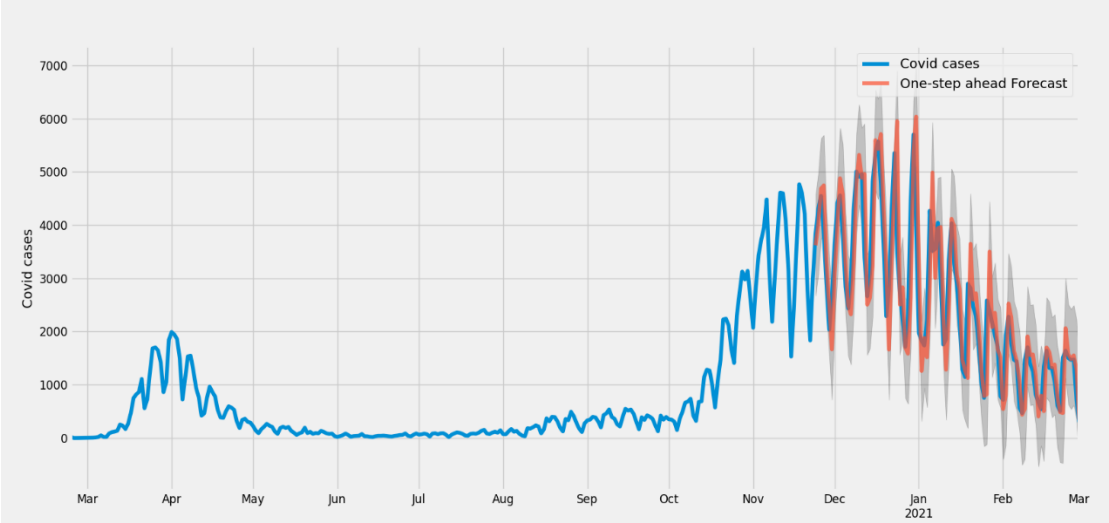
- το resid είναι η διαφορά μεταξύ του πραγματικού φαινομένου που μελετάται και του μοντέλου που χρησιμοποιείται για να το περιγράψει.
- ο θόρυβος είναι το μέρος του resid που δεν είναι εφικτό να μοντελοποιηθεί με οποιοδήποτε άλλο μέσο εκτός από μια καθαρά στατιστική περιγραφή.
- το error είναι εκείνο το στοιχείο του resid που παραμένει μετά τον υπολογισμό του θορύβου. σύμφωνα με τους παραπάνω ορισμούς: α) ο θόρυβος και το σφάλμα δεν συσχετίζονται β) τα resid μπορούν να μειωθούν είτε μειώνοντας τον θόρυβο είτε μειώνοντας το error. γ) το σφάλμα μπορεί να μειωθεί μόνο βελτιώνοντας το μοντέλο . Ωστόσο, ο θόρυβος μπορεί να μειωθεί είτε βελτιώνοντας τη διαδικασία μέτρησης είτε βελτιώνοντας την ακρίβεια του μοντέλου.

Διαγνωστικά βήματα πριν την πρόβλεψη

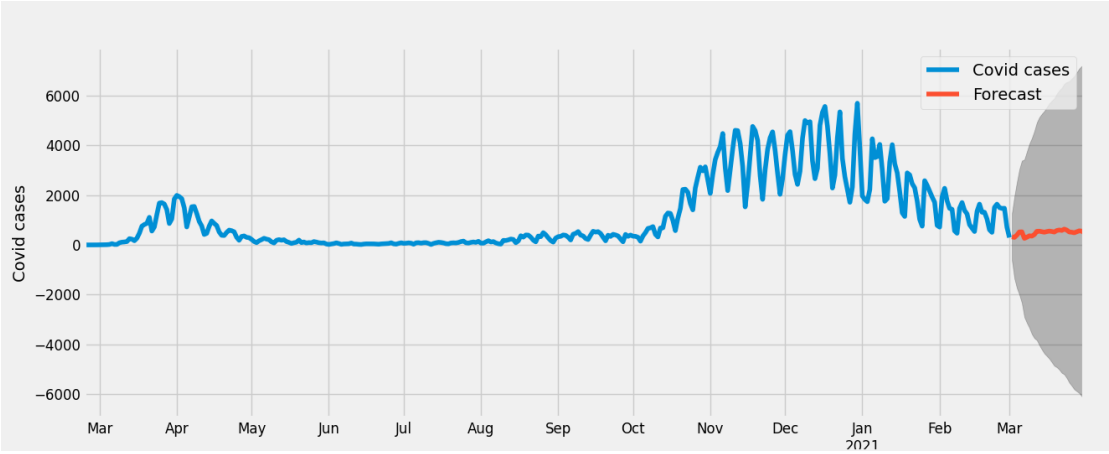
Στην συνέχεια προχωρόντας στην πρόβλεψη της χρονοσειράς για τον επόμενο έναν μήνα εφαρμόζω κάποια διαγνωστικά.



Κύριο μέλημά μας είναι να διασφαλίσουμε ότι τα υπολείμματα (resids) του μοντέλου μας δεν συσχετίζονται και κανονικά κατανέμονται με μηδενικό μέσο όρο. Εάν το εποχικό μοντέλο ARIMA δεν ικανοποιεί αυτές τις ιδιότητες, είναι μια καλή ένδειξη ότι μπορεί να βελτιωθεί περαιτέρω. Σε αυτήν την περίπτωση, τα διαγνωστικά των μοντέλων μας υποδηλώνουν ότι τα υπολείμματα του μοντέλου κατανέμονται κανονικά με βάση τα ακόλουθα: Στην επάνω δεξιά γραφική παράσταση, βλέπουμε ότι και οι δύο γραμμές με κανονική κατανομή με μέση τιμή 0 και τυπική απόκλιση 1 . Αυτή είναι μια καλή ένδειξη ότι τα υπολείμματα κατανέμονται κανονικά. Το qq-plot κάτω αριστερά δείχνει ότι η ταξινομημένη κατανομή των υπολειμμάτων (μπλε κουκκίδες) ακολουθεί κατά ένα ικανοποιητικό ποσοστό τη γραμμική τάση των δειγμάτων που λαμβάνονται από μια τυπική κανονική κατανομή με N (0, 1). Και πάλι, αυτή είναι μια ισχυρή ένδειξη ότι τα υπολείμματα κατανέμονται κανονικά. Από την γραφική παράσταση αυτόματης συσχέτισης κάτω δεξιά, η οποία δείχνει ότι τα υπολείμματα των χρονοσειρών έχουν συσχέτιση. Αυτές οι παρατηρήσεις μας οδηγούν στο συμπέρασμα ότι το μοντέλο μας παράγει μια ικανοποιητική εφαρμογή που θα μπορούσε να μας βοηθήσει να κατανοήσουμε τα δεδομένα των χρονοσειρών μας και να προβλέψουμε μελλοντικές τιμές. Αν και έχουμε ικανοποιητική εφαρμογή, ορισμένες παράμετροι του εποχιακού μοντέλου ARIMA θα μπορούσαν να αλλάξουν για να βελτιώσουν την εφαρμογή του μοντέλου. Για παράδειγμα, με το grid search θεωρούμε μόνο περιορισμένο σύνολο, επομένως ενδέχεται να βρούμε καλύτερα μοντέλα εάν διευρύνουμε την αναζήτηση.



Στο επόμενο βήμα παρατηρούμε την πορτοκαλί γραμμή η οποία ξεκινά να εφάπτεται από την ημερομηνία που της έχουμε ορίσει στον κώδικα. Το γκρι πεδίο δείχνει τις πιθανές διαφορετικές τιμές που θα μπορούσαν οι προβλεπτικές τιμές να πάρουν. Όμως, όπως βλέπουμε η πορτοκαλί γραμμή ακολουθεί την μπλέ γραμμή που προσδιορίζει την τα ημερήσια κρούσματα αρκετά ικανοποιητικά. Αποτελεί λοιπόν μια καλή ένδειξη ότι το προβλεπτικό διάγραμμα που ακολουθεί για τον επόμενο έναν μήνα και συγκεκριμένα τον Απρίλη θα είναι επαρκές.



Όπως περιμέναμε λοιπόν η πορτοκαλί γραμμή ξεκινάει από το σημείο όπου η μπλέ με τα ημερήσια κρούσματα σταμάτησε. Η πρόβλεψη για τις επόμενες 30 ημέρες δείχνει μια σταθερή κατανομή των κρουσμάτων χωρίς αυτό να σημαίνει ότι τους επόμενους μήνες δεν θα ξανά αυξηθούν.

Τέλος, τα αποτελέσματα που ο κώδικας μου εντόπισε μετά το grid search καθώς και το MSE και RMSE:

```
### Min_AIC_list ###
param                (0, 1, 1)
param_seasonal    (1, 1, 1, 12)
AIC                5266.209899
Name: 31, dtype: object
MSE: 574104.7648040811
RMSE: 757.6970138545361
```

Επίλογος

Η πρώτη επαφή με την ανάλυση των δεδομένων και συγκεκριμένα με τις χρονοσειρές ήταν πολύ ενδιαφέρουσα. Διάβασα πληθώρα άρθρων και αρκετά βιβλιογραφικά θέματα τόσο για τον χειρισμό των dataframes ενός csv αρχείου όσο και για τους διάφορους τρόπους πρόβλεψης. Για το data αρχείο που είναι σε μορφή txt δεν το άλλαξα σε csv διότι η δομή του ήταν comma-seperated values. Τέλος, λόγω του grid search που διαρκεί κατά μέσο όρο 94 δευτερόλεπτα ανάλογα τον όγκο του dataframe και το μεγάλο πλήθος των πόλεων στον φάκελο θα προσθέσω υποφακέλους της κάθε πόλης με τα αντίστοιχα διαγράμματα.