**Progetto Cyber Physical System**

**Anno accademico 2020/2021**

**Matteo Fresta e Matteo Pastorino Ghezzi**



**Studio e analisi algoritmi di previsione basati su serie temporali tramite l’utilizzo di dati relativi al virus SARS-CoV-2**

1. Introduzione

Il presente elaborato è stato redatto con l’obiettivo di studiare e analizzare algoritmi predittivi basati su serie temporali.

Dopo uno studio dei dati forniti dalla repository ufficiale della Protezione Civile **[1]** e aver verificato graficamente tutte le informazioni, abbiamo creato un modello tramite la regressione lineare da cui siamo partiti per fare una previsione e valutare l’accuratezza di essa.

Successivamente, siamo passati ad algoritmi basati esclusivamente su serie temporali ossia la Exponential Smoothing e la SARIMA studiandone nel dettaglio tutte le componenti e andando a verificare l’influenza di ogni parametro sulla previsione finale.

Un altro passo fondamentale è stato l’ampliamento del dataset per studiare come la quantità di dati influenzi la nostra predizione e verificarne l’accuratezza.

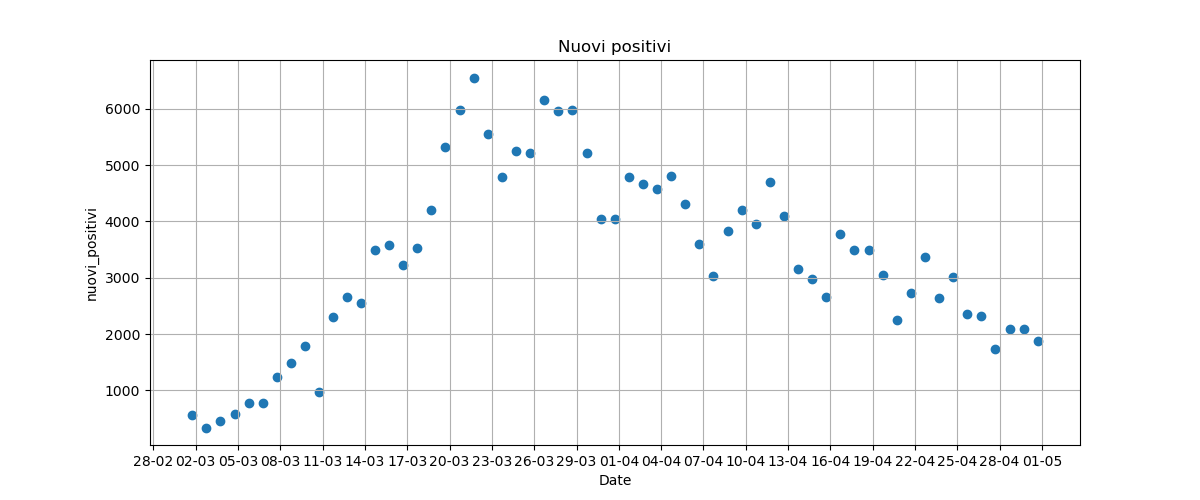
Ulteriori obiettivi che ci siamo preposti sono stati quelli di studiare il dataset ricavando tramite operazioni su esso dei dati non esplicitamente indicati, come fatto nel calcolo giornaliero totale del numero di vaccini somministrati.

Infine, per convalidare quanto appreso nel nostro studio abbiamo applicato tutti gli algoritmi studiati su un dataset completamente diverso **[2]**, successivamente abbiamo studiato l’accuratezza delle varie predizioni, traendo delle conclusioni.

1. Previsione tramite modello con regressione lineare

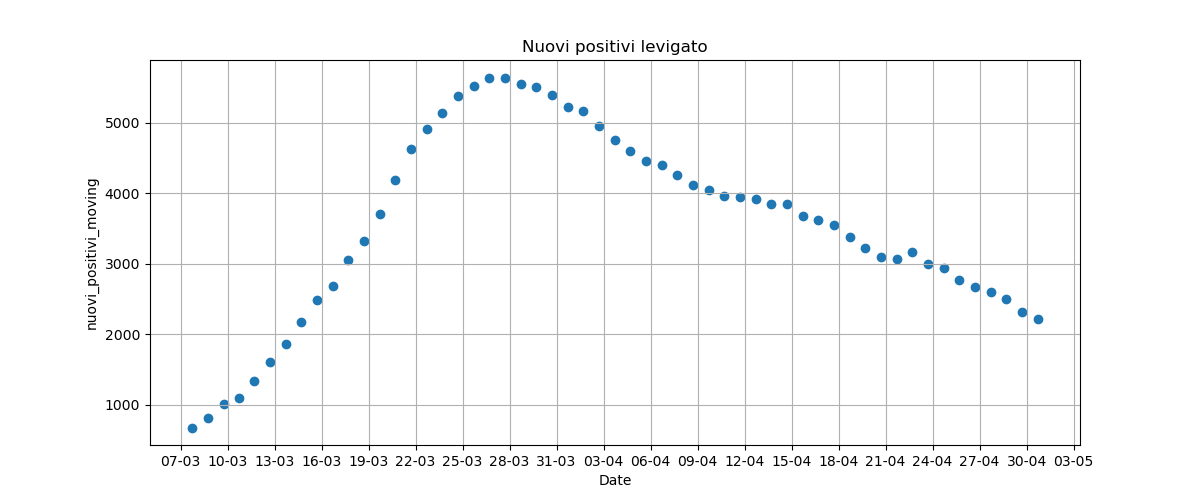
Ottenuti i dati dalla repository ufficiale della Protezione Civile, è stato deciso di utilizzare solo alcune delle informazioni messe a disposizione; nello specifico consideriamo solo il numero di positivi giornaliero (*nuovi\_positivi*), i deceduti (*deceduti*) e i tamponi effettuati (*tamponi*). Per queste due ultime variabili, è stata applicata la funzione di Pandas *diff* per avere l’incremento giornaliero di entrambe; il risultato è stato poi salvato in *diff\_deceduti* e *diff\_tamponi*.

Per verificare la corretta lettura del dataset, abbiamo visualizzato su grafico il numero di positivi giornaliero:

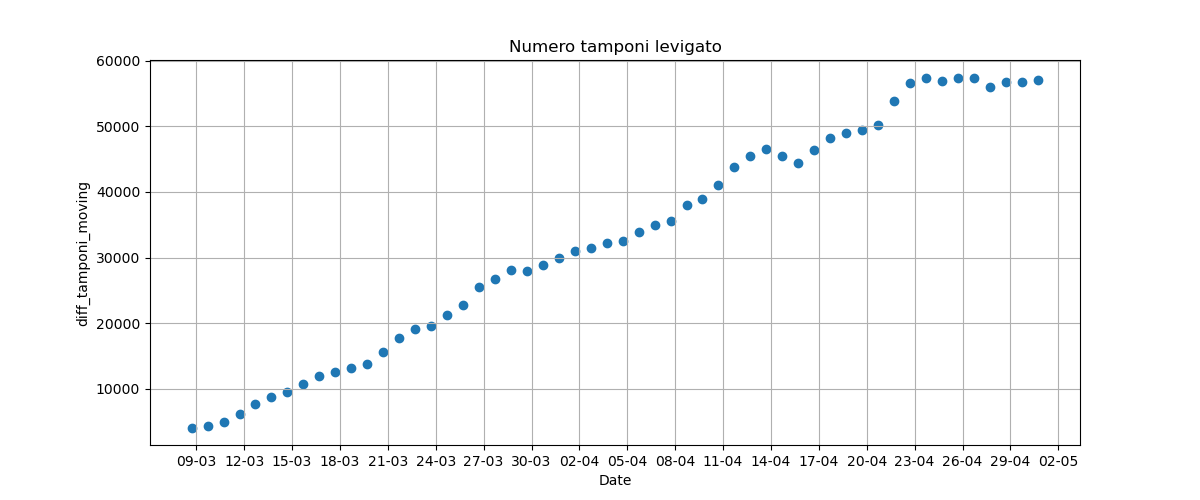


È possibile notare un pattern ricorrente nel grafico sovrastante: durante il weekend il numero di tamponi effettuati (e di conseguenza il numero di positivi) è sempre minore e questo aspetto si ripercuote sui dati del lunedì e del martedì. Per ovviare a ciò, nel grafico seguente, è calcolata la media mobile su 7 giorni: in questo modo, i dati sono ‘levigati’, riducendo le anomalie del weekend. L’implementazione su Python è stata resa possibile grazie ad una funzione di Pandas chiamata *rolling* che presenta un parametro *window*, da noi fissato a 7 (giorni).

Il grafico sottostante rappresenta il trend dei nuovi casi ‘pesati’ su un periodo di 7 giorni:



Come abbiamo già detto il numero di positivi è strettamente correlato alla quantità di tamponi effettuati. Visualizzando i dati del numero di tamponi giornalieri si può notare dal grafico seguente come questi sono aumentati in maniera significativa, ripercuotendosi sulla variabile *nuovi\_positivi*.

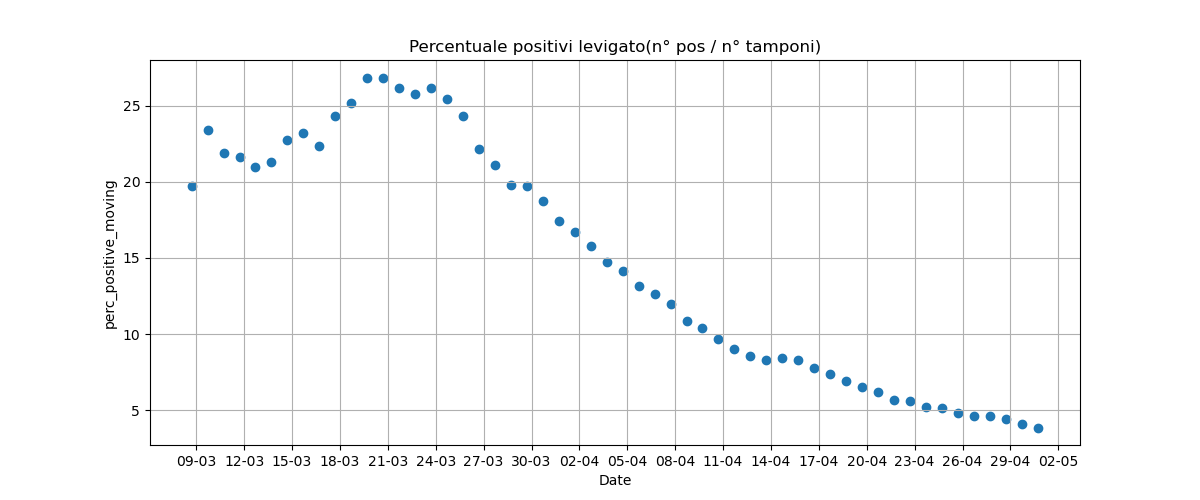


Vista la correlazione tra numero di nuovi positivi e quantità di tamponi effettuati giornalmente, il grafico del rapporto positivi/tamponi può darci un’informazione più significativa sulla situazione del virus nel Paese: è stata calcolata dunque la percentuale tra *nuovi\_positivi\_levigato* e *numero\_tamponi\_levigato* (in termine tecnico definito tasso di positività) per avere un trend più significativo da valutare, correggendo le fluttuazioni del weekend e tenendo in considerazione il numero di test totali.

Sebbene la variabile *perc\_positive* possa sembrare la più adatta per eseguire una predizione, in realtà esistono fattori esterni che ci suggeriscono di non utilizzarla: un aumento lineare del numero di tamponi, per esempio, porterebbe la percentuale a valori molto bassi perdendo informazioni utili sul numero di positivi giornalieri riscontrati.

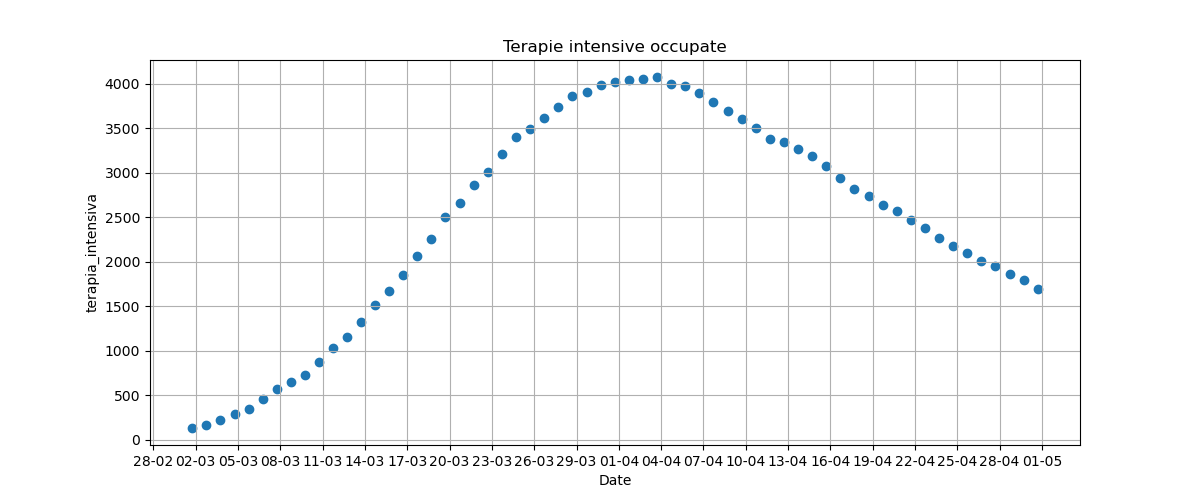
L’aumento del numero di test effettuati si può associare al progresso delle tecnologie che hanno migliorato e accelerato il processamento dei tamponi nel tempo, tenendo conto inoltre che i dati sui nuovi positivi, soprattutto ad inizio pandemia, possono riferirsi a test condotti nei giorni precedenti.

Di seguito il tasso di positività giornaliero levigato:

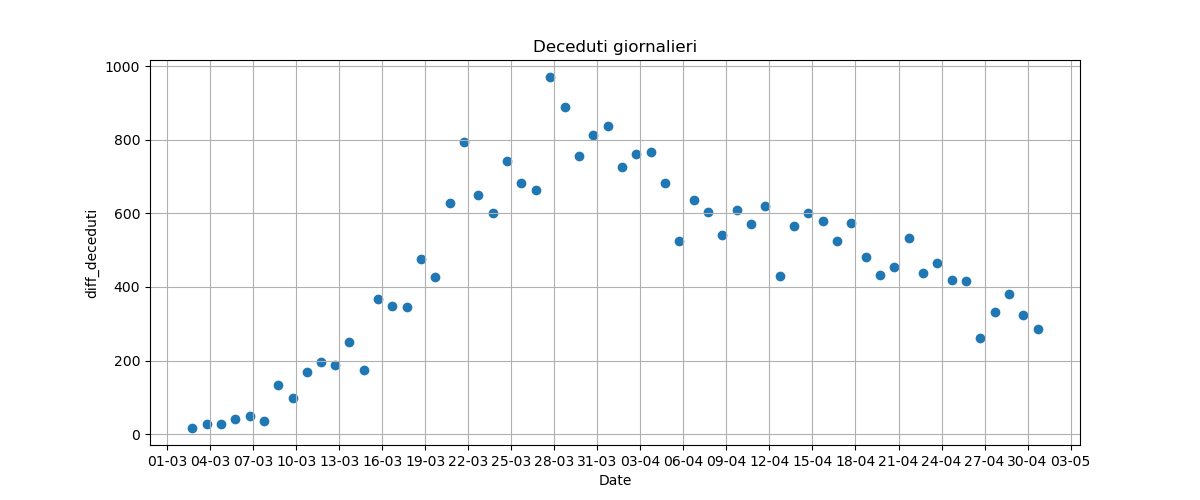


Continuando nell’analisi dei dati, abbiamo trovato variabili di particolare interesse statistico non correlati al numero di tamponi effettuati e perciò più stabili ed esenti dalle fluttuazioni del fine settimana: le terapie intensive occupate e il numero giornaliero di deceduti.

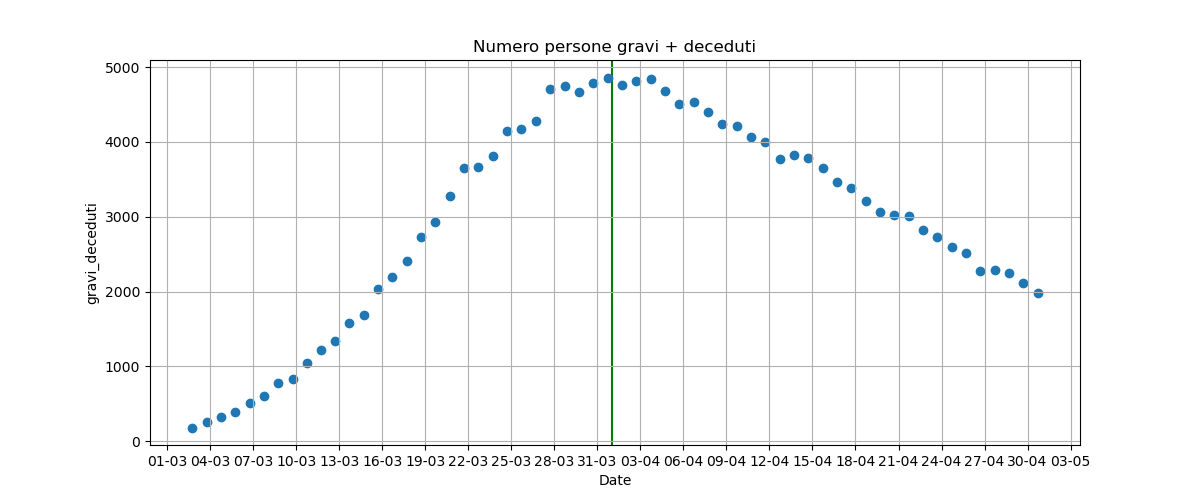
Nel grafico sottostante anche senza nessuna levigatura l’andamento terapie intensive occupate segue un trend molto lineare senza picchi dovuti a variazioni esterne.



Si può pensare che al diminuire delle terapie intensive la situazione sia in miglioramento; per verificare ciò, consideriamo anche il numero di deceduti giornaliero. Quest’ultimo dato è più ‘grezzo’ rispetto al precedente ma comunque significativo per la nostra analisi:



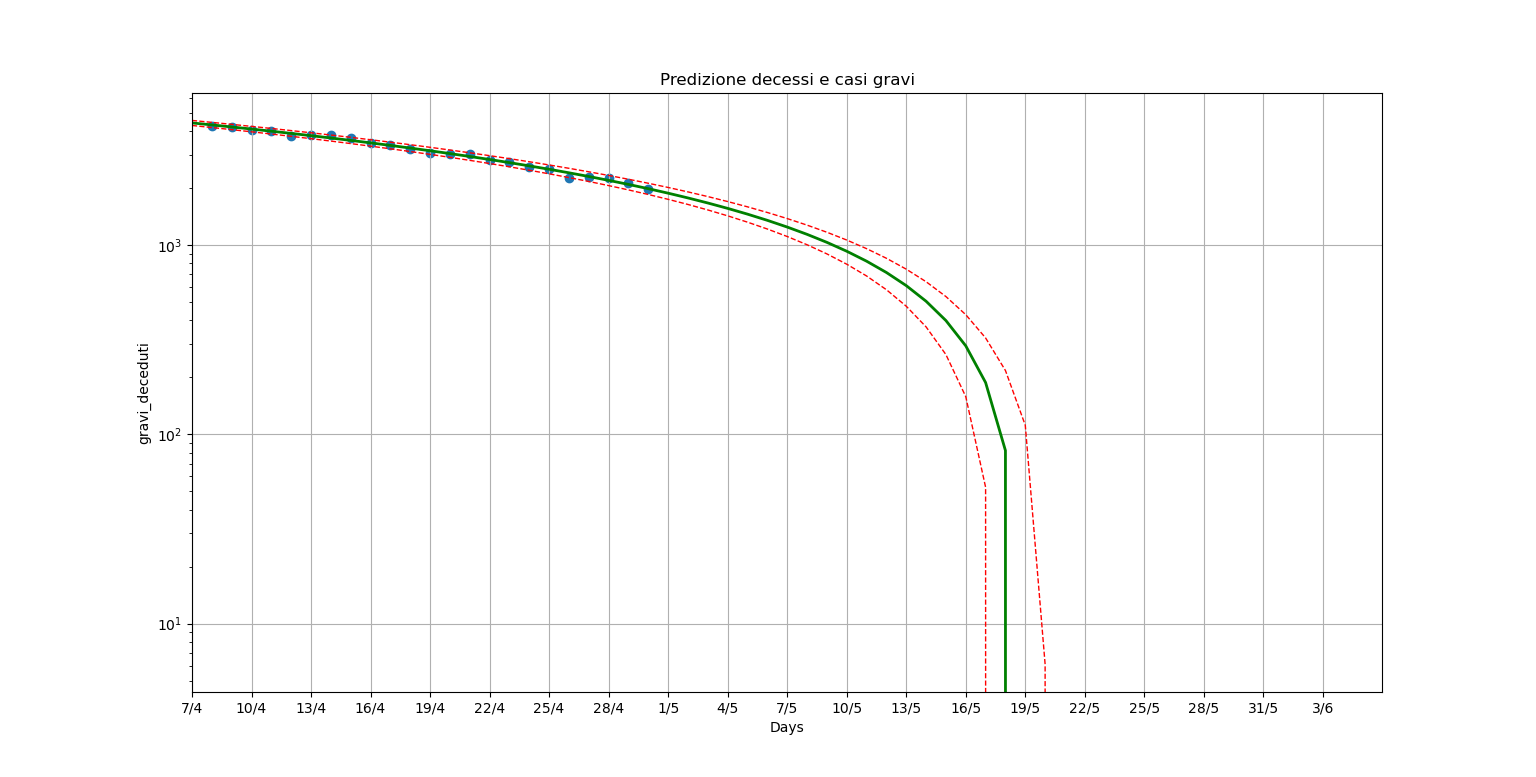
Mettendo insieme queste due variabili, si ha la certezza di non avere errori di acquisizione, in quanto sono dati acquisiti più cautamente e attentamente; inoltre rispecchiano di più la situazione nel Paese.



Nel grafico è stato evidenziato anche il picco massimo, dal quale è poi cominciato il trend discendente. A partire da questa nuova variabile (*gravi\_deceduti*), è possibile costruire un modello di regressione lineare e addestrarlo a partire dal 1° Aprile, ovvero dopo il picco. La regressione lineare è relativamente semplice da implementare ma semplifica eccessivamente i nostri dati. Per avere una valutazione sul modello è utilizzata la funzione *score()* che fornisce tramite i coefficiente R2 una stima sulla bontà del *fitting*. Nella regressione, R2 indica quanto i dati predetti sono vicini ai dati reali. Più R2 è vicino ad 1, più possiamo fidarci del nostro modello.

Per eseguire la predizione, è utilizzata la funzione *predict()*; inoltre teniamo conto del massimo errore commesso dal modello con *max\_error()*, errore che andremo rispettivamente a sommare e sottrarre per ottenere una tolleranza nella predizione.

Dopo aver calcolato in *y\_pred\_linear* il numero di persone gravi + deceduti dei prossimi 60 giorni e aver convertito l’asse delle ascisse nel formato *datetime* per una miglior visualizzazione dei dati, il risultato della predizione è il seguente:



La scala adottata per la visualizzazione è quella logaritmica. I punti blu corrispondono ai dati reali, fino al 30 Aprile; dal 1° Maggio, la linea verde e le linee rosse indicano rispettivamente la predizione effettuata dalla regressione lineare e l’errore massimo e minimo. Secondo quanto dichiarato dal grafico, avremmo dovuto avere zero terapie intensive occupate e zero deceduti intorno al 18 Maggio.

1. Exponential smoothing **[3]**,**[4]**

Poiché è chiaro che i dataset presi in considerazione contengono serie temporali, di seguito vengono analizzati dei metodi di predizione più adatti, a partire dalla Exponential smoothing; per cominciare a studiarla partiamo dalla Single Exponential Smoothing, la più semplice e basilare ma meno accurata nel risultato perché non prende in considerazione alcun trend o stagionalità negli input e la predizione è basata solamente solo sulla somma pesata delle passate osservazioni. È richiesto un solo parametro, α chiamato *smoothing factor*. Questo è un iperparametro e controlla quanto la storia pregressa influisce sulle predizioni: per valori vicino ad 1, il modello peserà di più le ultime osservazioni, mentre per valori bassi, vicini allo 0, sono i dati più vecchi che influiscono sul calcolo della media.

α è dunque un iperparametro; nel codice, *ExponentialSmoothing.py*, dopo aver estratto la colonna dei *nuovi\_positivi*, è utilizzata la funzione *SimpleExpSmoothing* della libreria *Statsmodels*.

date\_format = [pd.to\_datetime(d) for d in giorni]

variable = 'nuovi\_positivi'     # è una delle colonne del file csv

fig, ax = plt.subplots(figsize=(12, 5))     # con subplots creo tupla che contiene  > oggetto figura e assi così ci posso lavorare come entità separate

ax.grid()

ax.scatter(date\_format,data[variable])

ax.set(xlabel="Date",ylabel=variable,title=variable)

date\_form = DateFormatter("%d-%m")

ax.xaxis.set\_major\_formatter(date\_form)

ax.xaxis.set\_major\_locator(mdates.DayLocator(interval = 3))

plt.title('nuovi\_positivi')

#plt.show()

fit1= SimpleExpSmoothing(nuovi\_pos, initialization\_method="heuristic").fit(

smoothing\_level=0.2,optimized=False)

fcast1 = fit1.forecast(3).rename(r'$\alpha=0.2$')    #0.2 è valore basso -> faccio pesare di più storia pregressa; per capire significato paramentro alpha vedi: https://www.statsmodels.org/stable/examples/notebooks/generated/exponential\_smoothing.html

fit2 = SimpleExpSmoothing(nuovi\_pos, initialization\_method="heuristic").fit(

smoothing\_level=0.6,optimized=False)

fcast2 = fit2.forecast(3).rename(r'$\alpha=0.6$')

fit3 = SimpleExpSmoothing(nuovi\_pos, initialization\_method="estimated").fit()

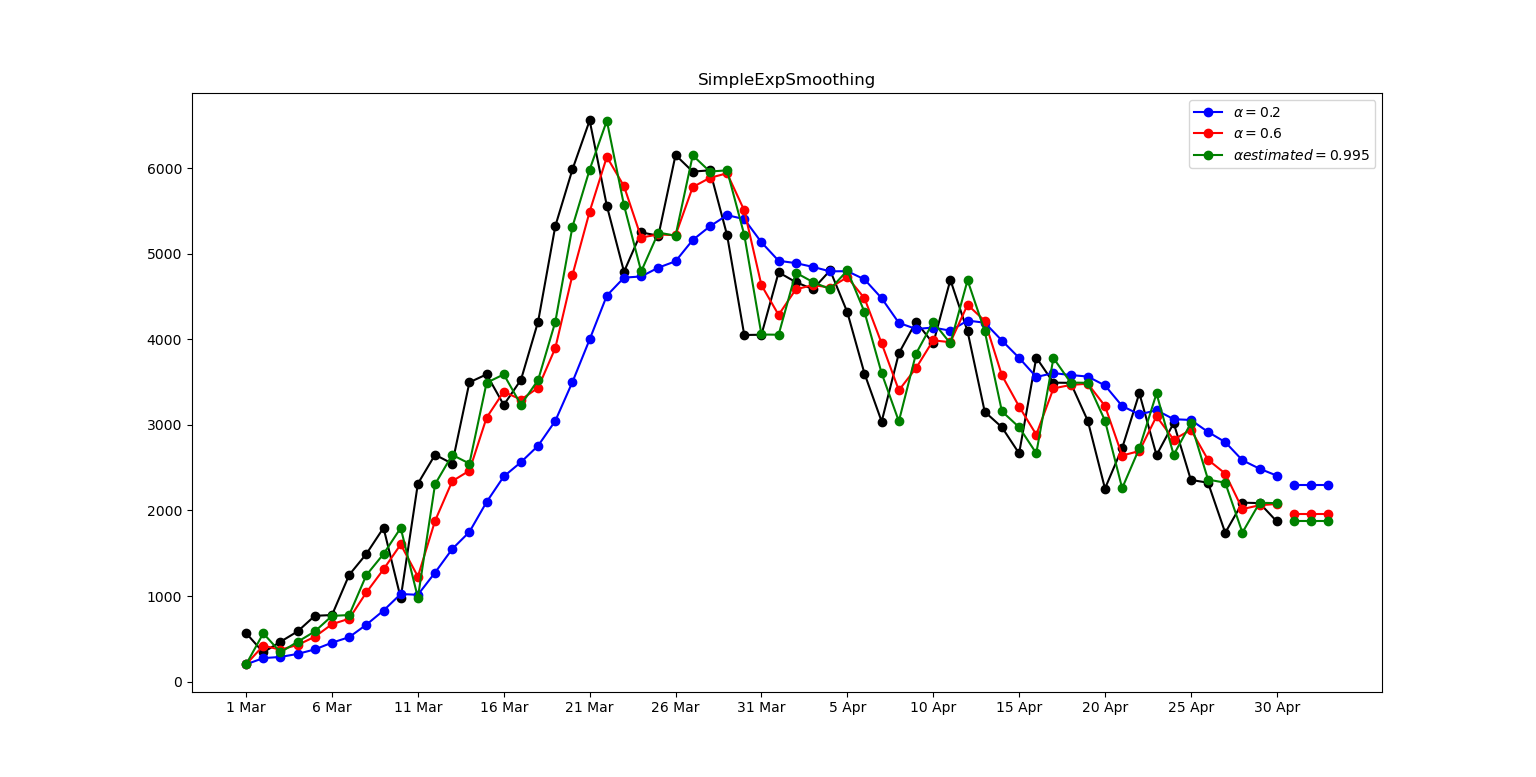
# qui alpha lo facciamo decidere a statsmodel: troviamo valore ottimizzato ed è approccio migliore

fcast3 = fit3.forecast(3).rename(r'$\alpha estimated=%s$'%fit3.model.params[

'smoothing\_level'])

*initialization\_method* indica come inizializzare la ricorsione e di default è ‘estimated’: questo metodo tratta i valori iniziali come parametri e la funzione li sceglie per minimizzare la somma dell’errore quadratico **[5]**. L’iperparametro α lo possiamo scegliere arbitrariamente, ma è anche possibile utilizzare la funzione *fit* che trova il valore di α che minimizza l’errore quadratico medio. Nel grafico, abbiamo scelto a piacere due valori arbitrari e abbiamo provato il valore alfa restituito dalla funzione *fit* per confrontare i risultati: sono stati scelti α = 0.2 e α = 0.6. Il terzo caso α è deciso direttamente da Statsmodels.

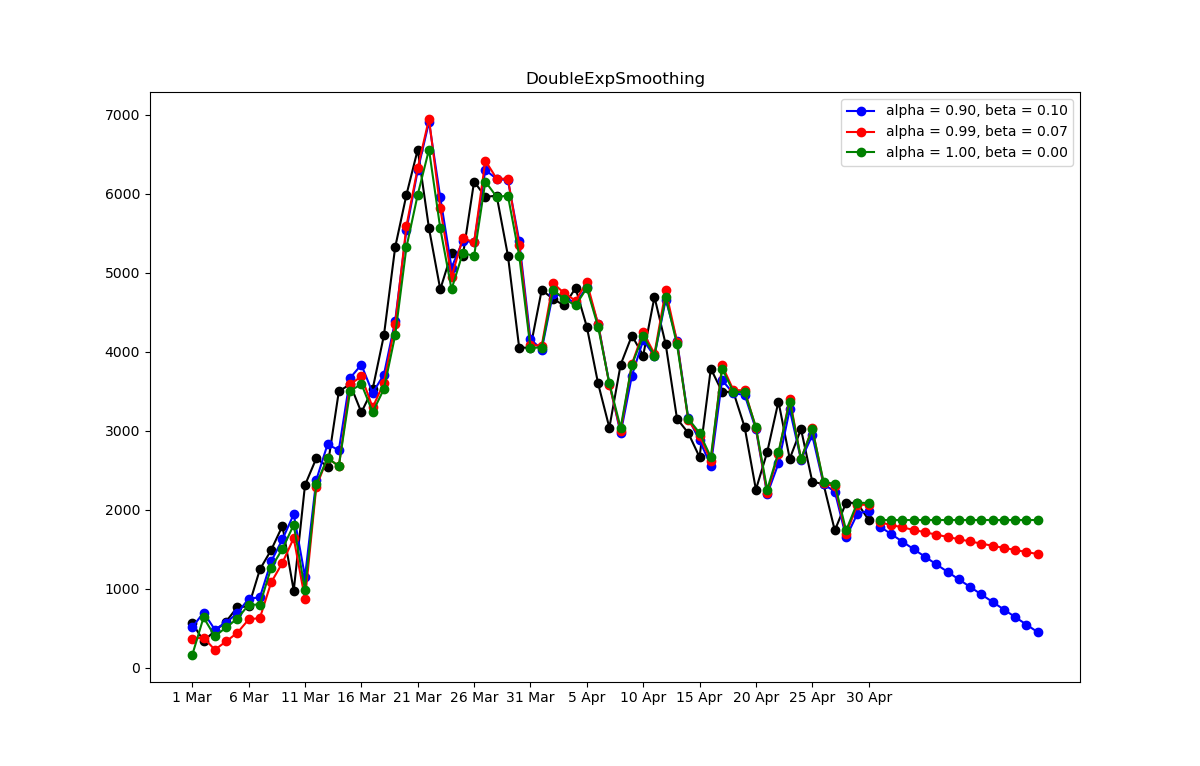
Di seguito, il grafico dei risultati:



Il *forecast* viene effettuato solamente per i successivi tre giorni ma è comunque possibile ricavare qualche informazione utile: quanto detto sull’iperparametro si è rivelato corretto, infatti per valori di α vicini ad 1, la linea verde e la linea rossa seguono bene la linea nera, corrispondente ai valori reali (per quella verde è possibile osservare che questa corrisponde semplicemente ad uno *shift* di un giorno rispetto alla linea nera). La linea blu invece, pesando di più la serie storica, sottostima fin dall’inizio il numero di nuovi positivi attestandosi a fine previsione sopra le altre due linee. Questo comportamento è dettato dalla rapida crescita di casi di metà marzo.

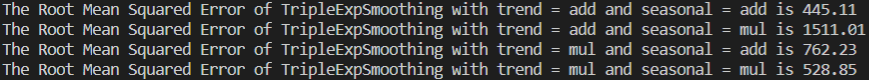
La *Double Exponential Smoothing* è un’estensione della *Simple* e grazie all’iperparametro β supporta i trend: β dunque controlla l’influenza nel cambio dell’andamento nella serie temporale. Il metodo inoltre consente di indicare se il trend è lineare o esponenziale: uno dei parametri della funzione *ExponentialSmoothing* è chiamato proprio *trend*, che può accettare come stringhe ‘add’ (per trend lineari) o ‘mul’ (per trend esponenziali). Nel codice sono stati provati entrambi; inoltre è stato sperimentato il parametro *damped\_trend= True*, per indicare qualora il trend sia da considerarsi smorzato.

I risultati ottenuti sono i seguenti:

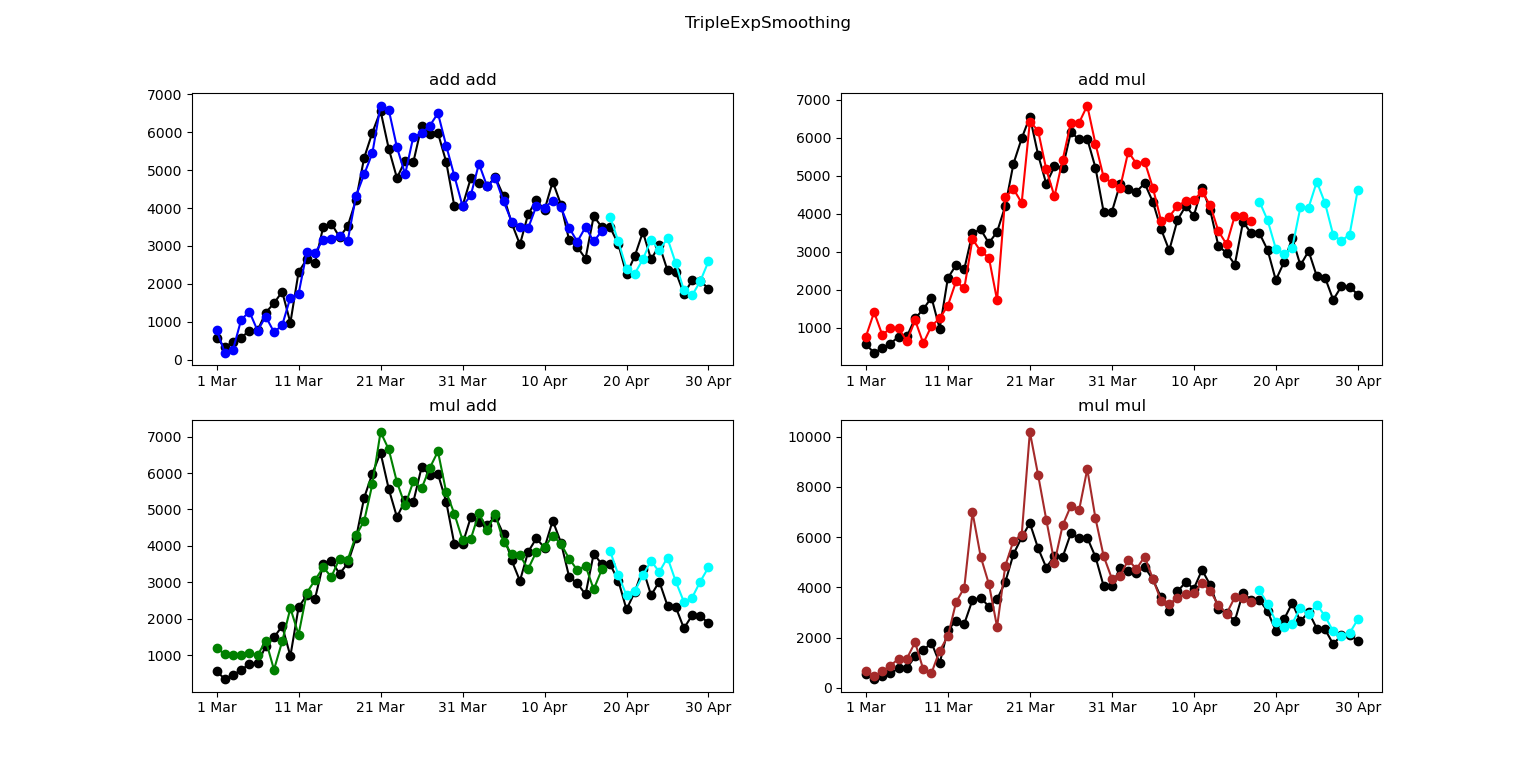


Questa volta α e β sono stati decisi in autonomia dalla libreria. La linea blu corrisponde alla predizione utilizzando il trend “add” ed è quella che segue di più la tendenza del grafico. La linea rossa è il trend “mul. Si può notare come sono stati predetti i successivi 15 giorni e che la linea verde, corrispondente a *damped\_trend= True*, non sia significativa per il nostro problema.

Infine, poiché è presente una stagionalità nel dataset come già visto nel primo paragrafo, la *TripleExponentialSmoothing* risulta essere la più adatta per il nostro studio in quanto la supporta. Essendo un’estensione della *Double*, presenta anch’esso un parametro, γ, che controlla l’andamento della stagionalità, lineare (‘add’) o esponenziale (‘mul’); se il parametro *seasonal* è esplicitato, è necessario indicare anche il numero di periodi all’interno di un ciclo stagionale (nel nostro caso impostato a 7 per indicare un ciclo settimanale). Per confrontare le prestazioni dei modelli con le varie combinazioni di *trend* e *seasonal* possibili, è calcolata la radice dell’errore quadratico medio (d’ora in avanti indicato con il corrispondente inglese *RMSE*): ancor prima di visualizzare i grafici, è possibile stabilire il modello migliore. Per la realizzazione di questi modelli sono state analizzate tutte le combinazioni *add* e *mul* per i parametri β e γ:

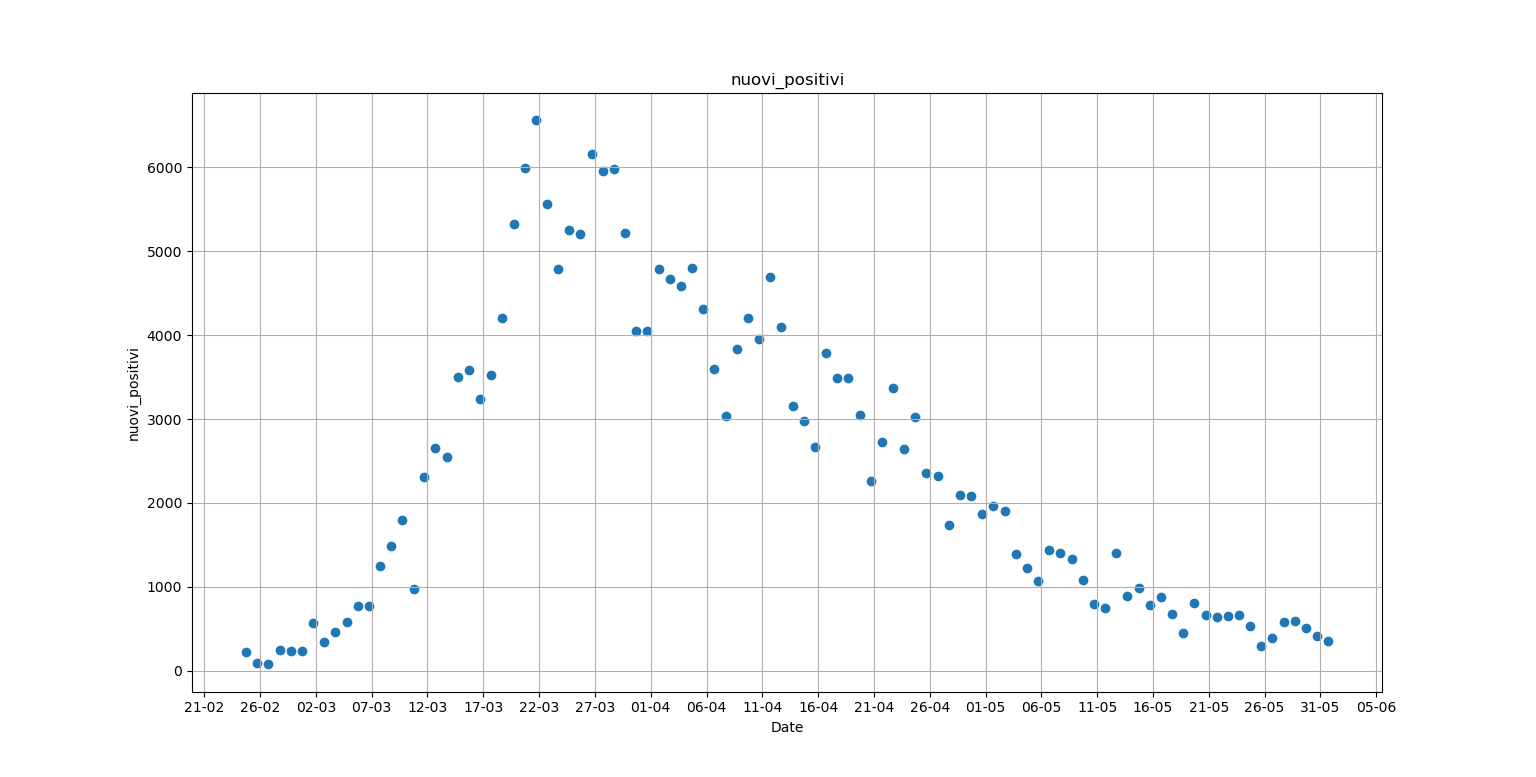


Sembra dunque che il modello che presenta *trend = add e seasonal = add* (ovvero andamenti lineari) sia il migliore. Verifichiamo ciò studiando il grafico sottostante:

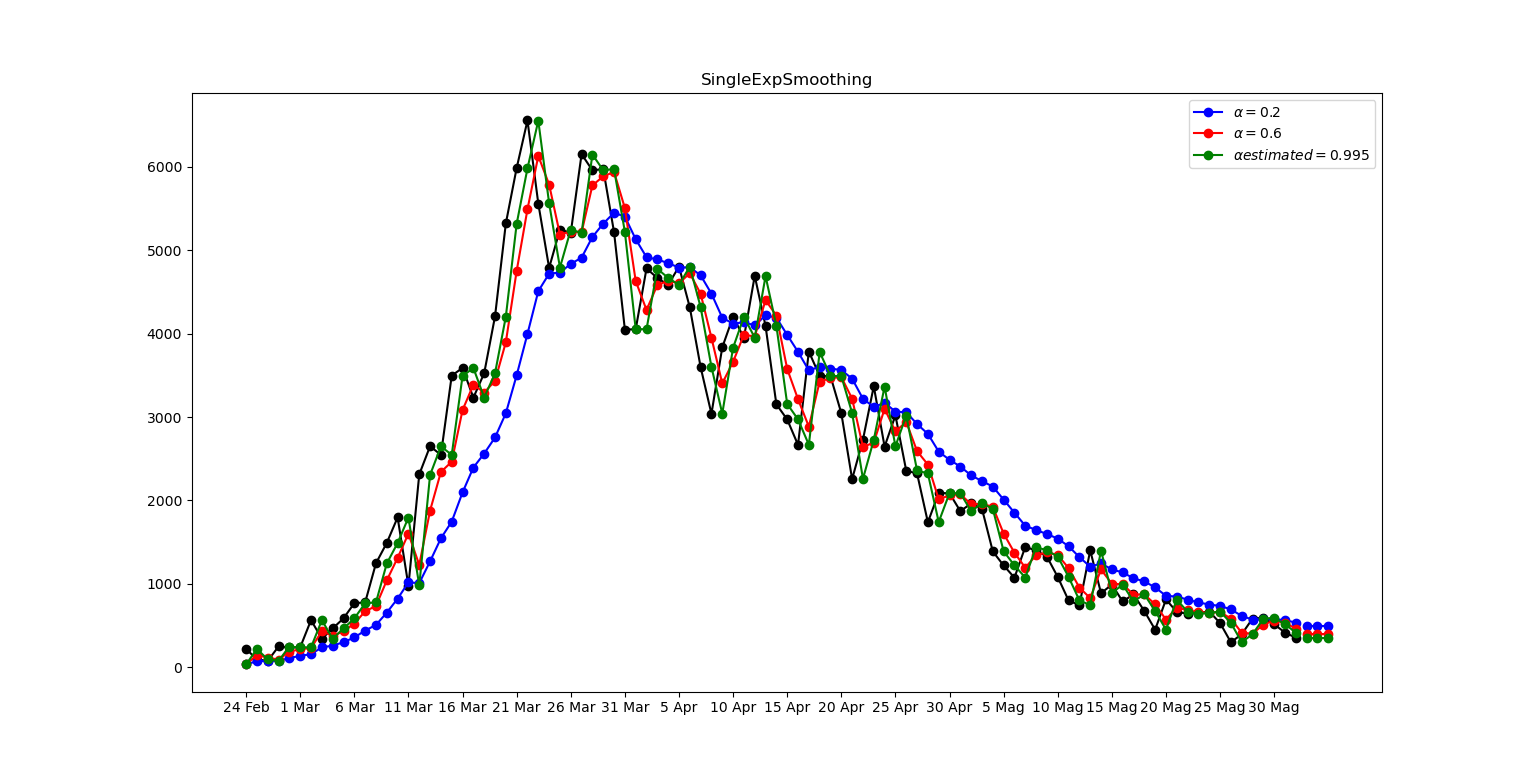


Come calcolato dal RMSE notiamo che la predizione nel primo grafico (linea azzurra) è quella che si discosta meno dai valori reali (linea nera). Si osservi inoltre come il secondo e il terzo grafico divergono nella predizione mentre l’ultimo grafico è accettabile.

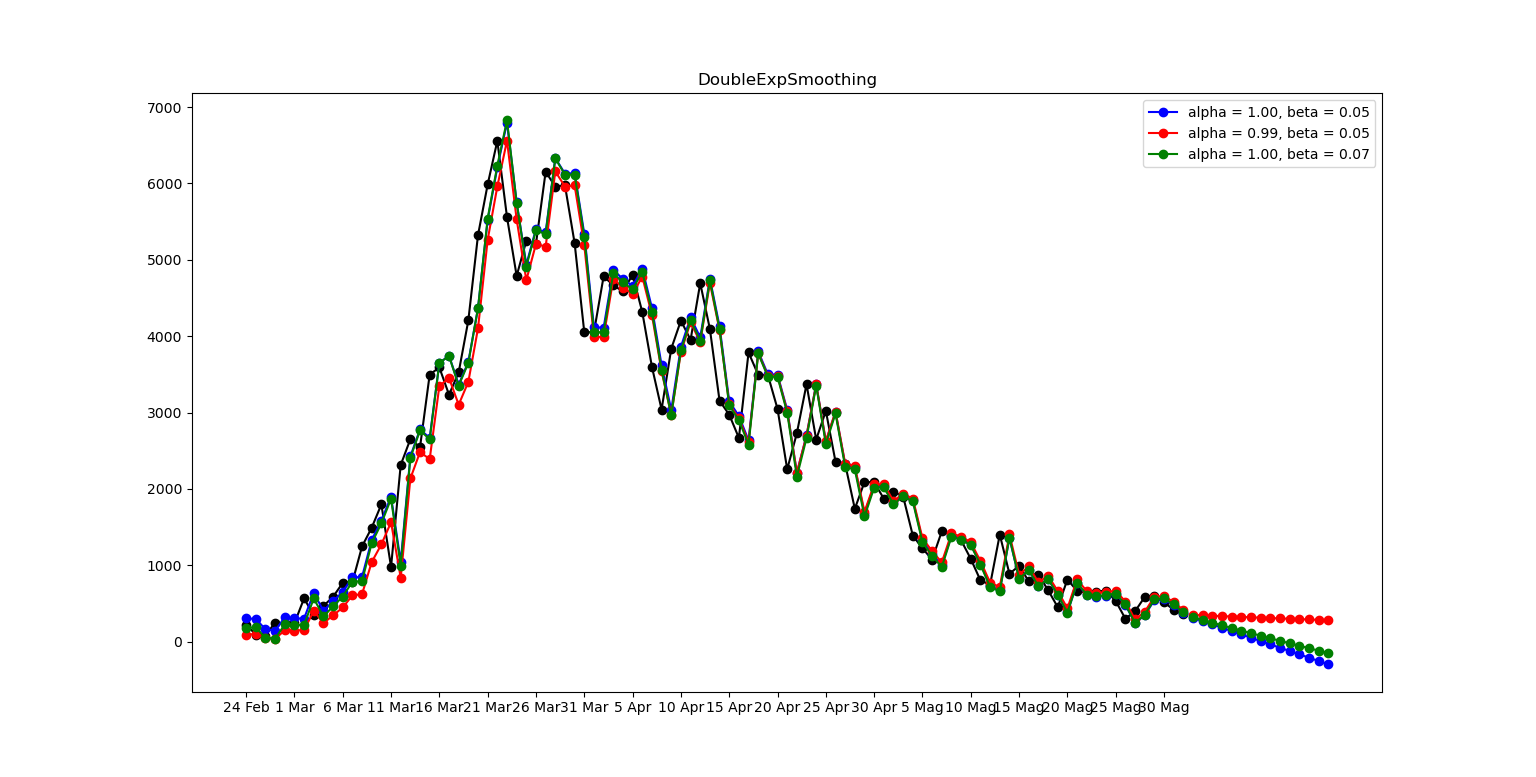
Dopo aver analizzato questi grafici, è interessante capire se i risultati precedentemente ottenuti siano diversi nel caso in cui il dataset sia più grande; pertanto in *big\_ExponentialSmoothing.py*, il file csv letto presenta anche il mese di maggio:



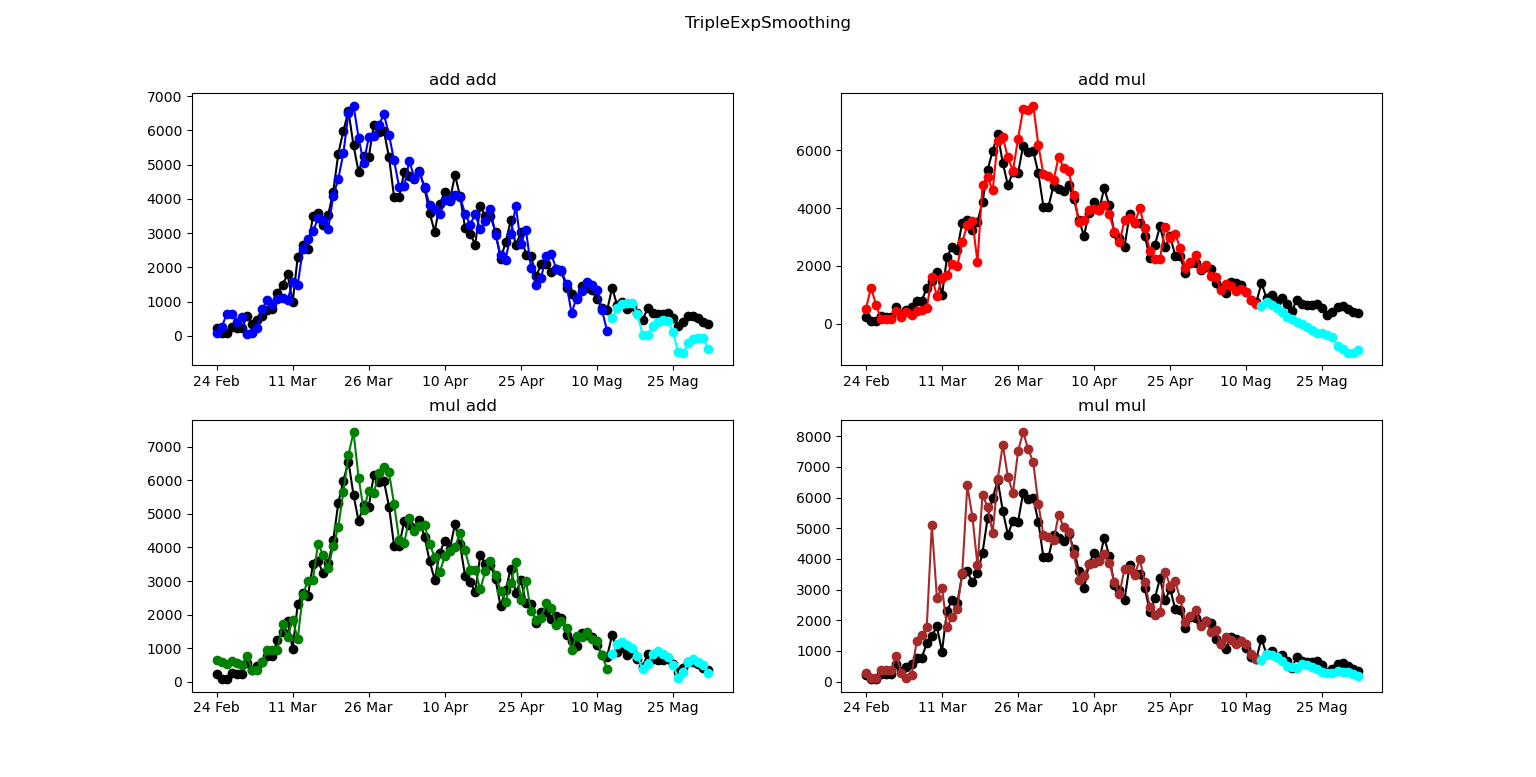
Come si vede, sono stati aggiunti nel grafico i valori dei *nuovi\_positivi* di fine febbraio e di maggio. Ora utilizziamo tutte le funzioni spiegate precedentemente. I grafici che ne risultano sono i seguenti:



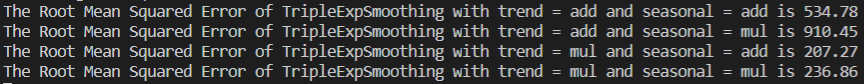
La Single Exponential Smoothing come abbiamo visto è poco utile alla previsione dei valori. Si può notare come nonostante il grafico sia molto più ampio, il valore stimato dalla funzione *fit* sia esattamente lo stesso.



Passando alla Double Exponential Smoothing, nulla di particolare interesse è rilevato. Si osservi però che il modello con *trend = mul* è quello più impreciso e ciò è coerente con quanto si può osservare nell’intero dataset (l’andamento non è certamente esponenziale, soprattutto verso la fine).



Infine sulla *TripleExpSmoothing*: se con un dataset ridotto il modello migliore era quello che presentava stagionalità e andamenti lineare, qui sembra che i migliori siano i modelli con *trend = mul, seasonal = add* e *trend = mul, seasonal =mul*; osserviamo i valori di RMSE per avere conferma:



Gli ultimi due valori di RMSE confermano le nostre previsioni iniziali.

Arriviamo dunque alla conclusione che aumentare la dimensione del dataset può aiutare a migliorare le prestazioni del modello di previsione.

1. SARIMA **[6]**

*Seasonal Autoregressive Integrated Moving Average* (SARIMA) è un’estensione di ARIMA che supporta dati di una serie storica (*time series data*) che presentano una componente di stagionalità. Il modello ha quattro iperparametri in aggiunta ai tre presenti in ARIMA; dunque con SARIMA è necessario scegliere 7 iperparametri:

* Non stagionali: **p, d, q**
* Stagionali: **P, D, Q, m**

Dove:

* **p**: numero di lag della componente Autoregressiva
* **d**: numero di integrazioni
* **q**: numero di lag della componente Media mobile
* **P**: numero di stagioni della componente Autoregressiva
* **D**: numero di integrazioni nelle stagioni
* **Q**: numero di stagioni della componente Media Mobile
* **m**: numero di passi temporali per un singolo periodo stagionale

Solitamente la scelta di questi avviene mediante metodi come la *grid search* in cui si provano diverse combinazioni di iperparametri e si scelgono quelli che hanno una miglior performance.

Nel codice, la *grid search* è così implementata:

def sarima\_grid\_search(y,seasonal\_period):

    p = d = q = range(0, 2)

    pdq = list(itertools.product(p, d, q))

    seasonal\_pdq = [(x[0], x[1], x[2],seasonal\_period) for x in list(

itertools.product(p, d, q))]

    mini = float('+inf')

    for param in pdq:

        for param\_seasonal in seasonal\_pdq:

            try:

                mod = sarima.SARIMAX(y, order=param, seasonal\_order=param\_seasonal,  enforce\_stationarity=False, enforce\_invertibility=False)

                results = mod.fit()

                if results.aic < mini:

                    mini = results.aic

                    param\_mini = param

                    param\_seasonal\_mini = param\_seasonal

            except:

                continue

    print('The set of parameters with the minimum AIC is: SARIMA{}x{} - AIC:{}'

.format(param\_mini, param\_seasonal\_mini, mini))

La bontà della *grid search* è espressa dal valore di AIC (*Akaike Information Criterion*): questo fornisce una misura della qualità della stima di un modello statistico tenendo conto sia della bontà di adattamento che della complessità del modello.

Dato un dataset è sufficiente chiamare *sarima\_grid\_search* una volta sola e salvare gli iperparametri trovati; come nei casi precedenti, la predizione è fatta sulla colonna del file csv *nuovi\_positivi*. È sufficiente a questo punto definire un *training set* e usarlo all’interno della *grid search*, definendo anche il periodo della stagionalità (per noi, 7 giorni).

A scopo educativo abbiamo provato la funzione sopraesposta con due *training set* diversi, uno più grande dell’altro (80% dell’intero dataset contro 70%): i sette iperparametri si sono rivelati uguali per entrambi i modelli ma con valori di AIC diversi (482 per il *training set* più grande contro 390 del più piccolo).

Trovati dunque gli iperparametri, questi vengono inseriti all’interno della funzione *sarima\_eva*:

# Call this function after pick the right(p,d,q) for SARIMA based on AIC

#

def sarima\_eva(y,order,seasonal\_order,seasonal\_period,pred\_date,y\_to\_test,y\_train):

    # fit the model

    mod = sarima.SARIMAX(y,

                                order=order,

                                seasonal\_order=seasonal\_order,

                                enforce\_stationarity=False,

                                enforce\_invertibility=False)

    results = mod.fit()

    print(results.summary().tables[1])

    # The dynamic=False argument ensures that we produce one-step ahead forecasts,

    # meaning that forecasts at each point are generated using the full history up  to that point.

    pred = results.get\_prediction(start=pred\_date, dynamic=False)

    pred\_ci = pred.conf\_int()

    y\_forecasted = pred.predicted\_mean

    mse = ((y\_forecasted - y\_to\_test) \*\* 2).mean()

    print('The Root Mean Squared Error of SARIMA with season\_length={} and dynamic = False {}'.format(seasonal\_period,round(np.sqrt(mse), 2)))

    # (1) If we are forecasting x(T+1), x(T+2)... x(T+h), using information up to

period T,

    # we may have to do it recursively by using forecast of x(T+1) in order to

forecast x(T+2) and so on.

    # (2) If we are forecasting x(T+1) using information up to time T, then

forecasting x(T+2) using information up to period T+1 and so on,

    # we may have to wait each step until we have updated information (or pretend

we are waiting).

    # Call (1) "Forecast with a FIXED INFORMATION SET" by recursively using

predicted values if needed.

    # This is what EViews calls "dynamic forecasting".

    # Call (2) "Forecast with a MOVING ESTIMATION SAMPLE" one-step- ahead only (or, to be more general, with a fixed h-step-ahead).

    # This is what EViews calls "static forecasting"

    pred\_dynamic = results.get\_prediction(start=pred\_date, dynamic=True,

full\_results=True)

    pred\_dynamic\_ci = pred\_dynamic.conf\_int()

    y\_forecasted\_dynamic = pred\_dynamic.predicted\_mean

    mse\_dynamic = ((y\_forecasted\_dynamic - y\_to\_test) \*\* 2).mean()

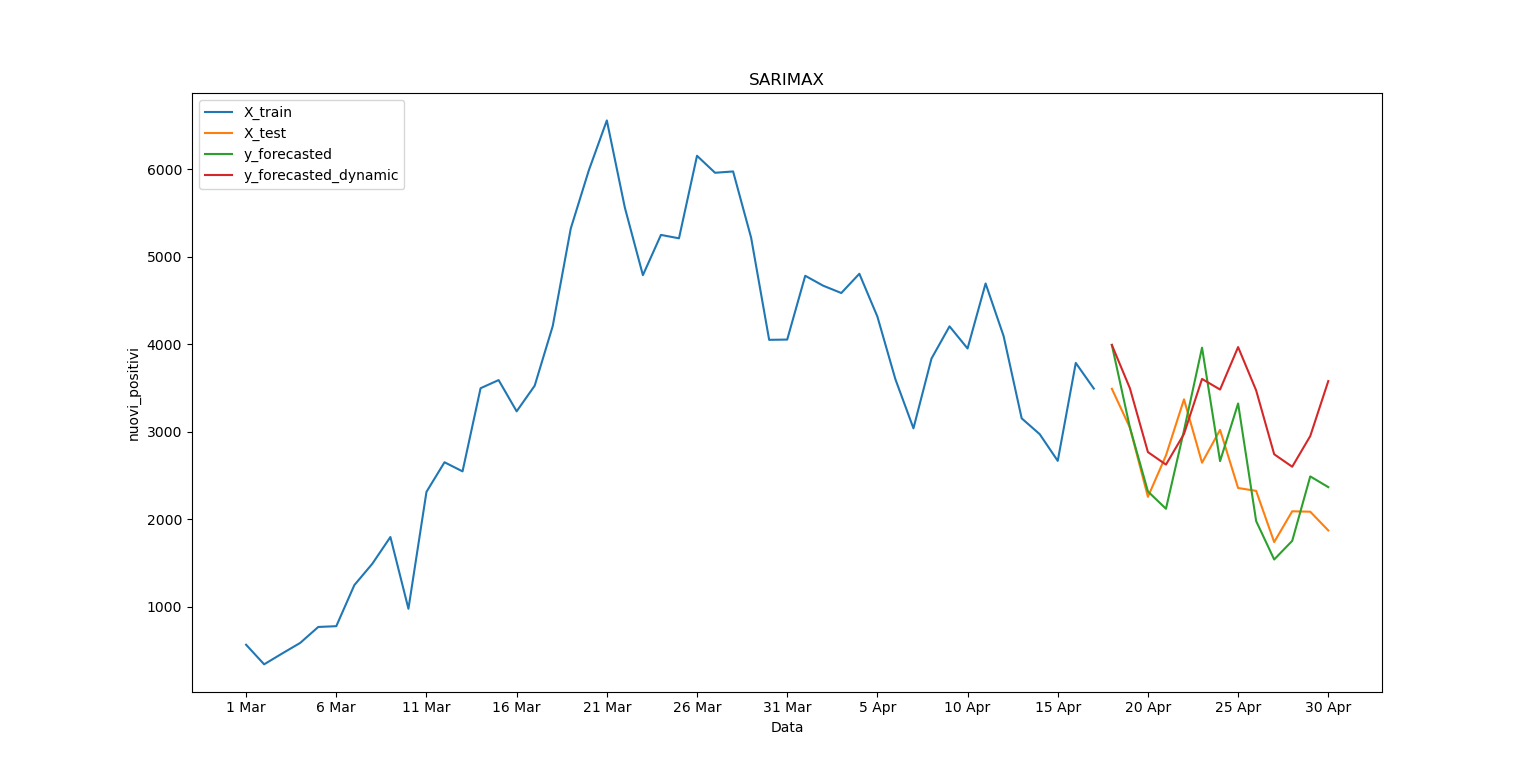
    print('The Root Mean Squared Error of SARIMA with season\_length={} and dynamic = True {}'.format(seasonal\_period,round(np.sqrt(mse\_dynamic), 2)))

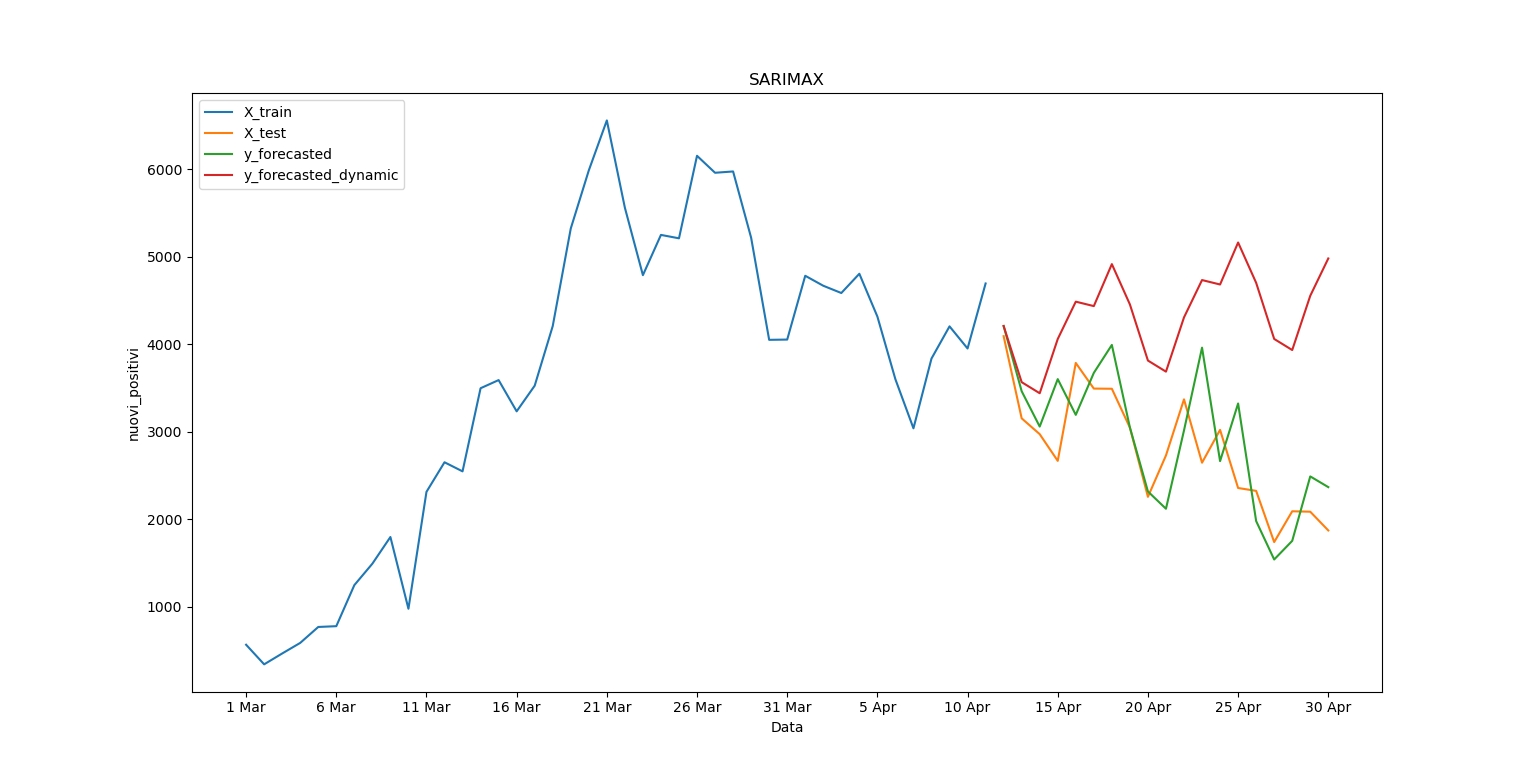
    return (results)

La predizione viene eseguita in due modi diversi: in un caso l’argomento *dynamic* è impostato su *false*, nell’altro su *true*; l’argomento *dynamic* nel caso *false* utilizza una *one-step-ahead prediction*, ossia la predizione avverrà di un singolo valore alla volta e poi si andrà a vedere il valore reale nel *test set*, in questo modo la previsione, tenendo conto dei valori reali, avrà un errore minore e si adatterà nel tempo ai cambiamenti della curva.

Nel caso *Dynamic=true* invece verrà sempre utilizzata la *one-step-ahead prediction* per un numero di punti, senza andare a controllare il valore effettivo reale del dataset: in questo modo è possibile effettuare una stima, senza avere necessariamente i dati effettivi, a discapito di una perdita di accuratezza nella predizione **[7]**. Un significato più approfondito di questo parametro è reperibile nei commenti sul codice. Si noti inoltre come all’interno della funzione venga valutato l’RMSE.

I risultati delle predizioni sono affidati ai grafici sottostanti, nei quali sono presenti *training set*, *testing set* e le due predizioni con i differenti valori di *dynamic*:





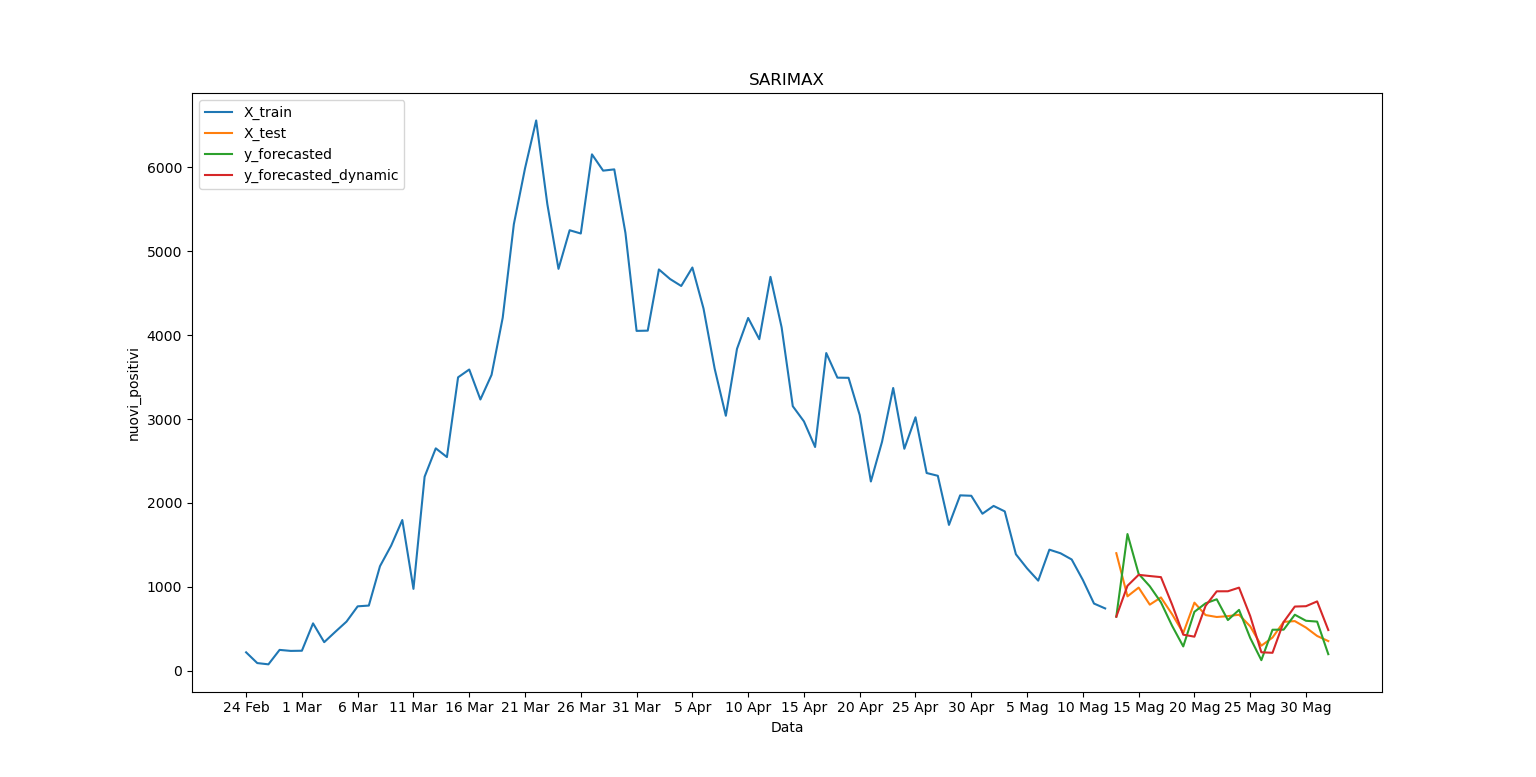
Per entrambi i grafici, la predizione con *dynamic = true* si rivela la più imprecisa; nel caso con *training set* più grande l’RMSE è il seguente:

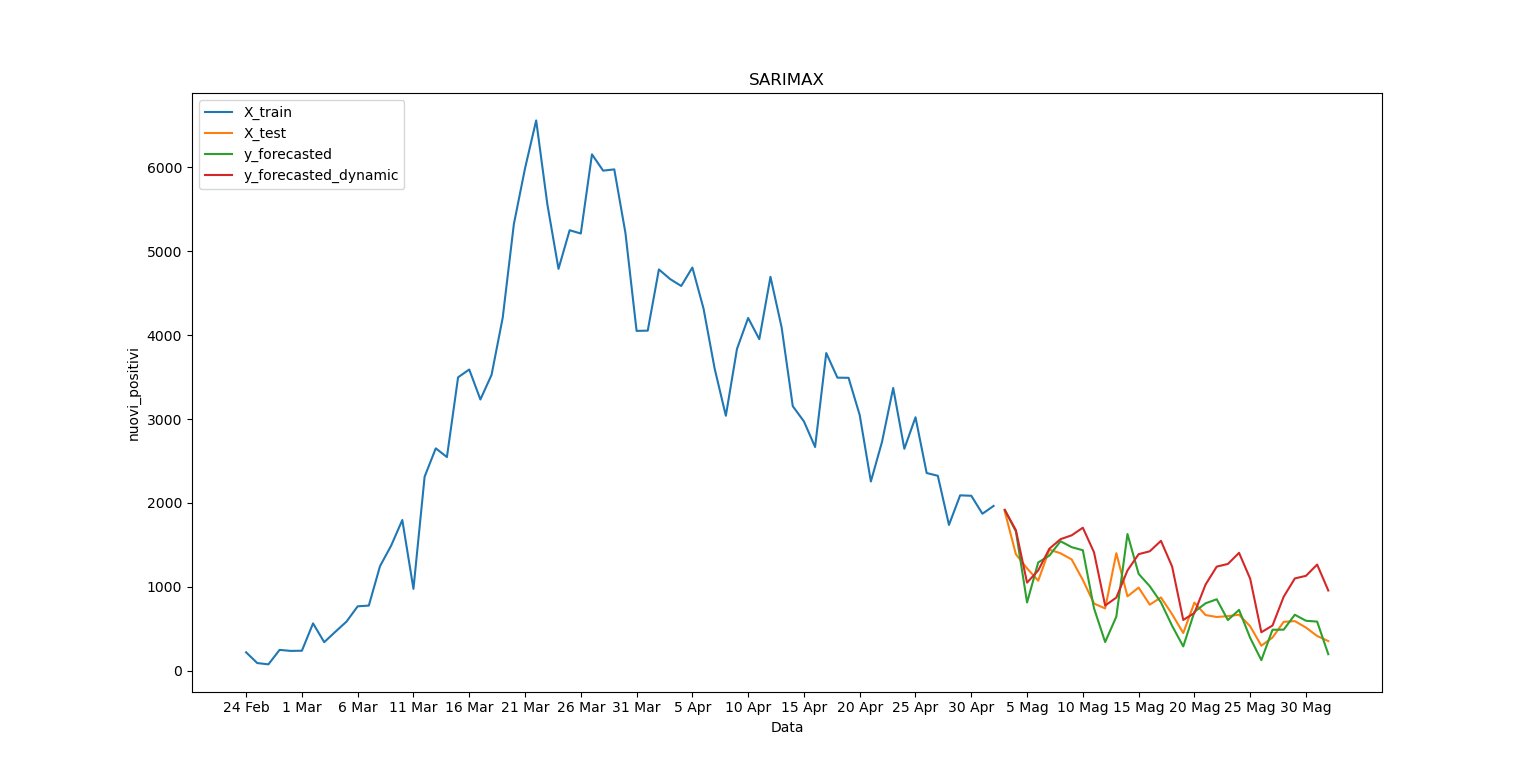


Invece per il *training set* corrispondete al 70% dell’intero dataset:



Come già accaduto nel paragrafo precedente, applichiamo quanto abbiamo visto sulla SARIMA con un dataset più grande e confrontiamo i risultati:









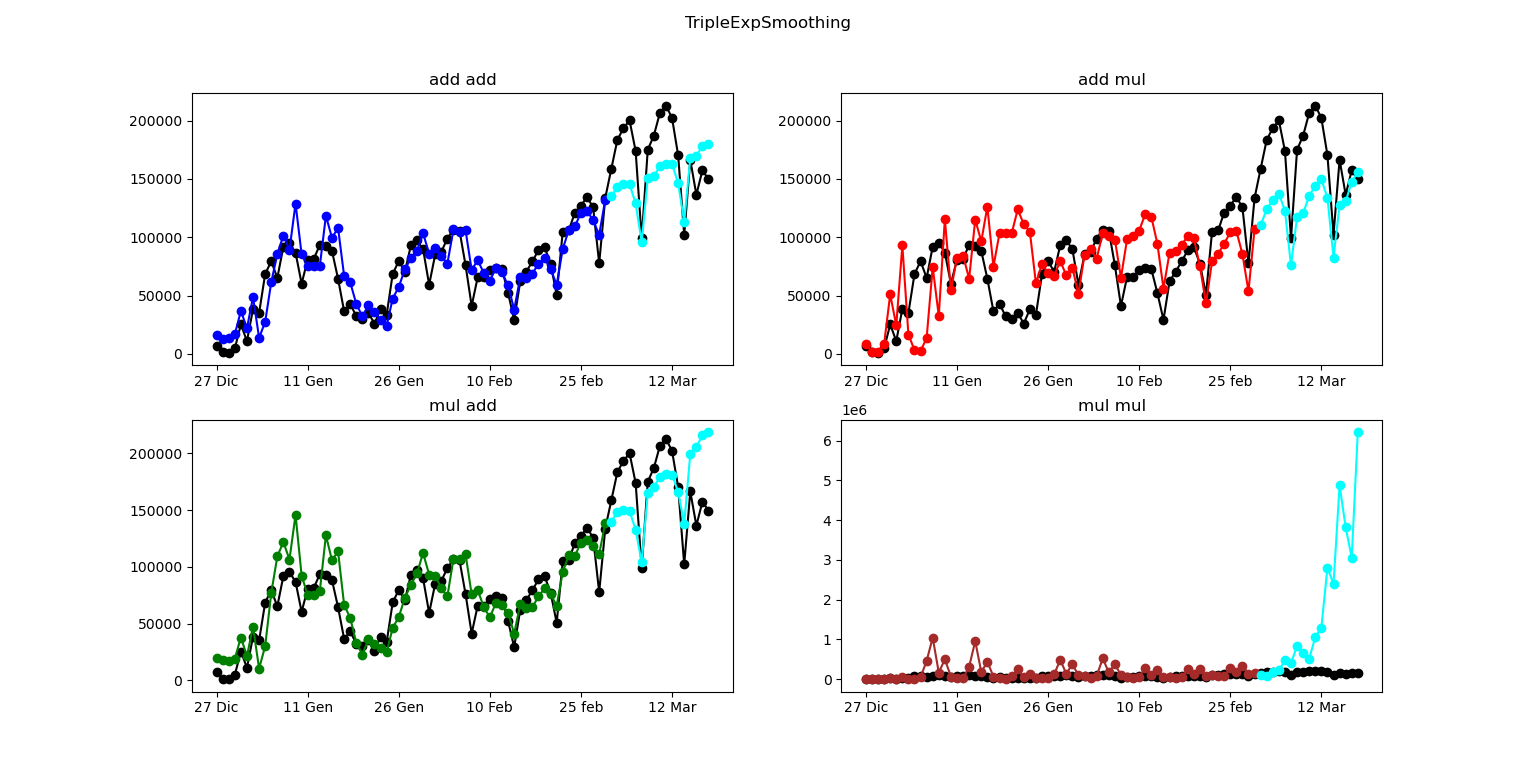
Notiamo un importante risultato: la radice dell’errore quadratico medio è di gran lunga ridotto se si espande il dataset; inoltre nel caso di *training set* corrispondente all’80% di tutti i dati, la differenza tra i due metodi di predizione è trascurabile.

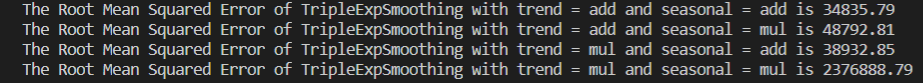
Dunque, come già avvenuto nella *Exponential Smoothing*, aumentare la dimensione del dataset aiuta il modello di predizione.

1. Vaccini

Per confermare quanto appreso negli scorsi paragrafi, abbiamo eseguito un’ulteriore predizione: abbiamo applicato i precedenti algoritmi di predizione su un nuovo dataset utilizzando un repository Github **[2]** al cui interno sono presenti i dati sulle vaccinazioni in Italia. Dunque, il codice presente all’interno di *vaccini.py* risulta pressoché identico a quanto già visto per i singoli modelli, con una differenza: poiché il dataset non presenta esplicitamente il numero totale di somministrazioni di vaccini giornalieri, ma un elenco regione per regione giornaliero, si è reso necessario l’utilizzo della funzione di Pandas *date\_range*, e delle operazioni di somma dei dati, per ricavare un array ordinato con le somministrazioni giornaliere in Italia; i dati contenuti in esso vanno dal 27 dicembre 2020, giorno in cui sono partite le vaccinazioni, fino al 18 marzo.

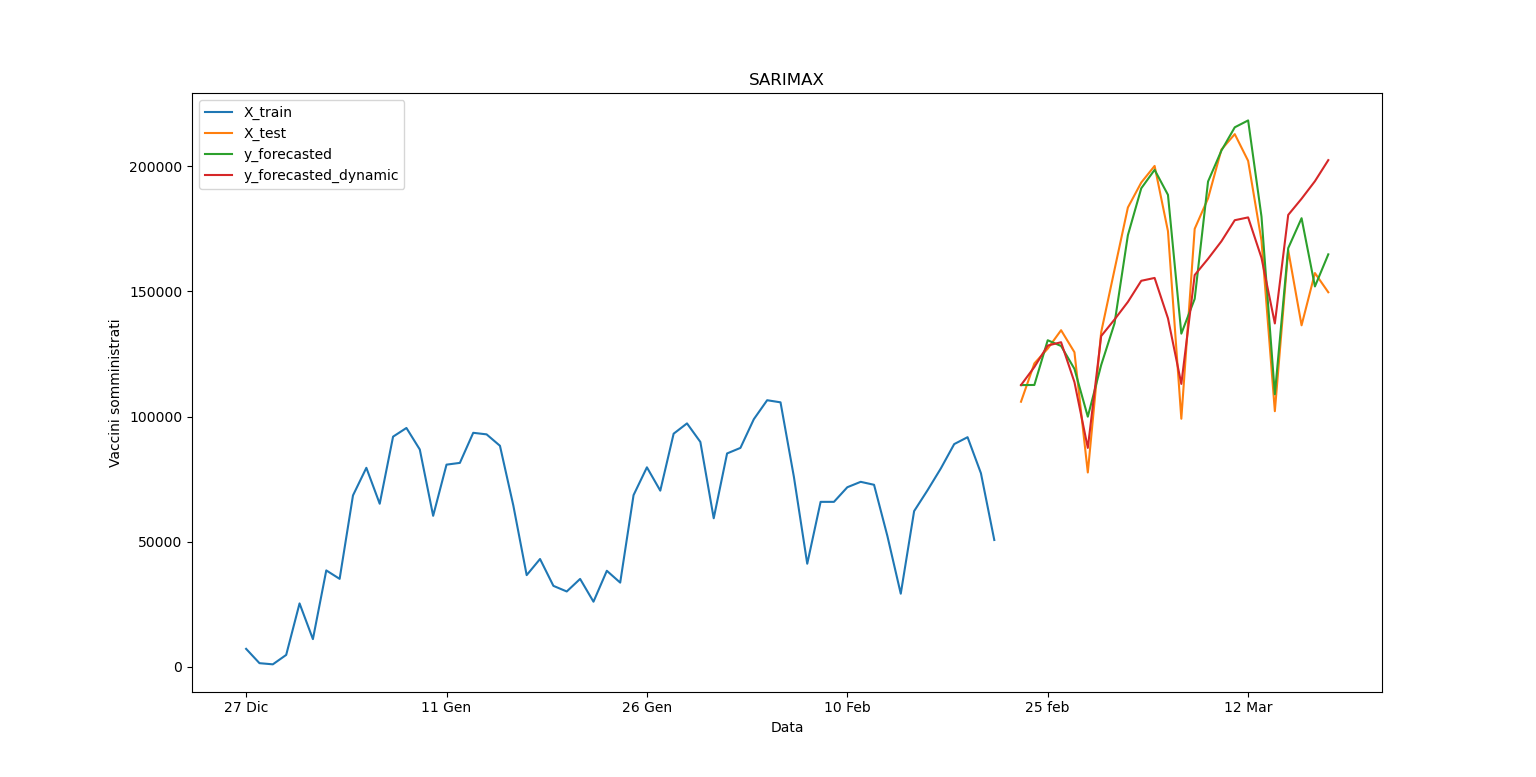
Di seguito, i risultati ottenuti con l’algoritmo Triple Exponential Smoothing:



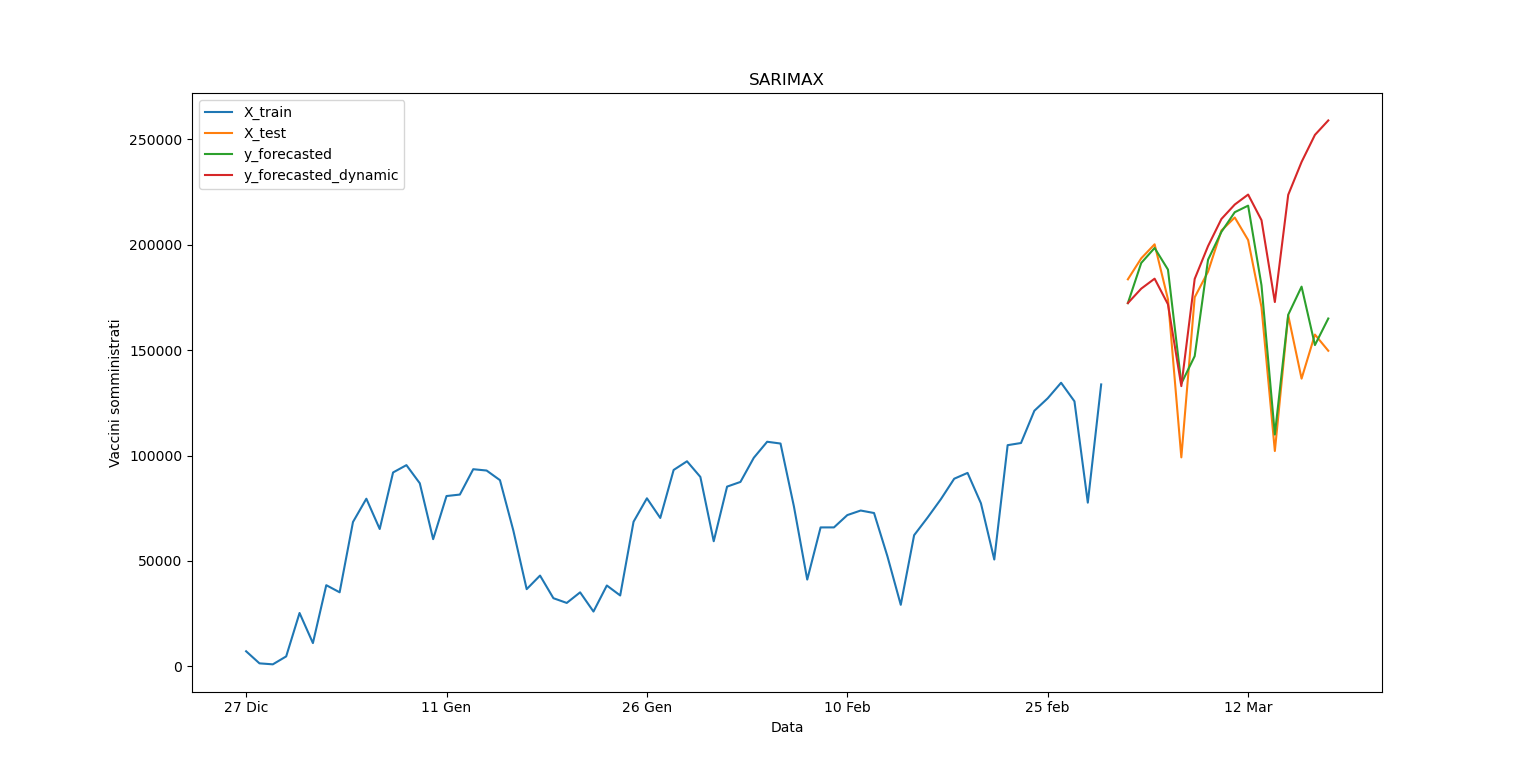


La radice dell’errore quadratico medio è di gran lunga maggiore a quanto visto con i contagi giornalieri: questo è dettato dal fatto che le vaccinazioni sono aumentate di molto nell’instate in cui il dataset è stato diviso in *training set* e in *testing set*. Per ottenere predizioni più precise, è sufficiente cambiare come viene diviso il dataset ma è interessante notare i risultati con questa suddivisione. Dai quattro grafici si nota come i risultati dei primi tre siano molto simili mentre il quarto grafico è completamente errato, da come si può vedere sia graficamente che dal RSME.

Qui sotto i risultati con l’algoritmo SARIMAX scegliendo rispettivamente nel primo grafico il 70% dei dati come training set mentre nel secondo l’80%.









Allo stesso modo, si confronti il *testing set* con le predizioni tramite SARIMA: l’RMSE è migliore nel caso di *training set* più piccolo poiché la suddivisione questa volta non avviene nel momento in cui la campagna vaccinale è andata a regime; come già avvenuto con i contagi, il modello con *dynamic=false* si rivela comunque migliore.

È importante notare in questi grafici come i valori reali del numero di vaccini somministrati sono comunque viziati da un fattore esterno non prevedibile: il 15 marzo l’Agenzia Italiana del Farmaco ha bloccato le somministrazioni del vaccino AstraZeneca e di conseguenza le vaccinazioni a partire dal giorno successivo si sono rivelate inferiori rispetto alle settimane precedenti. Questo stop è ben visibile dal picco presente poco dopo il *tick* del 12 marzo. Da questo si può dedurre che, nonostante tutti gli accorgimenti fatti nella scelta dei parametri, le previsioni non avranno mai un’accuratezza tale da garantire una sicurezza al 100%, poiché i soggetti su cui sono basate le previsioni possono essere influenzati positivamente o negativamente da fattori esterni assolutamente imprevedibili.

1. Conclusione e commenti

Gli obiettivi prefissati all’inizio di questo progetto possono considerarsi raggiunti: tramite le conoscenze acquisite durante il corso, abbiamo constatato l’efficacia e le potenzialità delle funzioni, applicando quanto imparato su dati di grande interesse ed attualità. Ciò nonostante, come sottolineato anche durante la stesura di questo elaborato, è necessario prestare attenzione: la raccolta dei dati deve essere accurata così come lo studio su di essi; inoltre è richiesta un’attenta lettura delle librerie che si vogliono utilizzare, poiché non è escluso che quanto si intenda implementare, non risulti di scarsa efficacia o addirittura non idoneo.

* Fonti

[1]

<https://github.com/pcm-dpc/COVID-19>

[2]

https://github.com/italia/covid19-opendata-vaccini

[3]

<https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/>

[4] <https://www.statsmodels.org/stable/examples/notebooks/generated/exponential_smoothing.html>

[5]

<https://stats.stackexchange.com/questions/509772/what-do-the-values-for-initialization-method-mean-in-statsmodels-simple-exponent>

[6]

<https://www.bounteous.com/insights/2020/09/15/forecasting-time-series-model-using-python-part-two/>

[7] <https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html#ARIMA-Postestimation:-Example-1---Dynamic-Forecasting>