The comparative analysis of SARIMA, Facebook Prophet, and LSTM for road traffic injury prediction in Northeast China.

Name, Surname : Yassine Ammour

Submission Date: 09-01-2023

Brief Introduction to the topic :

- This cross-sectional research aims to develop reliable predictive short-term prediction models to predict the number of Road trafic injuries in Northeast China.
- The study uses 3 different model and tries to conclude the best model for future practices
- The models are: Seasonal auto-regressive integrated moving average (SARIMA), Long Short-Term Memory (LSTM), and Facebook Prophet
- The three models were trained using data from 2015 to 2019.

Brief description of the steps :

1. Date from 2015 to 2020
2. Split the data into Training data and Test data.
3. Creat in sample predictions for Test set data Using the three different models.
4. We measure the predictive performance of the models using The (RMSE), (MAE) and, (MAPE) along side with the Plots.

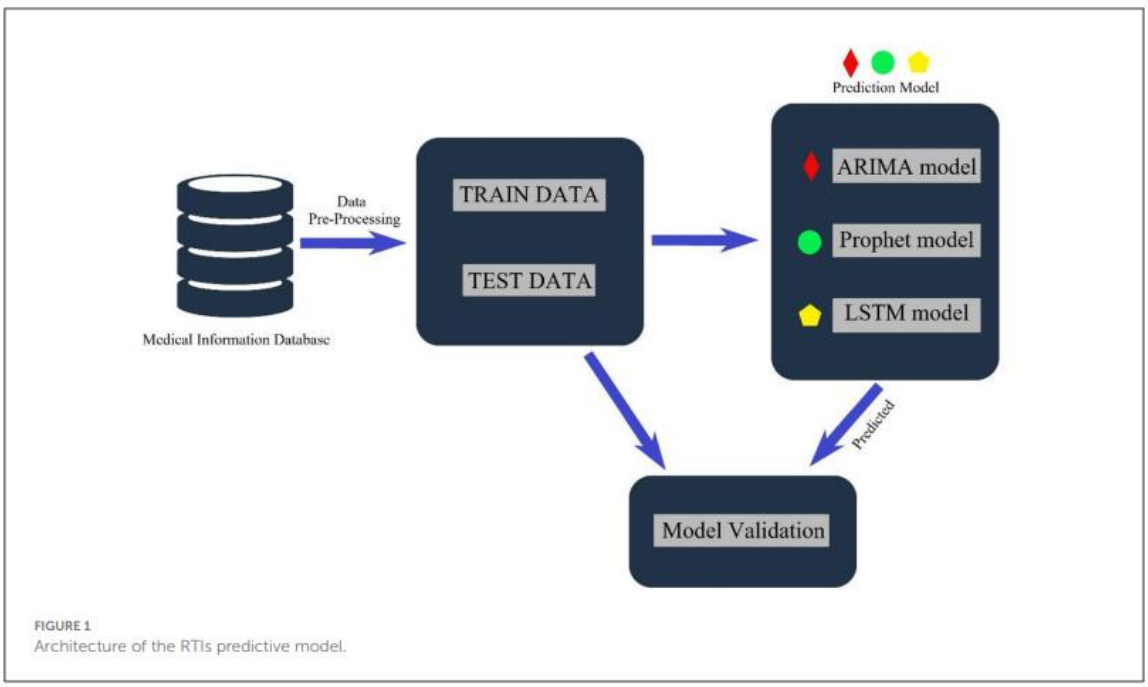


FIGURE 1
Architecture of the RTIs predictive model.

Methodology of the original article :

ARIMA Model :

- The Dickey-Fuller test was used to demonstrate that the data were non-stationary p <=0.05
- Using first-order differencing and seasonal differencing, we determines the parameters of p, d, and q from ACF and PACF.
- The parameters of the final model are then determined by means of a minimum AIC.
- The Ljung-Box test was used to test whether the residuals of the model are normaly distributed.
- Fit the model
- Plot the results

Facebook Prophet model :

- The Prophet model incorporates the effect of holidays, and there is no need to make the data stationary.

Prophet	Growth	Logistic
	Changepoint Range	0.8
	Holidays	CN
	Changepoint Prior Scale	0.05

LSTM model :

- The model used here is an LSTM improved RNN neural network
- First Normalize the data before feeding it into the LSTM model.
- Build the model
- Fit the model and plot the data

Method	Parameters	Values
LSTM	Layers	3
	No. of neurons	{16,32,64}
	Learning rate	0.01
	Dropout	0.3
	Optimizer	Adam
	Batch size	3
	Maximum Epochs	1,000
	Activation Function	Linear

Summary of the results of the original article

- The LSTM model had the highest prediction accuracy, followed by the Prophet model, and the SARIMA model had the lowest prediction accuracy.

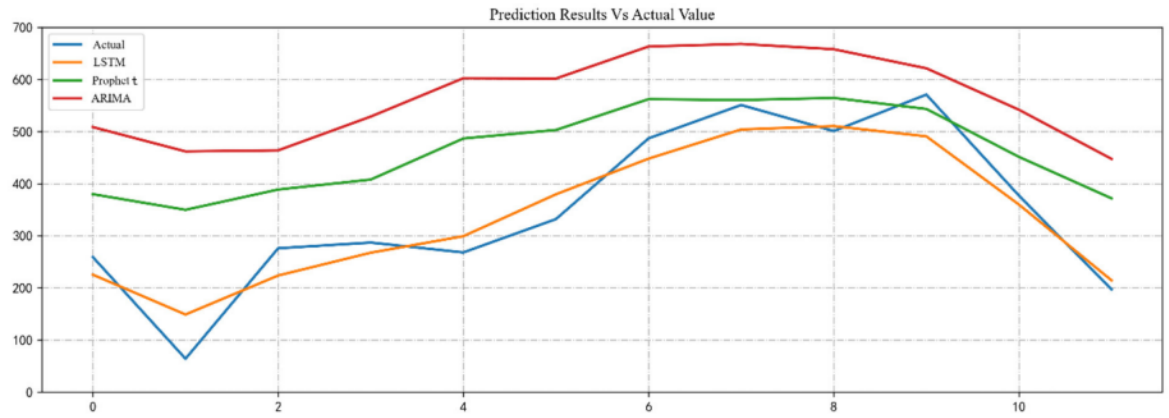


FIGURE 7
Visualizations of predicted and actual values for SARIMA, Prophet, and LSTM models.

	LSTM	Prophet	SARIMA $(1,1,0),(2,1,3)_{12}$
RMSE	46.12	143.58	208.95
MAE	39.93	121.2717	191.48
MAPE	20.26%	71.04%	92.82%

TABLE 4 Actual values vs. predicted values from the three models.

Mouth	Actual	LSTM	Prophet	SARIMA
January	259	225.29	379.84	408.59
February	64	148.88	349.81	361.90
March	276	223.70	388.75	363.91
April	287	267.50	407.86	528.79
May	268	299.20	486.73	602.42
June	332	379.61	503.07	601.83
July	487	448.03	562.20	663.29
August	551	504.08	560.51	668.26
September	501	510.49	564.63	658.03
October	571	490.83	543.43	621.43
November	377	359.91	451.45	541.87
December	197	214.41	371.83	447.52

Description of my emperical research :

- **Dataset**

```
df1
```

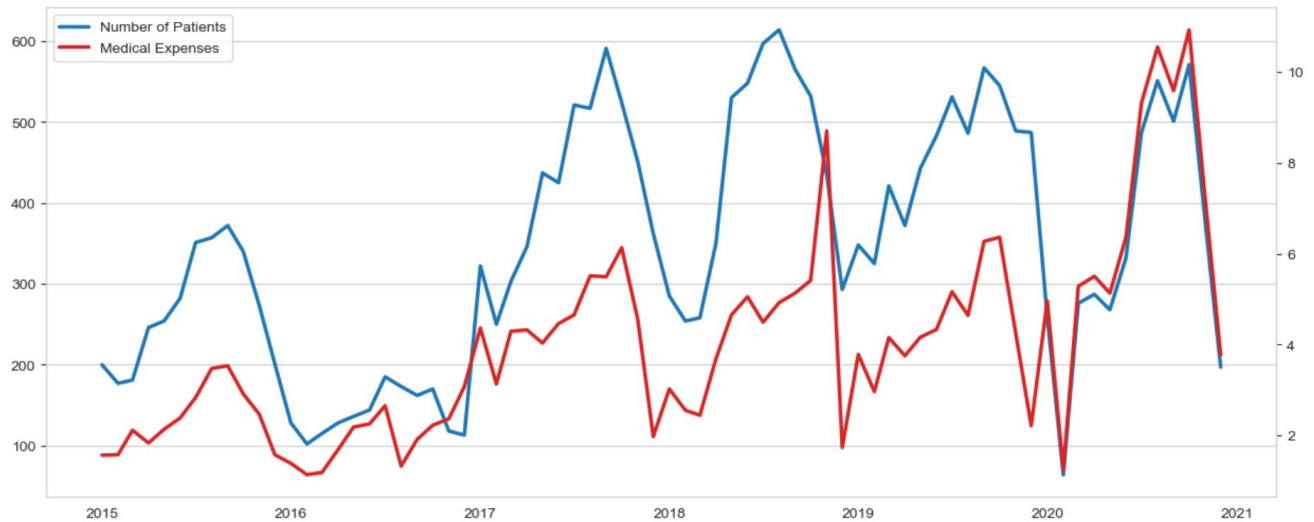
	Unnamed: 0	2015	2016	2017	2018	2019	2020
0	Jan.	200	128	322	285	348	259
1	Feb	177	102	250	254	325	64
2	Mar	181	115	303	258	421	276
3	Apr	246	128	346	350	372	287
4	May	254	136	437	530	443	268
5	Jun	282	144	425	548	483	332
6	Jul	351	185	521	597	531	487
7	Aug	357	173	517	614	486	551
8	Sep	372	162	591	565	567	501
9	Oct	340	170	524	532	545	571
10	Nov	273	118	451	435	489	377
11	Dec	201	113	362	293	487	197

```
s1
```

	value	value2
2015-01-01	200	1.56
2015-02-01	177	1.57
2015-03-01	181	2.11
2015-04-01	246	1.83
2015-05-01	254	2.13
...	...	...
2020-08-01	551	10.55
2020-09-01	501	9.59
2020-10-01	571	10.93
2020-11-01	377	7.22
2020-12-01	197	3.77

72 rows × 2 columns

```
fig, ax1 = plt.subplots(figsize=(15, 6))
sns.set_style("whitegrid")
ax2 = ax1.twinx()
l1, = ax1.plot(s1.value, label = 'label1', color='tab:blue', linewidth=2.5)
l2, = ax2.plot(s2.value2, label = 'label2', color='tab:red', linewidth=2.5)
plt.legend([l1, l2],['Number of Patients', 'Medical Expenses'])
ax2.yaxis.grid(False)
ax1.xaxis.grid(False)
plt.plot()
plt.show()
```



Software, Libraries Used:

```
import pandas as pd
import numpy as np
from pandas import DataFrame, to_datetime
from datetime import datetime
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.preprocessing import MinMaxScaler
from prophet import Prophet
import pmdarima as pm
from keras.models import Sequential
from keras.layers import LSTM, GRU, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow import keras
from tensorflow.keras import layers
from keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.metrics import mean_absolute_error, mean_squared_error
from math import sqrt
```

Methodology of the models and results

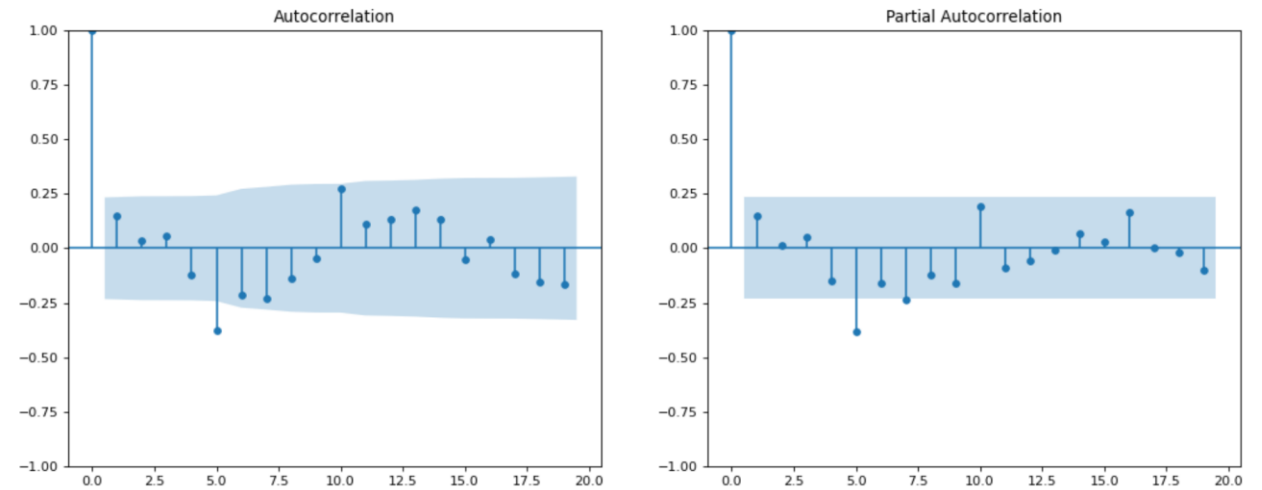
SARIMA Model

I am using two methods to determine the parameters of the models:

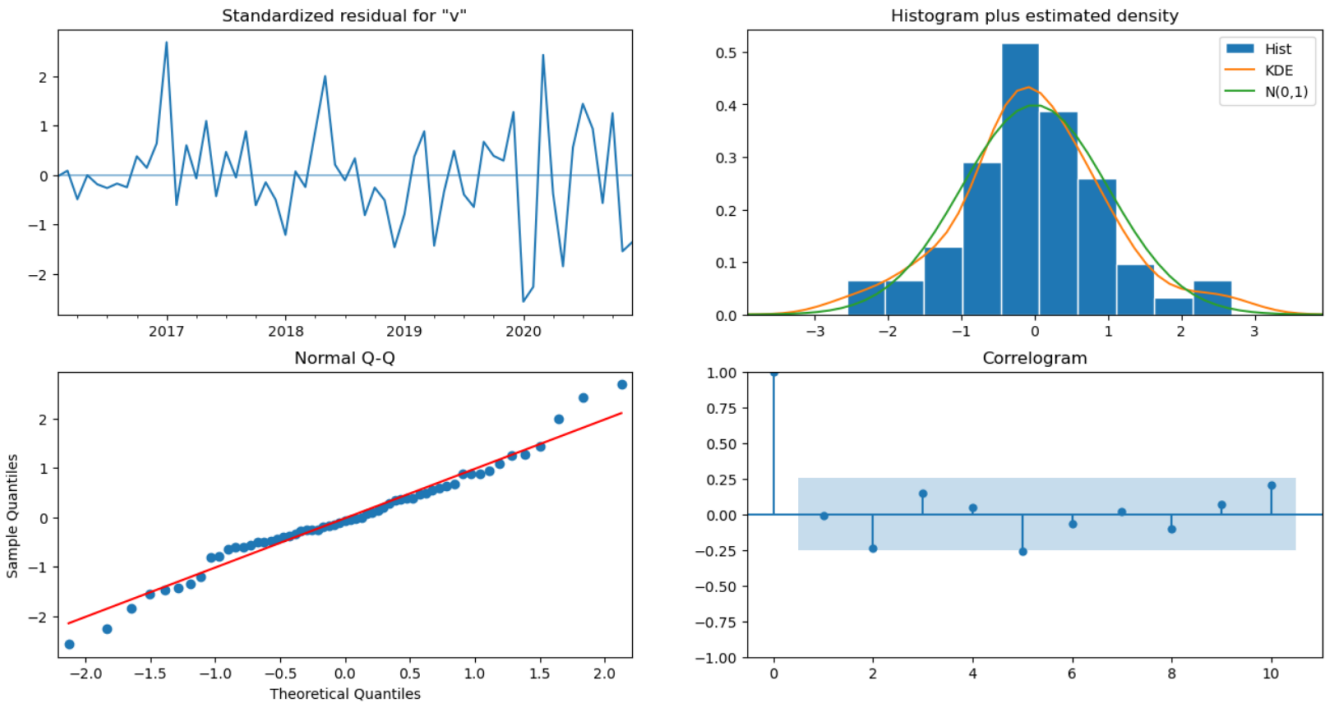
- **The Article method:**

1. ACF, PACF plots to determine (p,d,d) Visually
2. Minimum AIC to determine (P,D,Q)
3. Check if the residuals are normally distributed

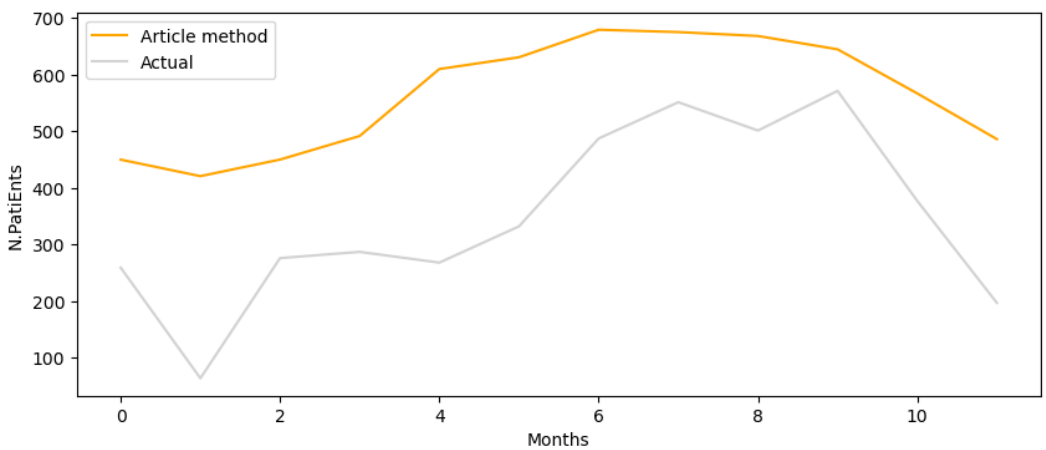
```
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(16,6), dpi= 80)
plot_acf(S.diff().dropna(), ax=ax1)
plot_pacf(S.diff().dropna(), ax=ax2);
```



```
pd.plotting.register_matplotlib_converters()
model = SARIMAX(S,
                order=(1, 1, 0),
                seasonal_order=(2, 1, 3, 12),
                enforce_invertibility=False,
                )
results = model.fit()
results.plot_diagnostics(figsize=(16, 8))
plt.show()
results.summary()
```




```
results = model.fit()
pred_dynamic = results.get_prediction(start=train_len, dynamic=True)
Y_pred1 = np.array(pred_dynamic.predicted_mean)
Y_true = np.array(y_to_test)

fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10,4)
plt.plot(Y_pred1, label="Article method", color="orange")
plt.plot(Y_true, label="Actual", color="lightgray")
plt.legend(loc="best")
plt.xlabel('Months', fontsize=10)
plt.ylabel('N.PatiEnts', fontsize=10)
```



My method :

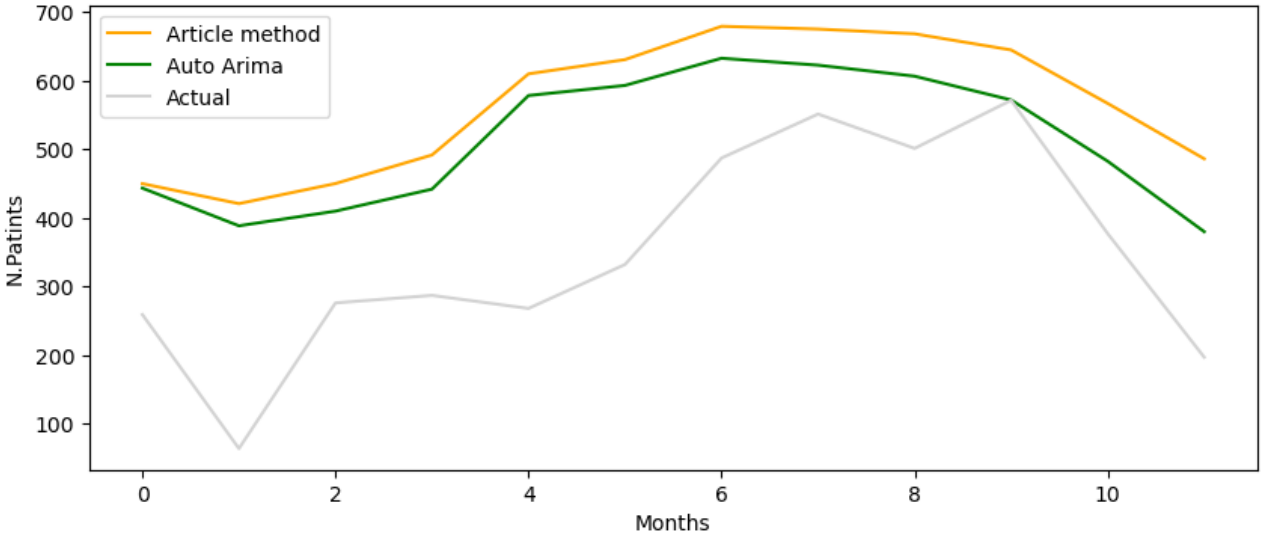
1. Auto Arima to determine all parameters based on minimum AIC

```
#ARIMA
S = s1
y_to_train = S.loc['2015-01-1':'2019-12-01']
y_to_test = S.loc['2020-01-1':]
train_len=len(y_to_train)

model = pm.auto_arima(y_to_train,
                      seasonal=True, m=12,
                      d=0, D=1,
                      start_p=0, start_q=0,
                      max_p=2, max_q=2,
                      max_P=2, max_Q=2,
                      stepwise=True,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      )
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=0.30 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=630.303, Time=0.02 sec


Best model:  ARIMA(2,0,1)(2,1,0)[12] intercept
Total fit time: 11.030 seconds
                                      SARIMAX Results
==========================================================================================
```

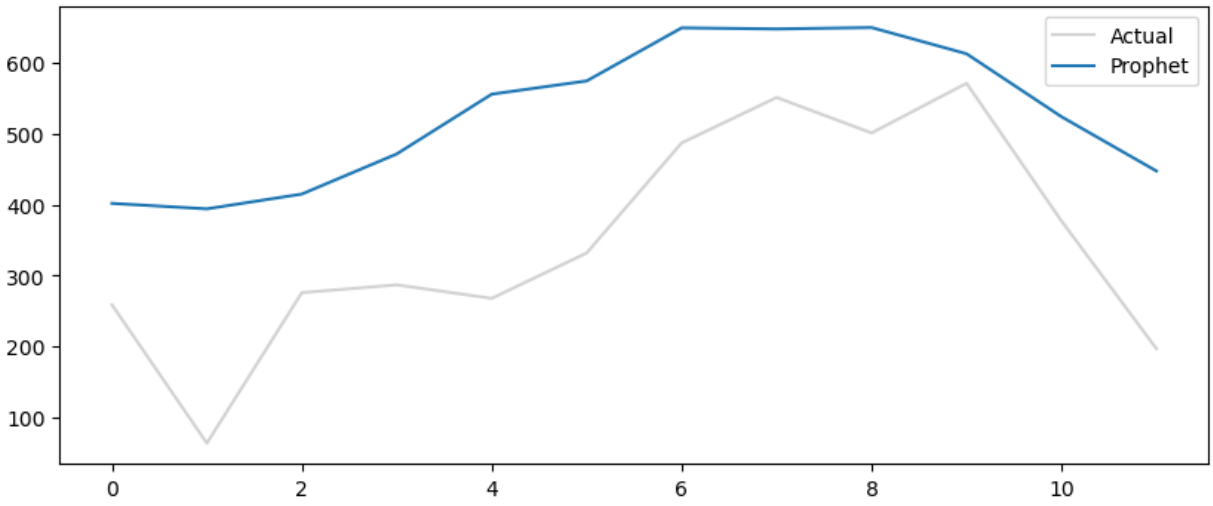
2. Fit the model and Plot the results

```
model = SARIMAX(S,
                order=(2, 0, 1),
                seasonal_order=(2, 1, 0, 12),
                enforce_invertibility=False,
                )
results = model.fit()
pred_dynamic = results.get_prediction(start=train_len, dynamic=True)
Y_pred11 = np.array(pred_dynamic.predicted_mean)
Y_true = np.array(y_to_test)

fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10,4)
plt.plot(Y_pred1, label="Article method", color="orange")
plt.plot(Y_pred11, label="Auto Arima ", color="green")
plt.plot(Y_true, label="Actual", color="lightgray")
plt.legend(loc="best")
plt.xlabel('Months', fontsize=10)
plt.ylabel('N.PatiEnts', fontsize=10)
```



Facebook Prophet

```
#FB.Prophet
S = s1
Strain = S.loc['2015-01-1':'2019-12-01']
Stest = S.loc['2020-01-1':]

# prepare expected column names
S['ds'] = S.index
S.rename(columns = {'value':'y'}, inplace = True)
# create test dataset, remove last 12 months
train = S.drop(S.index[-12:])
# define the model
model = Prophet()
# fit the model
model.fit(train)
# define the period for which we want a prediction
future = list()
for i in range(1, 13):
    date = '2020-%02d' % i
    future.append([date])
future = DataFrame(future)
future.columns = ['ds']
future['ds'] = to_datetime(future['ds'])
# use the model to make a forecast
forecast = model.predict(future)
# calculate MAE between expected and predicted values for december
Y_true = S['y'][-12:].values
Y_pred2 = forecast['yhat'].values
mae = mean_absolute_error(Y_true, Y_pred2)
print('MAE: %.3f' % mae)
# plot expected vs actual

fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10,4)
pyplot.plot(Y_true, label='Actual')
pyplot.plot(Y_pred2, label='Prophet')
pyplot.legend()
pyplot.show()
```



LSTM

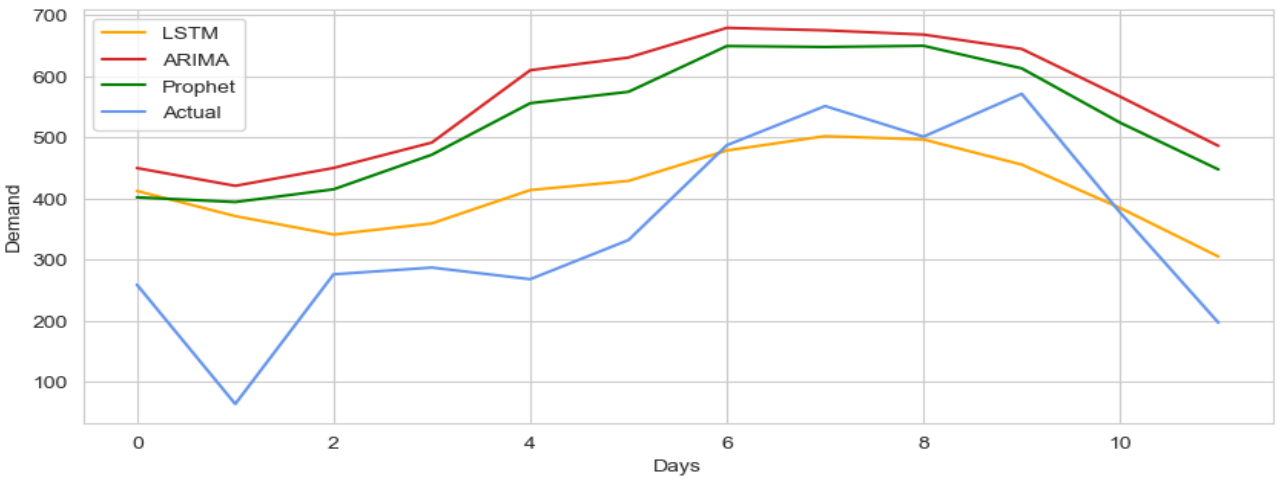
```
#LSTM
S = s1
train = S.loc['2015-01-1':'2019-12-01']
test = S.loc['2020-01-1':]
train_len = len(train)

scaler = MinMaxScaler()
scaler.fit(train)
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)

n_input = 12
n_features=1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=3)

model = Sequential()
model.add(LSTM(16, return_sequences=True, activation='relu', input_shape=(n_input, n_features)))
model.add(LSTM(32, return_sequences=False, activation='relu', input_shape=(n_input, n_features)))
model.add(Dropout(0.3))
model.add(Dense(1))
model.compile(optimizer= Adam(learning_rate = 0.01), loss='mae')
model.summary()
model.fit_generator(generator,epochs=100)
model.history.history.keys()
loss_per_epoch = model.history.history['loss']

first_eval_batch = scaled_train[-12:]
first_eval_batch = first_eval_batch.reshape((1, n_input, n_features))
test_predictions = []
first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))
for i in range(len(test)):
    current_pred = model.predict(current_batch)[0]
    test_predictions.append(current_pred)
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)

true_predictions = scaler.inverse_transform(test_predictions)
test['Predictions'] = true_predictions
Y_true = test['value']
Y_pred3 = test["Predictions"]
```

Method	Parameters	Values
LSTM	Layers	3
	No. of neurons	{16,32,64}
	Learning rate	0.01
	Dropout	0.3
	Optimizer	Adam
	Batch size	3
	Maximum Epochs	1,000
	Activation Function	Linear

- Plot all the results

```
fig = matplotlib.pyplot.gcf()
fig.set_size_inches(10,4)
plt.plot(Y_pred3, label="LSTM", color="orange")
plt.plot(Y_pred1, label="ARIMA", color="tab:red")
plt.plot(Y_pred2, label="Prophet", color="green")
plt.plot(Y_true, label="Actual", color="cornflowerblue")
plt.legend(loc="best")
plt.xlabel('Days', fontsize=10)
plt.ylabel('Demand', fontsize=10)
```

My Results



Their Results

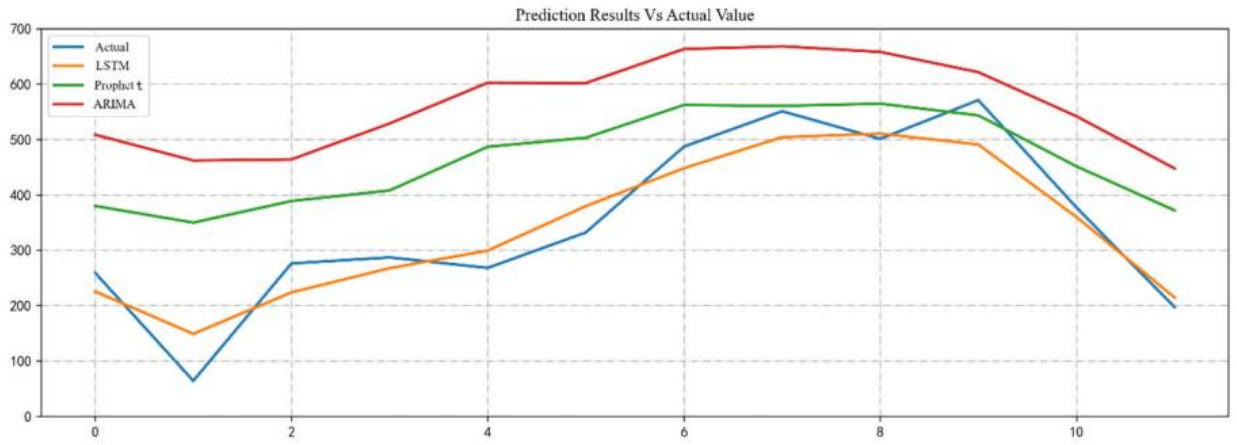


FIGURE 7
Visualizations of predicted and actual values for SARIMA, Prophet, and LSTM models.

Table of the means

	LSTM	PROPHET	SARIMA
RMSE	94.44	181.01	216.61
MAE	124.29	197.43	231.98

Conclusions:

- The results are very consistent : the LSTM is the best model, after it comes the F.prophet, and the worst of all is the SARIMA model.

Bibliography :

- https://pubmed.ncbi.nlm.nih.gov/35937210/
- **Download the file and the data here :** https://www.frontiersin.org/articles/10.3389/fpubh.2022.946563/full
- **Supplementary material :**
- https://towardsdatascience.com/creating-an-arima-model-for-time-series-forecasting-ff3b619b848d