```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from google.colab import files
from google.colab import drive
import io
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
from statsmodels.tsa.base.datetools import dates_from_str
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import datetime as datetime
import warnings
#from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
```

uploading file

```
data=files.upload()
```

Choose Files dengue_dataset.csv

- **dengue_dataset.csv**(text/csv) - 172848 bytes, last modified: 9/28/2023 - 100% done

```
Saving dengue_dataset.csv to dengue_dataset.csv
```

```
#uploading only San Juan training data
sj_data = pd.read_csv(io.StringIO(data['dengue_dataset.csv'].decode('utf-8')))
```

```
sj_data.head()
```

	year	weekofyear	week_start_date	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_a
0	1990	18	30-04-90	0.122600	0.103725	0.198483	0.177617	12.42	297.572857	
1	1990	19	07-05-90	0.169900	0.142175	0.162357	0.155486	22.82	298.211429	
2	1990	20	14-05-90	0.032250	0.172967	0.157200	0.170843	34.54	298.781429	
3	1990	21	21-05-90	0.128633	0.245067	0.227557	0.235886	15.36	298.987143	
4	1990	22	28-05-90	0.196200	0.262200	0.251200	0.247340	7.52	299.518571	

5 rows × 24 columns

Univariate Time Series Model on Weekly Data for San Juan

Note the week are not periodic, every year the week starts from 1st date of the year

```
ts_data = sj_data[['week_start_date', 'total_cases']].copy()
ts_data["week_start_date"] = pd.to_datetime(ts_data['week_start_date'], format='%d-%m-%y')
ts_data.set_index('week_start_date', inplace=True)
```

```
ts_data.head()
```

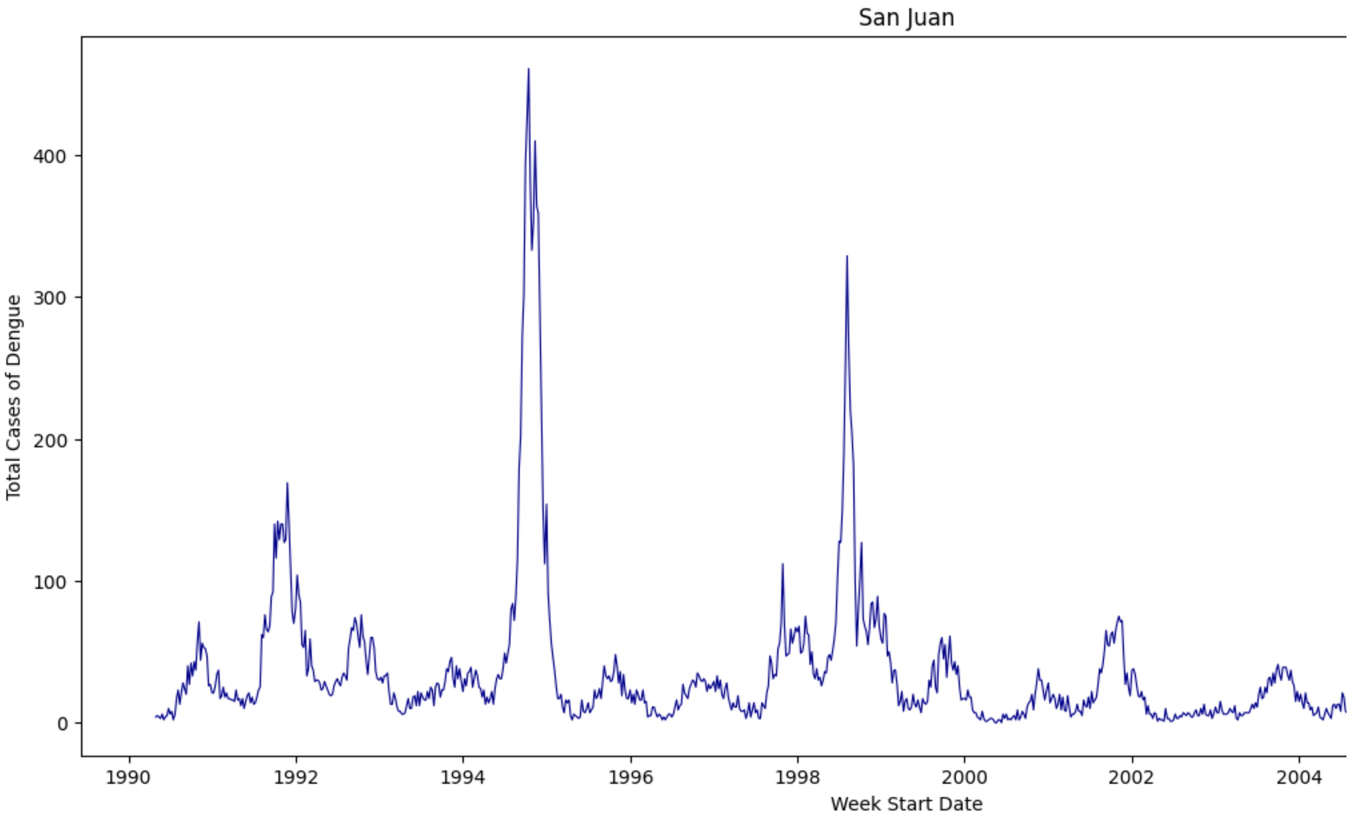
	total_cases
week_start_date	
1990-04-30	4
1990-05-07	5
1990-05-14	4
1990-05-21	3
1990-05-28	6

```
ts_data.index.year
```

```
Int64Index([1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990, 1990,
            ...
            2008, 2008, 2008, 2008, 2008, 2008, 2008, 2008, 2008, 2008],
           dtype='int64', name='week_start_date', length=936)
```

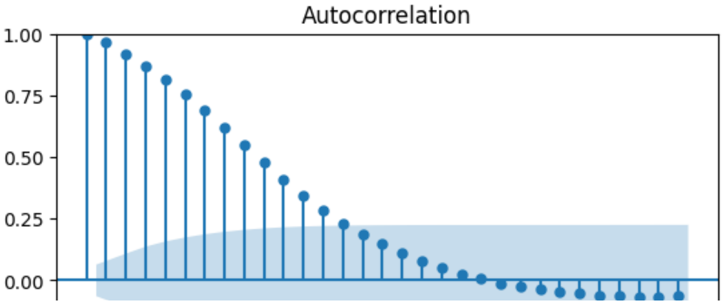
```
plt.figure(figsize=(16,7))
plt.plot(ts_data, color="darkblue", linewidth=.8)
plt.xlabel("Week Start Date")
plt.ylabel("Total Cases of Dengue")
plt.title("San Juan")
```

```
Text(0.5, 1.0, 'San Juan')
```



ACF gives autocorrelation(correlation coefficient) between y and shifted y

```
plot_acf(ts_data)
```




```
# pearson correlation coefficient
def compute_covariance(x,y):
    x = np.asarray(x)
    y = np.asarray(y)
    mean_x = np.mean(x)
    mean_y = np.mean(y)
    std_x = np.std(x)
    std_y = np.std(y)
    l = len(x)
    return np.sum((x - mean_x)*(y-mean_y))/(std_x*std_y*l)
```

```
# computing variance indicating ACF
eda_ts = ts_data.copy()
shift = 13
eda_ts['shift'+str(shift)] = eda_ts.shift(shift)
eda_ts.dropna(inplace=True)
eda_ts.head()
compute_covariance(x=eda_ts['total_cases'], y = eda_ts['shift'+str(shift)])
```

```
0.23285224943845023
```

```
plot_pacf(ts_data)
```

Partial Autocorrelation

p,d and q

1. p is auto regressive part
2. d stands for differencing, used in time series with trend, necessary to remove the trend to make time series stationary, as all time series are based on assumption that time series is stationary
3. q is for moving average

```
from statsmodels.tsa.stattools import adfuller

# Test for stationarity
result = adfuller(ts_data)
print('p-value:', result[1])
```

```
p-value: 5.1473186737591e-09
```

ARIMA Model for Univariate Time Series Model on Weekly Data Without Train Data Updation on San Juan

```
X = ts_data.values
size = int(len(X) * 0.855)
train, test_actual = X[0:size], X[size:len(X)]
print("train size: %f, test size: %f, total size %f: "%(len(train), len(test_actual), len(ts_data)))

model = sm.tsa.arima.ARIMA(train, order=(2,0,1))
model_fit = model.fit()

#model = ARIMA(train, order=(2,0,1))
#model_fit = model.fit()
test_pred = [] # creating list to store new prediction for test set
conf_int_low = [] #higher confidence interval
conf_int_high = [] #lower confidence interval
forecast_results = model_fit.get_forecast(steps=len(test_actual))
forecast = forecast_results.predicted_mean
conf_int = forecast_results.conf_int()

for t in range(len(test_actual)):
    conf_int_low.append(conf_int[t][0]) # adding low range of confidence interval
    conf_int_high.append(conf_int[t][1]) # adding upper range of confidene interval
    test_pred.append(forecast[t]) # adding single week prediction
    print('iteration :%s, predicted=%f, actual=%f' % (t+1, test_pred[t], test_actual[t]))
```

```
train size: 800.000000, test size: 136.000000, total size 936.000000:
iteration :1, predicted=130.003011, actual=112.000000
iteration :2, predicted=127.778873, actual=82.000000
iteration :3, predicted=124.525529, actual=73.000000
iteration :4, predicted=120.429240, actual=43.000000
iteration :5, predicted=115.663441, actual=55.000000
iteration :6, predicted=110.387940, actual=55.000000
iteration :7, predicted=104.748430, actual=53.000000
iteration :8, predicted=98.876260, actual=46.000000
iteration :9, predicted=92.888462, actual=43.000000
iteration :10, predicted=86.887967, actual=29.000000
iteration :11, predicted=80.964013, actual=22.000000
iteration :12, predicted=75.192683, actual=26.000000
iteration :13, predicted=69.637574, actual=13.000000
iteration :14, predicted=64.350553, actual=17.000000
iteration :15, predicted=59.372579, actual=8.000000
iteration :16, predicted=54.734582, actual=13.000000
iteration :17, predicted=50.458358, actual=10.000000
iteration :18, predicted=46.557492, actual=17.000000
iteration :19, predicted=43.038256, actual=19.000000
iteration :20, predicted=39.900513, actual=9.000000
iteration :21, predicted=37.138580, actual=9.000000
iteration :22, predicted=34.742057, actual=9.000000
iteration :23, predicted=32.696619, actual=3.000000
iteration :24, predicted=30.984751, actual=7.000000
iteration :25, predicted=29.586433, actual=7.000000
iteration :26, predicted=28.479775, actual=0.000000
iteration :27, predicted=27.641582, actual=2.000000
iteration :28, predicted=27.047873, actual=3.000000
iteration :29, predicted=26.674336, actual=3.000000
iteration :30, predicted=26.496724, actual=1.000000
iteration :31, predicted=26.491205, actual=3.000000
iteration :32, predicted=26.634648, actual=3.000000
iteration :33, predicted=26.904872, actual=3.000000
iteration :34, predicted=27.280837, actual=7.000000
iteration :35, predicted=27.742797, actual=3.000000
iteration :36, predicted=28.272417, actual=5.000000
iteration :37, predicted=28.852845, actual=11.000000
```

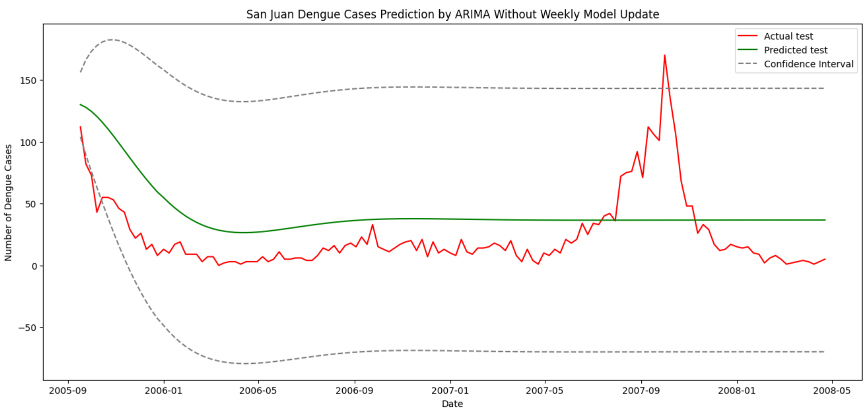
```
iteration :38, predicted=29.468765, actual=5.000000
iteration :39, predicted=30.106405, actual=5.000000
iteration :40, predicted=30.753535, actual=6.000000
iteration :41, predicted=31.399436, actual=6.000000
iteration :42, predicted=32.034853, actual=4.000000
iteration :43, predicted=32.651932, actual=4.000000
iteration :44, predicted=33.244144, actual=8.000000
iteration :45, predicted=33.806206, actual=14.000000
iteration :46, predicted=34.333986, actual=12.000000
iteration :47, predicted=34.824409, actual=16.000000
iteration :48, predicted=35.275360, actual=10.000000
iteration :49, predicted=35.685585, actual=16.000000
iteration :50, predicted=36.054592, actual=18.000000
iteration :51, predicted=36.382559, actual=15.000000
iteration :52, predicted=36.670237, actual=23.000000
iteration :53, predicted=36.918863, actual=17.000000
iteration :54, predicted=37.130077, actual=33.000000
iteration :55, predicted=37.305846, actual=15.000000
iteration :56, predicted=37.448384, actual=13.000000
iteration :57, predicted=37.560095, actual=11.000000
```

```
mse_error = mean_squared_error(test_actual, test_pred)
mae_error = mean_absolute_error(test_actual, test_pred)
print('Test Square MSE: %.3f and Test Absolute MAE: %.3f'% (mse_error, mae_error))
```

```
Test Square MSE: 1211.843 and Test Absolute MAE: 29.702
```

```
# plot
test_actual = pd.DataFrame(data=test_actual, index=ts_data[size:].index)
test_pred = pd.DataFrame(data=test_pred, index=ts_data[size:].index)
conf_int_low = pd.DataFrame(data=conf_int_low, index=ts_data[size:].index)
conf_int_high = pd.DataFrame(data=conf_int_high, index=ts_data[size:].index)


plt.figure(figsize=(16,7))
plt.plot(test_actual, color='red', label="Actual test")
plt.plot(test_pred, color='green', label="Predicted test")
plt.plot(conf_int_low, '--', color='grey', label="Confidence Interval")
plt.plot(conf_int_high, '--', color='grey')
plt.legend()
plt.title("San Juan Dengue Cases Prediction by ARIMA Without Weekly Model Update")
plt.ylabel("Number of Dengue Cases")
plt.xlabel("Date")
plt.savefig("ArimaPredictionWithoutUpdation_SJ.png")
plt.show()
```



ARIMA Model for Univariate Time Series Model on Weekly Data With Train Data Updation on San Juan

```
X = ts_data.values
size = int(len(X) * 0.855)
train, test_actual = X[0:size], X[size:len(X)]
print("train size: %f, test size: %f, total size %f: "%(len(train), len(test_actual), len(ts_data)))

train = [x for x in train] # creating list of training data, just to make it model usable
test_pred = [] # creating list to store new prediction for test set
conf_int_low = []
conf_int_high = []
for t in range(len(test_actual)):
    model = sm.tsa.arima.ARIMA(train, order=(2,0,1))
    model_fit = model.fit()

    forecast_results = model_fit.get_forecast(steps=1)
    forecast = forecast_results.predicted_mean
    conf_int = forecast_results.conf_int()

    yhat = forecast[0] # next week prediction
    conf_int_low.append(conf_int[0][0]) # adding low range of confidence interval
    conf_int_high.append(conf_int[0][1]) # adding upper range of confidene interval
    test_pred.append(yhat) # adding single week prediction
    train.append(test_actual[t]) # adding actual test value to train set to make next prediction
    print('iteration :%s, predicted=%f, actual=%f' % (t+1, test_pred[t], test_actual[t]))
```

```
train size: 800.000000, test size: 136.000000, total size 936.000000:
iteration :1, predicted=130.003011, actual=112.000000
iteration :2, predicted=108.183707, actual=82.000000
iteration :3, predicted=75.320607, actual=73.000000
iteration :4, predicted=66.459978, actual=43.000000
iteration :5, predicted=34.745423, actual=55.000000
iteration :6, predicted=49.813067, actual=55.000000
iteration :7, predicted=50.782700, actual=53.000000
iteration :8, predicted=49.350967, actual=46.000000
iteration :9, predicted=42.392133, actual=43.000000
iteration :10, predicted=39.865807, actual=29.000000
iteration :11, predicted=25.312463, actual=22.000000
iteration :12, predicted=18.714171, actual=26.000000
iteration :13, predicted=24.089764, actual=13.000000
iteration :14, predicted=10.592297, actual=17.000000
iteration :15, predicted=15.846129, actual=8.000000
iteration :16, predicted=6.653040, actual=13.000000
iteration :17, predicted=12.847008, actual=10.000000
iteration :18, predicted=9.987291, actual=17.000000
iteration :19, predicted=17.964254, actual=19.000000
iteration :20, predicted=20.188073, actual=9.000000
iteration :21, predicted=9.344268, actual=9.000000
iteration :22, predicted=9.672159, actual=9.000000
iteration :23, predicted=9.921024, actual=3.000000
iteration :24, predicted=3.588663, actual=7.000000
iteration :25, predicted=8.298333, actual=7.000000
iteration :26, predicted=8.426926, actual=0.000000
iteration :27, predicted=0.921088, actual=2.000000
iteration :28, predicted=3.432345, actual=3.000000
iteration :29, predicted=4.683383, actual=3.000000
iteration :30, predicted=4.778420, actual=1.000000
iteration :31, predicted=2.686368, actual=3.000000
iteration :32, predicted=4.971839, actual=3.000000
iteration :33, predicted=4.995244, actual=3.000000
iteration :34, predicted=5.012176, actual=7.000000
iteration :35, predicted=9.350941, actual=3.000000
iteration :36, predicted=4.894982, actual=5.000000
iteration :37, predicted=7.096567, actual=11.000000
iteration :38, predicted=13.545940, actual=5.000000
iteration :39, predicted=6.818598, actual=5.000000
iteration :40, predicted=6.842513, actual=6.000000
iteration :41, predicted=7.941696, actual=6.000000
iteration :42, predicted=7.919859, actual=4.000000
iteration :43, predicted=5.738873, actual=4.000000
iteration :44, predicted=5.793589, actual=8.000000
iteration :45, predicted=10.161743, actual=14.000000
iteration :46, predicted=16.544635, actual=12.000000
iteration :47, predicted=14.092310, actual=16.000000
iteration :48, predicted=18.266829, actual=10.000000
iteration :49, predicted=11.523499, actual=16.000000
iteration :50, predicted=18.025714, actual=18.000000
iteration :51, predicted=19.991231, actual=15.000000
iteration :52, predicted=16.527728, actual=23.000000
iteration :53, predicted=25.113425, actual=17.000000
iteration :54, predicted=18.303043, actual=33.000000
iteration :55, predicted=35.556059, actual=15.000000
iteration :56, predicted=15.533983, actual=13.000000
iteration :57, predicted=13.542674, actual=11.000000
```

```
mse_error = mean_squared_error(test_actual, test_pred)
mae_error = mean_absolute_error(test_actual, test_pred)
print('Test Square MSE: %.3f and Test Absolute MAE: %.3f'% (mse_error, mae_error))
```

```
Test Square MSE: 140.521 and Test Absolute MAE: 7.191
```

```
# plot
test_actual = pd.DataFrame(data=test_actual, index=ts_data[size:].index)
test_pred = pd.DataFrame(data=test_pred, index=ts_data[size:].index)
conf_int_low = pd.DataFrame(data=conf_int_low, index=ts_data[size:].index)
conf_int_high = pd.DataFrame(data=conf_int_high, index=ts_data[size:].index)


plt.figure(figsize=(16,7))
plt.plot(test_actual, color='red', label="Actual test")
plt.plot(test_pred, color='green', label="Predicted test")
plt.plot(conf_int_low, '--', linewidth = .5, color='grey', label="Confidence Interval")
plt.plot(conf_int_high, '--', linewidth = .5, color='grey')
plt.legend()
plt.title("San Juan Dengue Cases Prediction by ARIMA With Weekly Model Update")
plt.ylabel("Number of Dengue Cases")
plt.xlabel("Date")
plt.savefig("ArimaPredictionWithUpdation_SJ.png")
plt.show()
```