# Foundation of Data Science (25 points)

## Homework 4

Student Name: Yamini Lakshmi Narasimhan

Student Netid: yl9822

---

## Part 1: Critique this plan (10 points)

1. After a few beers your CIO invited his buddy from Green Berry consulting to propose a project using data mining to improve the targeting of the new service that you have been a principal in developing. The service has been quite successful so far, being marketed over the last 6 months via your ingenious, and very inexpensive, word-of-mouth campaign. You've already garnered a pretty large customer base without any targeting, and you've been seeing this success as your best stepping stone to bigger and better things in the firm.

After some reflection, you've decided that your best course of action is to play a key role in ensuring the success of this data mining project as well. You agree with your CIO's statement in a meeting with Green Berry, that accurate targeting might cost-effectively expand your audience substantially to consumers that word-of-mouth would not reach. You accept that what Green Berry says about the characteristics of your service is accurate.

Based on what we have covered in class, identify the four most serious weaknesses/flaws in this abridged version of Green Berry's proposal, and suggest how to ameliorate them. Your answer should be 4 bullet points, each comprising 2-4 sentences: 1-2 sentences stating each weakness, and 1-2 sentences suggesting a better alternative. Maximal credit will be given when the 4 points are as independent as possible.

```
--- ---------------------------------------------------------------------------- ---
                         Targeted Audience Expansion
                     Prepared by Green Berry Consulting, Inc.

Your problem is to expand the audience of your new service.  We (Green Berry) have a
large database of consumers who can be targeted.  We will build a predictive model
to estimate which of these consumers are the most likely to adopt the product, and
then target them with the special offer you have designed.

More specifically, we will build a decision tree (DT) model to predict adop-
```

```
tion of the service by a consumer, based on the data on your current customers of
this service.  The model will be based on their demographics and their usage of
the service. We believe that DT is the best choice of method be-
cause it is a tried-and-true modeling technique, and we can easily
interpret the model to infer whether the attributes make sense. We will apply the model to o
database of consumers, and select out those who have not yet subscribed and whom
the DT model predicts to be likely to subscribe.  To these we will send
the targeted offer. As this is a fixed-profit-per-customer service, this also
will in effect rank them by expected profit.
--- ------------------------------------------------------------------------------ ---
```

1. Input; Features Considered : Demographics and usage of the service are too limited, there could be additional features like economic status, family members etc based on the service provided
2. Model; Decision Tree : Decision tree applying to the large population of data is not suggested as

- A small change in the data can cause a large change in the structure of the decision tree causing instability.
- For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
- Decision tree often involves higher time to train the model.
- Decision tree training is relatively expensive as the complexity and time has taken are more.

Instead we can use Logistic regression or any classification model like Resnets with 1 and 0 class as predicting using the service or not.

3. Service : Time contraint: People might not need the service the time when tested but might need it later on - so this model inferencing needs to go on iteratively and regularly to see if new people might need the service at a later stage. Or we can also add some other model that forecasts when the service might be needed example ARIMA model.

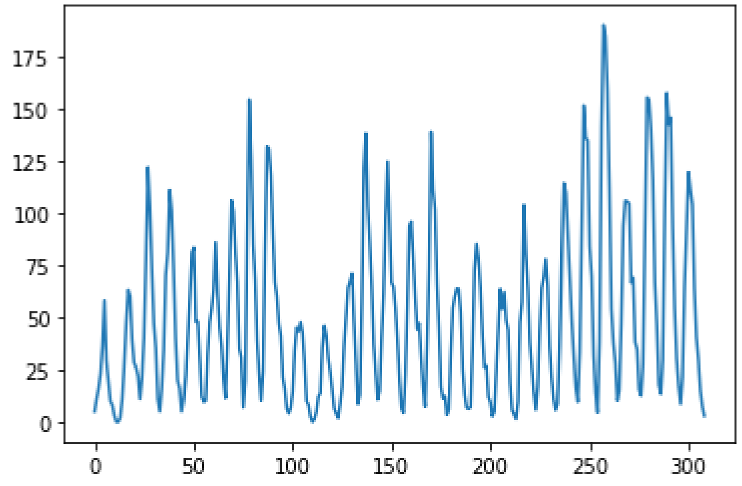
4. Ethical; Privacy: The large database that is their system needs to be verified by the people if the company can use their data for other purposes like this and they should also give them an option to unsubscribe so that its not called invasion of privacy

## Part 2: Working with Time Series (15 points + option for 4 bonus points)

Here we will analyze a timeseries of number of Yellow Fever cases from a specific Asian country by year. The data is from 1700 – 2008 (use file cases.csv for this section).

1. Load the timeseries data set, and prepare the dataset by converting the variables to date-time format (hint: use date tools and the library statsmodels). (1 point)

```
#write your code here
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
df = pd.read_csv("./cases (2).csv")
plt.plot(df["YFCASES"])
plt.show()
df['YEAR_date'] = pd.to_datetime(df['YEAR'],format='%Y')
len(df["YEAR"].unique())
```



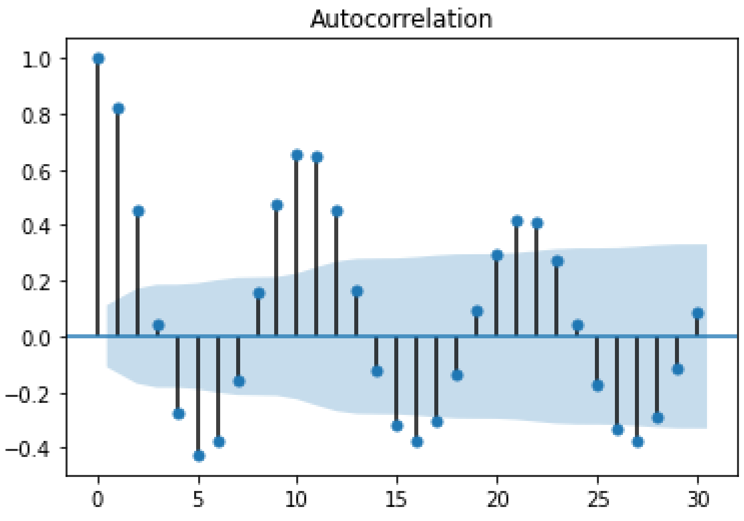
```
309
```

2. Plot the autocorrelation function (ACF) for the cases timeseries (hint: use statsmodels plot_acf for that). (2 points)

To learn more about how to interpret these graphs, you may find this useful: https://medium.com/analytics-vidhya/interpreting-acf-or-auto-correlation-plot-d12e9051cd14

```
#write your code here
import statsmodels.api as sm
figure(figsize=(14, 20), dpi=80)

from statsmodels.graphics.tsaplots import plot_acf
dta = pd.read_csv("./cases (2).csv")
df2 = dta.copy()
dta.index = pd.Index(sm.tsa.datetools.dates_from_range('1700', '2008'))
del dta["YEAR"]
plot_acf(dta.values.squeeze(), lags = 30)
plt.show()
```

```
<Figure size 1120x1600 with 0 Axes>
```



```
df2["YEAR"].nunique()
```

```
309
```

3. An approach to assess the presence of a significant autocorrelation in the data is to use the Durbin-Waton (DW) statistic. The value of the DW statistic is close to 2 if the errors are uncorrelated. What is the DW for our data? Does this suggest that the data has a relatively high or a relatively low autocorrelation? (2 point)

```
#write your code here
from statsmodels.stats.stattools import durbin_watson
from statsmodels.formula.api import ols

ols_dw = ols('YFCASES~YEAR', data=df).fit()
durbin_watson(ols_dw.resid)
```

```
0.36883767267223694
```

```
x = sm.add_constant(df["YEAR"].tolist())
model = sm.OLS(df['YFCASES'].tolist(), x).fit()
print("DW test value: " + str(durbin_watson(model.resid)))
```

```
DW test value: 0.36883767267223694
```

2 - no correlation 0-2 : positively correlated 2-4 : negatively correlated

Which means its highly positively correlated

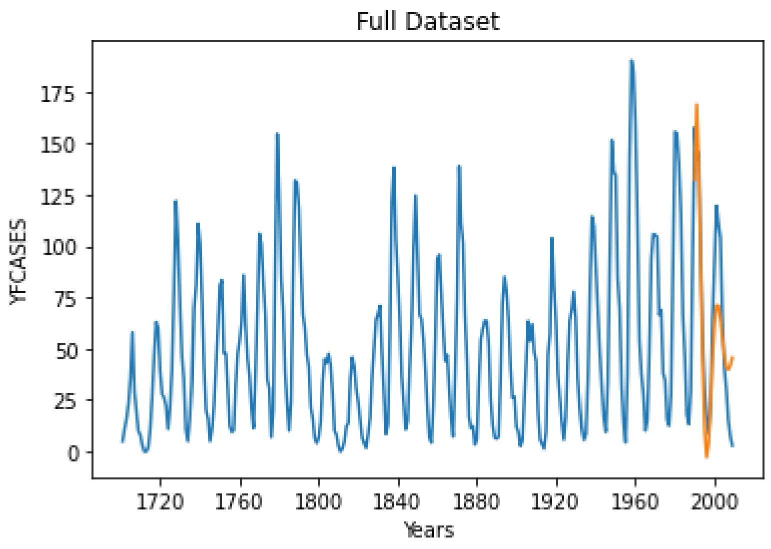
4. Now we will make a forecast on this time series. What time horizon will you use and how do you pick it? (2 points)

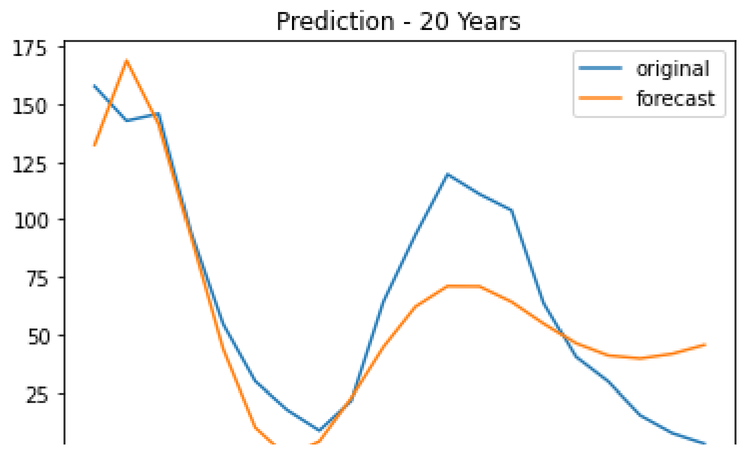
I would prefer a smaller time horizon that has the most recent data because the recent 50 years will have more impact on the next 20 so years rather than 1700(s).

USING ARMA MODEL

```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()
allset = temp_20
trainset = temp_20.loc[:'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=289, end=309)
allset["forecast"] = predictions

plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
plt.title("Full Dataset")
plt.show()
print()


filter_df = allset.dropna()
plt.plot(filter_df["YFCASES"], label="original")
plt.plot(filter_df["forecast"], label="forecast")
plt.title("Prediction - 20 Years")
plt.legend()
plt.show()


print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
```

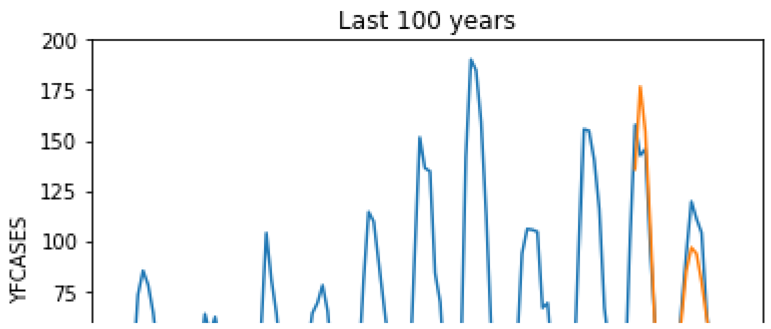




```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
```

```
forecast = pd.DataFrame()
allset = temp_20.loc['1889-12-31':]
trainset = temp_20.loc['1889-12-31':'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=100, end=121)
allset["forecast"] = predictions

plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
plt.title("Last 100 years")
plt.show()
print()

filter_df = allset.dropna()
plt.plot(filter_df["YFCASES"], label="original")
plt.plot(filter_df["forecast"], label="forecast")
plt.title("Prediction - 20 Years")
plt.legend()
plt.show()

print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  # This is added back by InteractiveShellApp.init_path()
```



```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()
allset = temp_20.loc['1939-12-31':]
trainset = temp_20.loc['1939-12-31':'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=50, end=71)
allset["forecast"] = predictions

plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
plt.title("Last 50 years")
```

```
    plt.show()

    print()

    filter_df = allset.dropna()
    plt.plot(filter_df["YFCASES"], label="original")
    plt.plot(filter_df["forecast"], label="forecast")
    plt.title("Prediction - 20 Years")
    plt.legend()
    plt.show()

    print()
    print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame.
```

Taking the last 50 years to predict next 20 years gives a lot less mean absolute devitation error that is 17.194114258546172

Full dataset -> 21.054256305362422

Last 100 years -> 17.379147771034987

Last 50 years -> 17.194114258546172

Which proves that last 50 years have the most impact on the next 20 years than the entire past 289 years

```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()
allset = temp_20.loc['1939-12-31':]
trainset = temp_20.loc['1939-12-31':'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=50, end=71)
allset["forecast"] = predictions

plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
```

```
    plt.title("Last 50 years")
    plt.show()


    filter_df = allset.dropna()
    plt.plot(filter_df["YFCASES"], label="original")
    plt.plot(filter_df["forecast"], label="forecast")
    plt.title("Prediction - 20 Years")
    plt.legend()
    plt.show()


    print()
    print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame
```

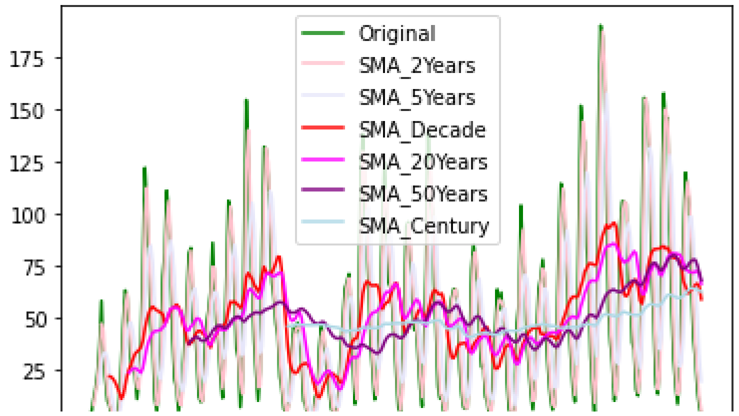
Short term horizon helps in forecasting the next 20 years and least mean absolute error of 17

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
```

5. Now use a simple moving average. Show plots for a forecast for your horizon with a series of different averaging windows, to show how you can pick one (3 points)

```
yfcases = temp_20["YFCASES"]

yfcases_average_2year = yfcases.rolling(window=2).mean()
yfcases_average_5year = yfcases.rolling(window=5).mean()
yfcases_average_decade = yfcases.rolling(window=10).mean()
yfcases_average_20 = yfcases.rolling(window=20).mean()
yfcases_average_50 = yfcases.rolling(window=50).mean()
yfcases_average_century = yfcases.rolling(window=100).mean()

plt.plot(yfcases, 'green', label='Original')
plt.plot(yfcases_average_2year, 'pink', label='SMA_2Years')
plt.plot(yfcases_average_5year, 'lavender', label='SMA_5Years')
plt.plot(yfcases_average_decade, 'red', label='SMA_Decade')
plt.plot(yfcases_average_20, 'magenta', label='SMA_20Years')
plt.plot(yfcases_average_50, 'purple', label='SMA_50Years')
plt.plot(yfcases_average_century, 'lightblue', label='SMA_Century')
plt.xlabel("Full Dataset")
plt.legend()
```

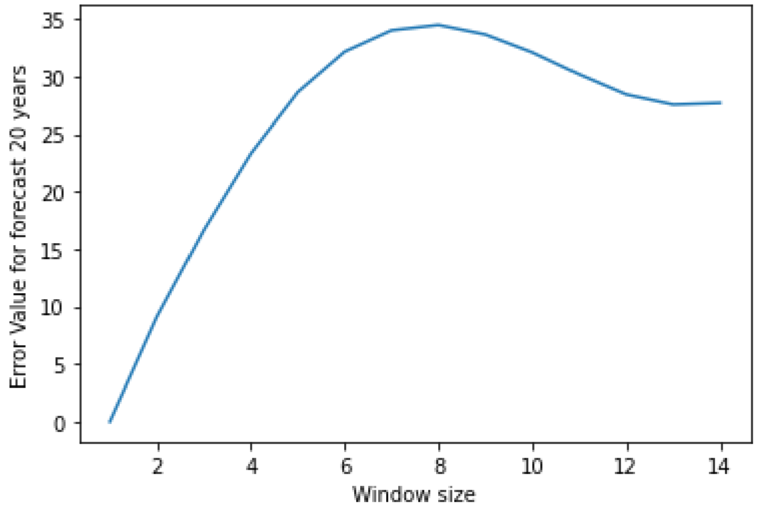
```
<matplotlib.legend.Legend at 0x7fd25983eb90>
```



20 year forecast dataset

```
yfcases = temp_20.dropna()["YFCASES"]
error_lis = []
index_lis = []
for i in range(1,15):
  yfcases_average = yfcases.rolling(window=i).mean()
  error = mean_absolute_error(yfcases[i-1:], yfcases_average.dropna())
  error_lis.append(error)
  index_lis.append(i)
plt.xlabel("Window size")
plt.ylabel("Error Value for forecast 20 years")
plt.plot(index_lis,error_lis)
```

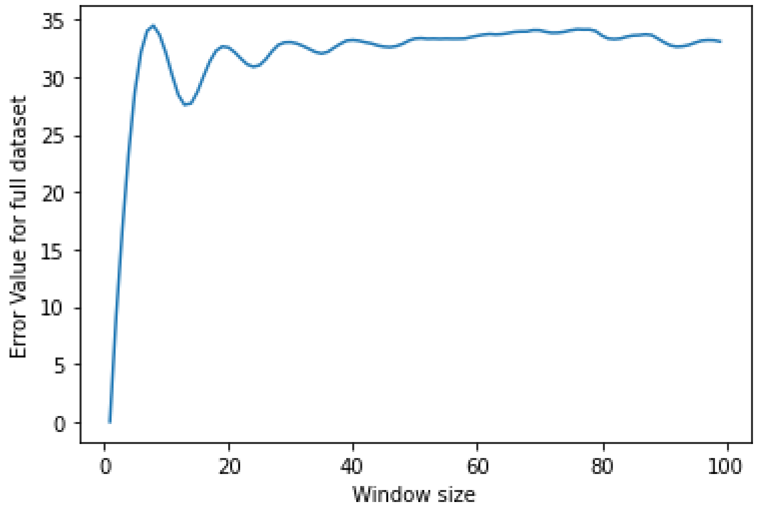
```
[<matplotlib.lines.Line2D at 0x7fd25965d250>]
```



Full Dataset

```
yfcases = temp_20["YFCASES"]
error_lis = []
index_lis = []
for i in range(1,100):
  yfcases_average = yfcases.rolling(window=i).mean()
  error = mean_absolute_error(yfcases[i-1:], yfcases_average.dropna())
  error_lis.append(error)
  index_lis.append(i)
plt.xlabel("Window size")
plt.ylabel("Error Value for full dataset")
plt.plot(index_lis,error_lis)
```

```
[<matplotlib.lines.Line2D at 0x7fd257de0b10>]
```



I am thinking of something around the range of 10-12 as we should not loose too much information by having a huge window or have all the information by using a smaller window

Window size from 10-12 is a good range because it captures the essence of the series by not focusing on each and every details like the above red line

When you look at the error rate it shows that around 10-12 the error is dipping after which its increasing again so i guess window of that size is okay

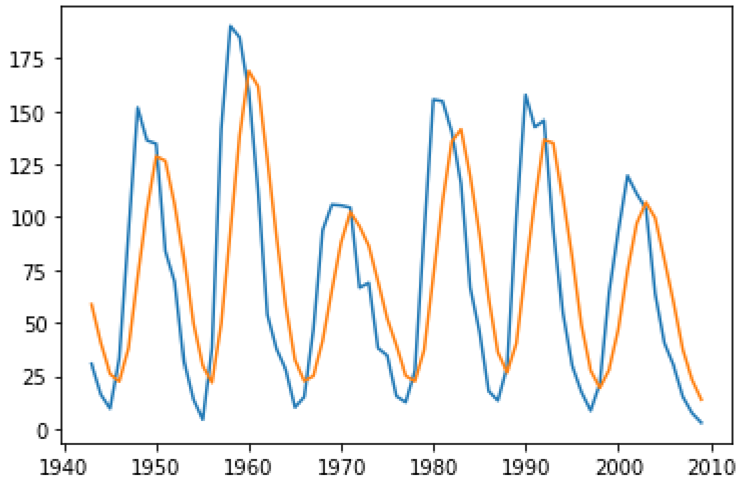
6. Will a weighted moving average be helpful here? (2 points)

Weighted Moving Average - giving customised weights to specific years like recent years might be of great help here because we have the ability to vary the weights as the most recent years have a greater impact on next few years than the impact that 1720 data has on 2009.

So Weighted Moving average will definitely help

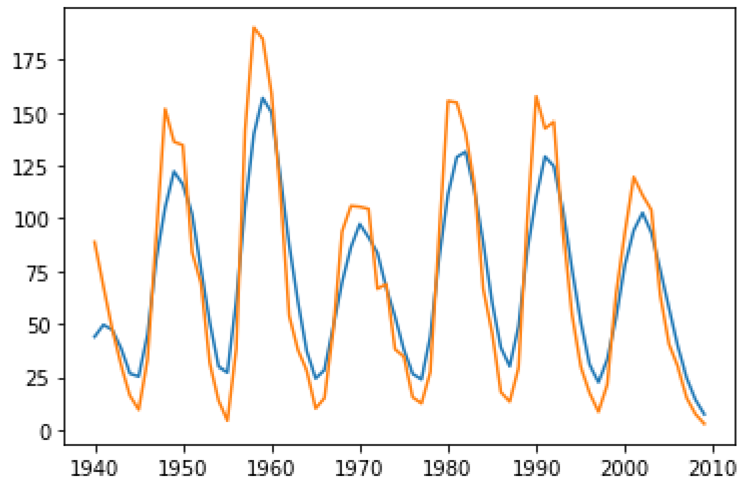
```
temp_20 = dta.copy()
trainset = temp_20.loc['1939-12-31':]
trainset["forecast"] = trainset.rolling(window=4).mean()
trainset.dropna(inplace = True)

plt.plot(trainset)
plt.show()


print()
print("ERROR VALUE",mean_absolute_error(trainset["YFCASES"]['1989-12-31':'2009-12-31'
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  This is separate from the ipykernel package so we can avoid doing imports unti
/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py:311: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  return func(*args, **kwargs)
```



```
ERROR VALUE 31.221250000000005
```

```
import numpy as np
def wma(arr, period):
    kernel = np.arange(period, 0, -0.75)
    kernel = np.concatenate([np.zeros(period - 1), kernel / kernel.sum()])
    print("KERNEL",kernel)
    return np.convolve(arr, kernel, 'same')

allset = temp_20.loc['1939-12-31':]
allset['wma'] = wma(allset['YFCASES'], 4)
plt.plot(allset['wma'])
plt.plot(allset['YFCASES'])
plt.show()

print()
print("ERROR VALUE",mean_absolute_error(allset["YFCASES"][:-20], allset["wma"][:-20])
KERNEL [0.         0.         0.         0.31372549 0.25490196 0.19607843
 0.1372549  0.07843137 0.01960784]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  if __name__ == '__main__':
```



```
ERROR VALUE 17.480196078431376
```

In this case convolve flips the kernel and convolves it with the df that is kernel = [0.1 0.2 0.3 0.4 0 0 0] - which means latest years are getting more weights than the past years and this is predicted with an error rate of 17.480196078431376

when compared to the same window size 4 simple moving average which is predicted with an error rate of 31.221250000000005

This clearly suggests weighted moving average is better than simple moving average

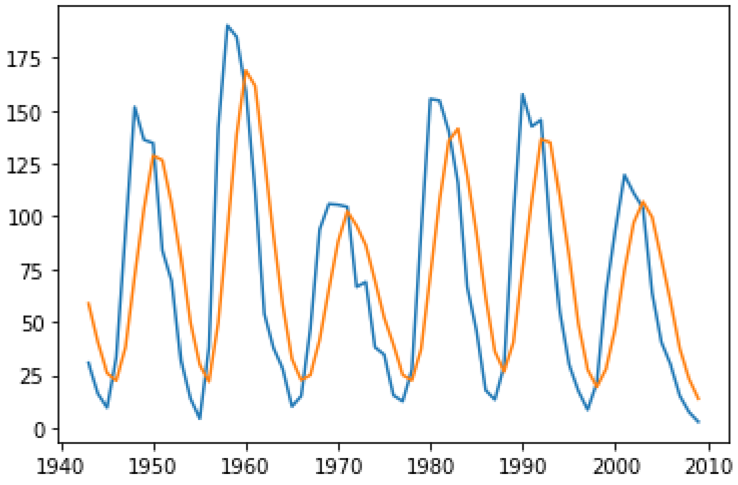
7. Evaluate your selected moving average with an appropriate evaluation metric. What do you use and what does it show you? (3 points)

Using Rolling function()

I used Mean absolute deviation as a error calculation to choose which performs better

```
temp_20 = dta.copy()
trainset = temp_20.loc['1939-12-31':]
trainset["forecast"] = trainset.rolling(window=4).mean()
trainset.dropna(inplace = True)
plt.plot(trainset)
plt.show()

print()
print("ERROR VALUE",mean_absolute_error(trainset["YFCASES"]['1989-12-31':'2009-12-31'
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  This is separate from the ipykernel package so we can avoid doing imports unti
/usr/local/lib/python3.7/dist-packages/pandas/util/_decorators.py:311: SettingWi
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  return func(*args, **kwargs)
```



```
ERROR VALUE 31.221250000000005
```

USING ARIMA MODEL

TESTSET FORECAST - Keeping testset constant - 20 years and varying trainset as 50 years, 100 years and full data set

```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()
allset = temp_20
trainset = temp_20.loc[:'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=289, end=309)
allset["forecast"] = predictions
```

```
plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
plt.title("Full Dataset")
plt.show()

print()
filter_df = allset.dropna()
plt.plot(filter_df["YFCASES"], label="original")
plt.plot(filter_df["forecast"], label="forecast")
plt.title("Prediction - 20 Years")
plt.legend()
plt.show()

print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
```

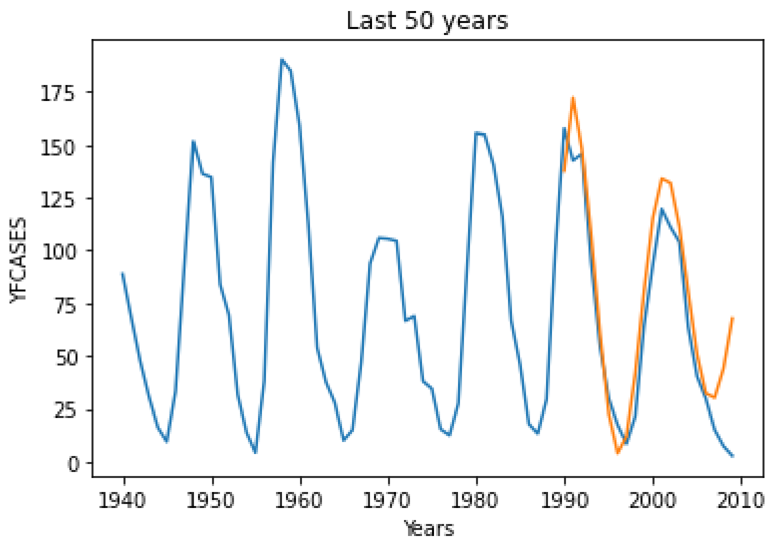
Full Dataset

```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()
allset = temp_20.loc['1889-12-31':]
trainset = temp_20.loc['1889-12-31':'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=100, end=121)
allset["forecast"] = predictions

plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
plt.title("Last 100 years")
plt.show()

print()
filter_df = allset.dropna()
plt.plot(filter_df["YFCASES"], label="original")
plt.plot(filter_df["forecast"], label="forecast")
plt.title("Prediction - 20 Years")
plt.legend()
plt.show()

print()

print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

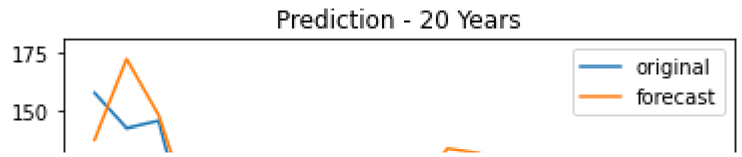
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.
```

```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()
allset = temp_20.loc['1939-12-31':]
trainset = temp_20.loc['1939-12-31':'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=50, end=71)
allset["forecast"] = predictions

plt.plot(allset["YFCASES"], label="original")
plt.plot(allset["forecast"], label="forecast")
plt.xlabel("Years")
plt.ylabel("YFCASES")
plt.title("Last 50 years")
plt.show()


filter_df = allset.dropna()

plt.plot(filter_df["YFCASES"], label="original")
plt.plot(filter_df["forecast"], label="forecast")

plt.title("Prediction - 20 Years")
plt.legend()

print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  # This is added back by InteractiveShellApp.init_path()
```



```
ERROR VALUE 17.194114258546172
```



TESTSET FORECAST - Keep Testset constant and trainset varying

TRAINSET : FULL DATA, 100 years, 50 years

TESTSET : 20 years

I still used Mean Absolute Deviation for that

Using full dataset for 20 years forecast : 21.054256305362422

Using last 50 years dataset for 20 years forecast : 17.379147771034987

Using last 100 years for 20 years forecast : 17.194114258546172

---

TRAINSET FORECAST

Keep trainset full dataset

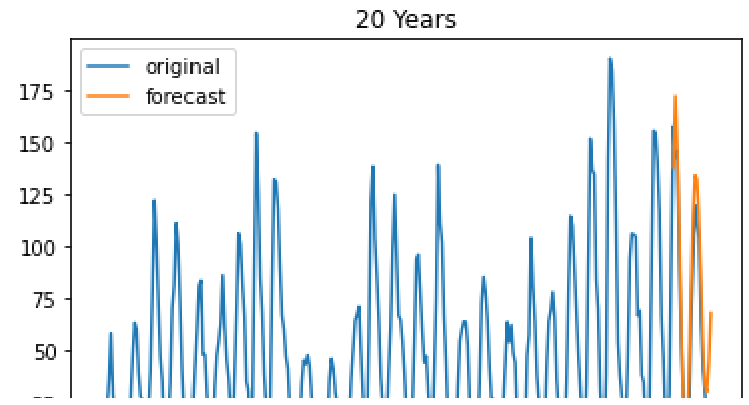
and Testset varying that is 20, 50 and 100 years

When you forecast the next 20, 50 and 100 years then 20 years is a lot easier because not a lot changes within 20 years

Forecasts are more accurate for shorter than longer time horizons. The shorter the time horizon of the forecast, the lower the degree of uncertainty. Data do not change very much in the short run.

```
from statsmodels import tsa
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
temp_20 = dta.copy()
forecast = pd.DataFrame()

trainset = temp_20.loc['1939-12-31':'1989-12-31']
model = sm.tsa.ARMA(trainset, order=(5,0,1)).fit()

predictions = model.predict(start=50, end=71)
temp_20["forecast"] = predictions

plt.plot(temp_20["YFCASES"], label="original")
plt.plot(temp_20["forecast"], label="forecast")
plt.title("20 Years")
plt.legend()

filter_df = temp_20.dropna()

print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

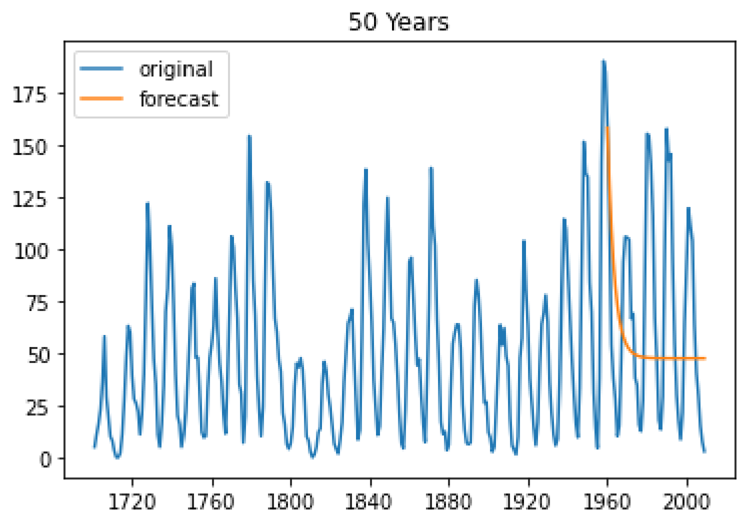
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)

ERROR VALUE 17.194114258546172
```



```
from google.colab import drive
drive.mount('/content/drive')


from statsmodels import tsa
import statsmodels.api as sm
temp_50 = dta.copy()
forecast = pd.DataFrame()

trainset = temp_50.loc[:'1959-12-31']
model = sm.tsa.ARMA(trainset, order=(1,1,1,1,0)).fit()

predictions = model.predict(start=259, end=309)
temp_50["forecast"] = predictions

plt.plot(temp_50["YFCASES"], label="original")
plt.plot(temp_50["forecast"], label="forecast")
plt.legend()
plt.title("50 Years")
```

```
filter_df = temp_50.dropna()

print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
    /usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
    statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
    been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
    between arima and model) and
    statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

    statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
    is both well tested and maintained.

    To silence this warning and continue using ARMA and ARIMA until they are
    removed, use:

    import warnings
    warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                            FutureWarning)
    warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                            FutureWarning)

      warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
    /usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
      % freq, ValueWarning)

    ERROR VALUE 42.27122554327288
```



```
#place your code here
from statsmodels import tsa
import sklearn.metrics
import statsmodels.api as sm

temp_100 = dta.copy()
forecast = pd.DataFrame()
```

```
trainset = temp_100.loc[:'1909-12-31']

model = sm.tsa.ARMA(trainset, order=(1,1,1,1,0)).fit()
predictions = model.predict(start=209, end=309)
temp_100["forecast"] = predictions

plt.plot(temp_100["YFCASES"], label="original")
plt.plot(temp_100["forecast"], label="forecast")
plt.legend()
plt.title("100 Years")

filter_df = temp_100.dropna()

print()
print("ERROR VALUE",mean_absolute_error(filter_df["YFCASES"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
```

TRAINSET : FULL DATA - TESTSET

TESTSET : 20 years, 50 years and 100 years

I used Mean Absolute Deviation and it shows that short term horizon forecast is better and has less error when compared to mid term and long term forecast because shorter the time horizon of the forecast, the lower the degree of uncertainty. Data do not change very much in the short run.

100 years forecast : 40.44988435481796

50 years forecast : 42.27122554327288

20 years forecast : 17.373933952775204

After a point of time 50 and 100 years forecast tends to saturate like shown in the plots above as there is not much behavior that can be replicated. So short term forecasting is what I chose.

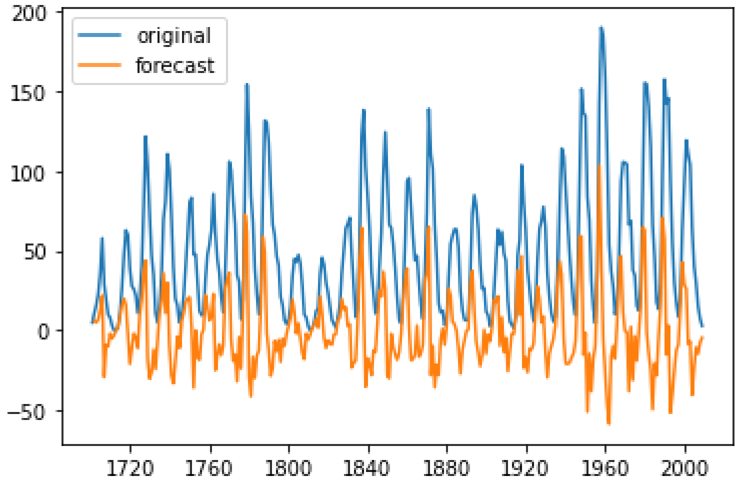
8. (Bonus). You can also investigate an ARIMA model for forecasting. First remove any trend/seasonality (hint, use differencing). What parameters for the AR and MA components of the model do you choose? How does the forecast using ARIMA compare to just the moving average? (up to 4 bonus points).

## Differencing and Removing Seasonality

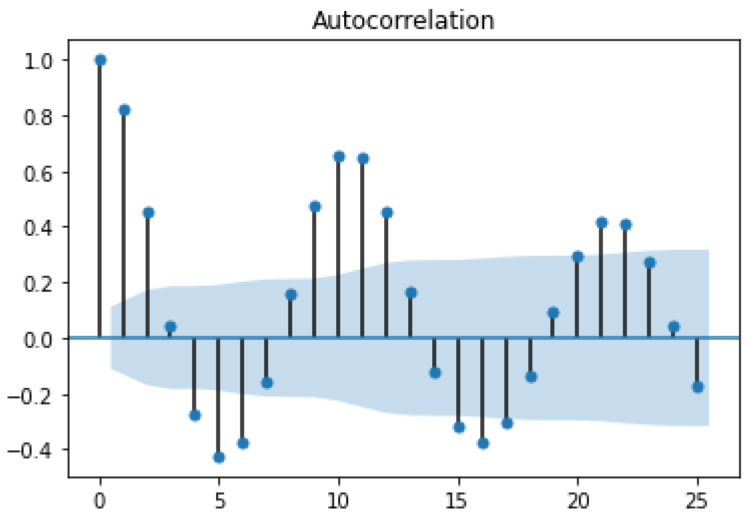
```
from statsmodels.tsa.stattools import adfuller
from numpy import log
result = adfuller(dta.dropna())
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])

print("p-value is greater than 0.05 so it requires differencing and the current time
```

```
df1 = dta.copy()
l = df1.diff()
df1["diff"] = l
plt.plot(df1["YFCASES"], label="original")
plt.plot(df1["diff"], label="forecast")
plt.legend()
plt.show()
plt.show()
```

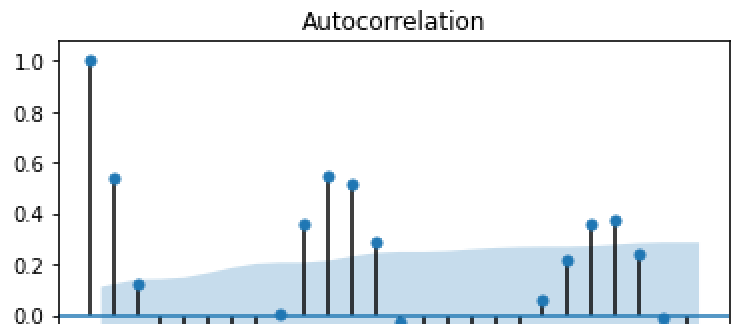
```
    ADF Statistic: -2.837781
    p-value: 0.053076
    p-value is greater than 0.05 so it requires differencing and the current time se
```



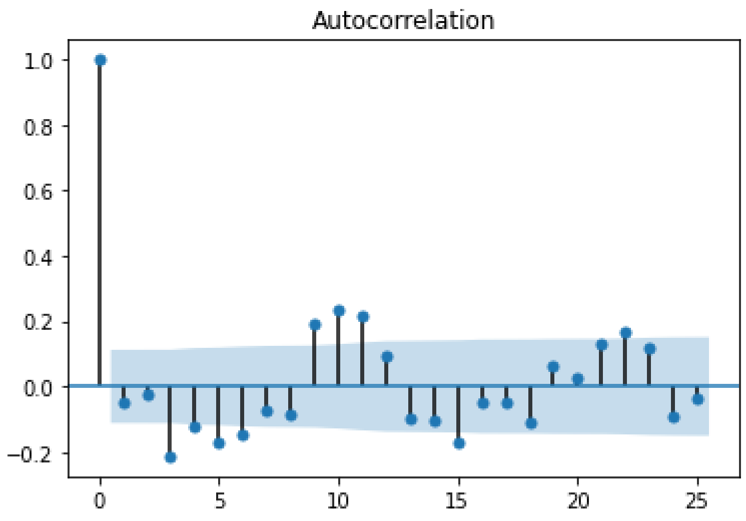
```
# No Differencing
dta_diff = dta.copy()
plot_acf(df1["YFCASES"])
plt.show()
```



```
# 1st Differencing
dta_firstdiff = dta_diff.diff()
dta_firstdiff.iloc[0]=0
plot_acf(dta_firstdiff.dropna())
plt.show()
```



```
# 2nd Differencing
dta_seconddiff = dta_firstdiff.diff()
dta_seconddiff.iloc[0]=0
plot_acf(dta_seconddiff.dropna())
plt.show()
```

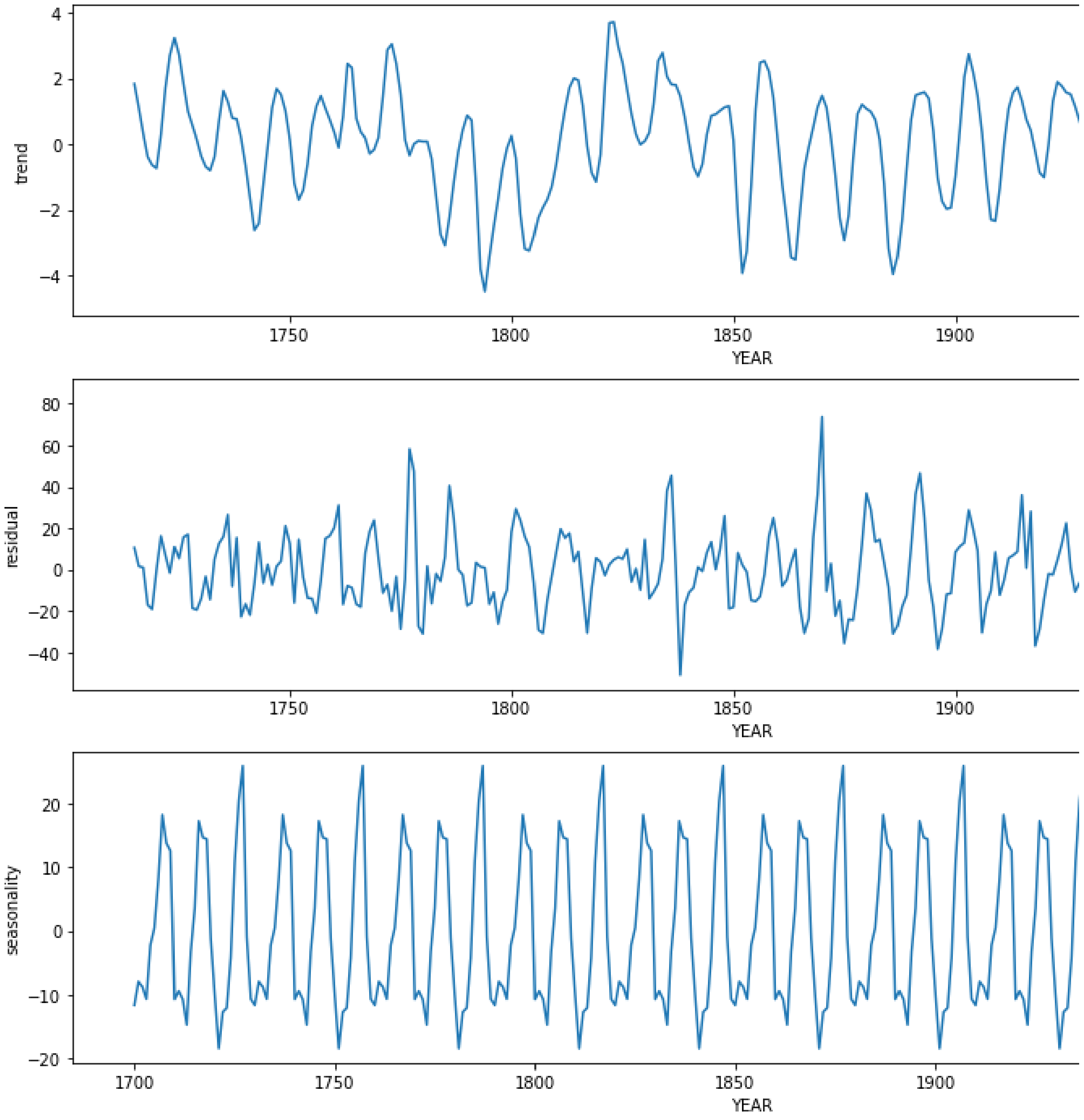


Looks like 2 order differencing is overdifferencing so lets keep d =1

As we can see that in second-order differencing the immediate lag has gone on the negative side, representing that in the second-order the series has become over the difference.

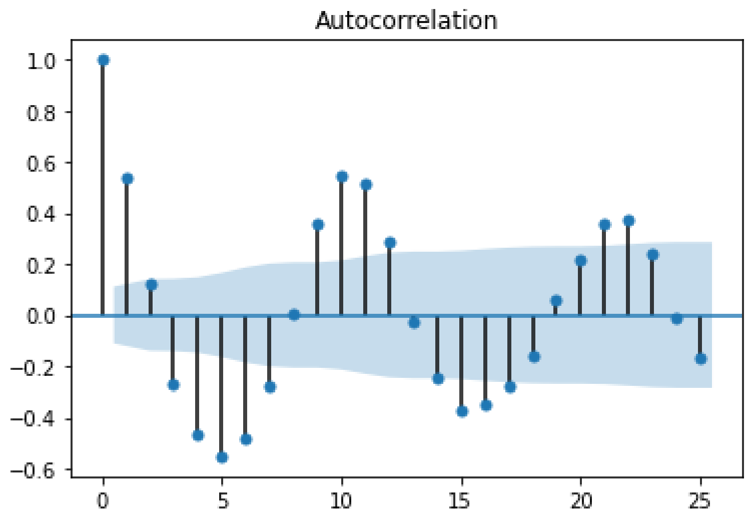
```
from statsmodels.tsa.seasonal import seasonal_decompose

d = df.diff()
df["diff"] = d["YFCASES"]
df["diff"].iloc[0] = 0
df2 = df.copy()
df2.reset_index(inplace=True)
df2 = df2.set_index('YEAR')
df2 = df2.drop(['index','YEAR_date'], axis=1)

res = seasonal_decompose(df2["diff"], model = "additive",period = 30)

fig, (ax1,ax2,ax3) = plt.subplots(3,1, figsize=(15,12))
res.trend.plot(ax=ax1,ylabel = "trend")
res.resid.plot(ax=ax2,ylabel = "residual")
res.seasonal.plot(ax=ax3,ylabel = "seasonality")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1732: SettingWith
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
  self._setitem_single_block(indexer, value, name)
```

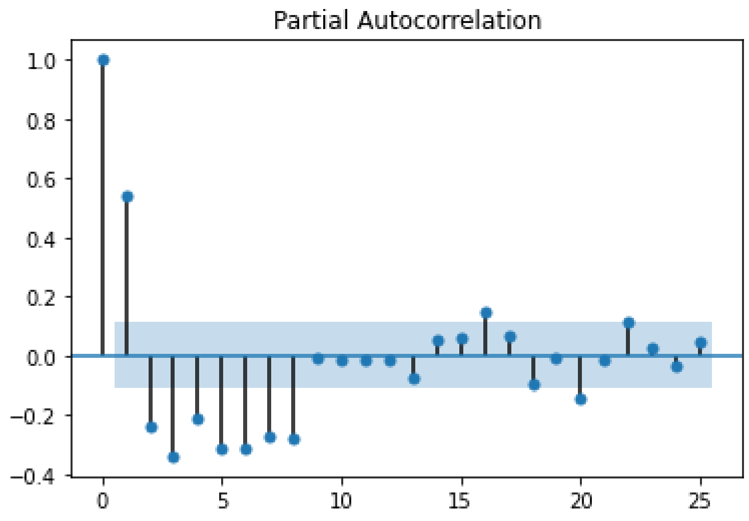


## ACF and PACF

```
df1["diff"].iloc[0] = 0
plot_acf(df1["diff"])
plt.show()
```



```
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(df1["diff"])
plt.show()
```



ARIMA MODEL ‘p’ is the order of the ‘Auto Regressive’ (AR) term. It refers to the number of lags of Y to be used as predictors. And ‘q’ is the order of the ‘Moving Average’ (MA) term. It refers to the number of lagged forecast errors that should go into the ARIMA Model.

D value

d = 1 because differencing

d = 2 overdifferencing

d = 0 series requires some differencing as p value>0.05

P Value : using AR term which is PACF I chose 2 because first lag is significantly out of the limit and the second one is also out of the significant limit but close to one whereas the third one is below so we can select the order of the p as 2.

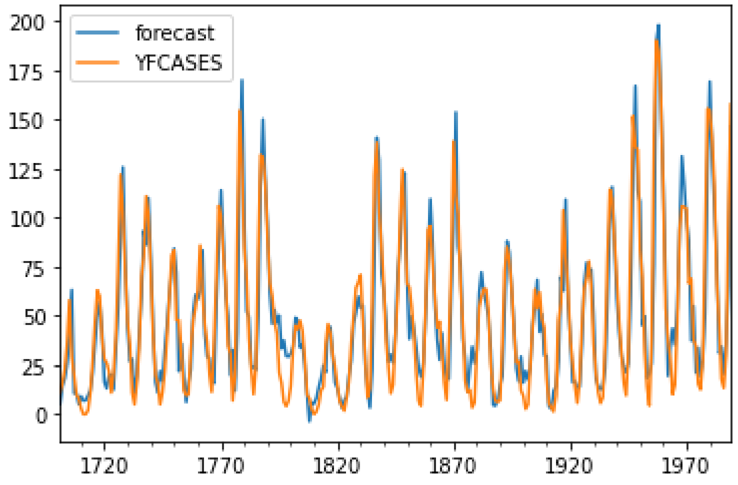
Q value : MA term for this we look at ACF plot, which will tell us how much moving average is required to remove the autocorrelation from the stationary time series. In our case we can see 4 of them being significantly outside in a group so q =4

```
from statsmodels import tsa
import sklearn.metrics
import statsmodels.api as sm
temp = df1.copy()
forecast = pd.DataFrame()


trainset = df1.loc[:'1989-12-31'][["YFCASES"]]
model = sm.tsa.ARIMA(trainset, order=(2,1,4)).fit()
predictions = model.predict(start=289, end=309)
temp["forecast"] = predictions


plt.plot(temp["diff"], label="original")
plt.plot(temp["forecast"], label="forecast")
plt.legend()


filter_df = temp.dropna()
print()
print("ERROR VALUE", mean_absolute_error(filter_df["diff"], filter_df["forecast"]))
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472: Futur
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:527: Va
  % freq, ValueWarning)

ERROR VALUE 11.382691835637733
```

```
model.plot_predict(dynamic=False)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:132: Fu
  date_key = Timestamp(key, freq=base_index.freq)
```



Compared to Moving Average the ARIMA model performs better because the mean absolute deviation is around 11.38 compared to ARMA which is 17 for 20 years forecast