On the Fly

Authors: Disha Gupta and Tathagata Munshi

Date: 01/10/2020

Context

In the age of data-driven processes and concerned strategy making, companies are extremely reliant on gathering the correct insights and generting forecasts from collected data. And when it comes to the airlines, that are solely reliant on the volumes of passengers that avail them, data modelling plays an immense role. Whether an airlines company wishes to embark on business expansion or plans on a cost-cutting process or even prepares basic business strategies, they would wish to understand how the demand for airlines can work out for the future. A common problem that requires solving in this regard is ," **What will the average number of airpassengers be at a time-point x in the future?**" Business decisions and marketing models are built successfully upon how well we answer this question. In this project,we provide our effort in answering the above question by understanding the data and fitting the best possible model to explain the data, that'll help generate quality and relevant forecasts for airpassengers in the future.

Contents

1.Importing the necessary libraries and the Airpassengers dataset

In [1]:
```
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import statsmodels.tsa.stattools as sts
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error
from math import sqrt
import warnings
warnings.filterwarnings('ignore')
```

The dataset that has been used for the project, is the AirPassengers dataset from Kaggle. The dataset provides monthly totals of a US airline passengers from the year of 1949 to 1960.

In [2]:
```
df = pd.read_csv('AirPassengers.csv')
df['Month'] = pd.to_datetime(df['Month'],infer_datetime_format=True)
df = df.set_index(['Month'])
df.head(5)
```

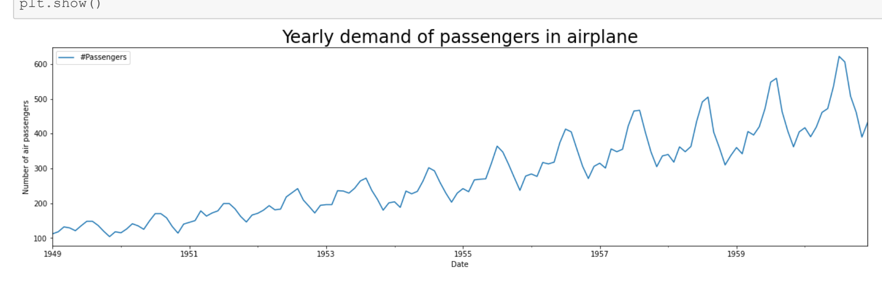
Out[2]:

Month	#Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

2.Visualizing the data

Here,we use a simple line graph to present our data visually, to get an idea of how the data pans out ina time chart. Once we have generated the line plot, we'll move onto more advanced charts to understand our data better.

In [3]:
```
df.plot(figsize=(20,5))
plt.title("Yearly demand of passengers in airplane",size=24)
plt.xlabel('Date')
plt.ylabel('Number of air passengers')
plt.show()
```



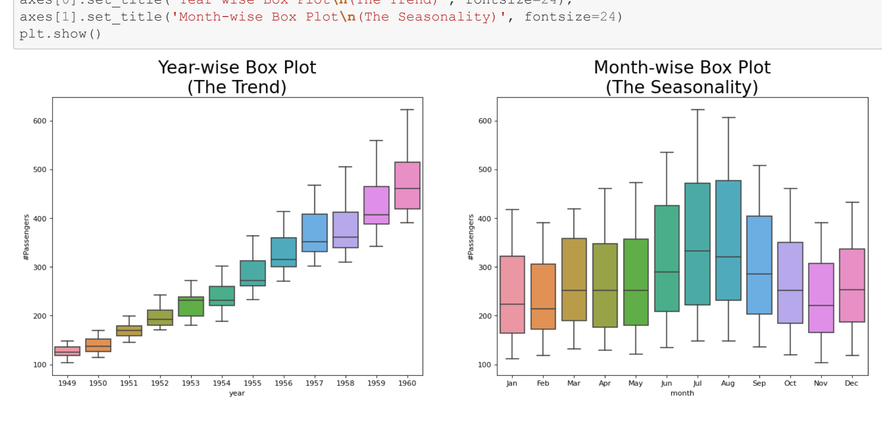
Now that we've our line plot ready, we'll move on to the Box Plots before we make an attempt to read into our dataset. Box Plots are unique tools in statistical analysis, in which numerical data can be represented via boxes and clearly demarcated using their quartile values.

In Time-Series data, aggregated box plots for months and years provide us a much more lucid outlook on the patterns of data than a simple line chart.So, we move ahead and plot our box plots:

- First, for charting out the trend.
- And then, to get a clearer idea of the seasonality.

In [4]:
```
df['year'] = [d.year for d in df.index]
df['month'] = [d.strftime('%b') for d in df.index]
years = df['year'].unique()

#plotting the data
fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi= 80)
sns.boxplot(x='year', y='#Passengers', data=df, ax=axes[0])
sns.boxplot(x='month', y='#Passengers', data=df)
axes[0].set_title('Year-wise Box Plot\n(The Trend)', fontsize=24);
axes[1].set_title('Month-wise Box Plot\n(The Seasonality)', fontsize=24)
plt.show()
```



Before making significant observations from the plotted data, we must understand the key concepts that are used to describe a time series data :

- **Trend**: A long term pattern in our data that shows a pattern, either increasing or decreasing or constant .
- **Seasonality**: Regular changes in the datapoints, that occur in a cyclical manner over periods that are usually functions of time.
- **Irregular Patterns**: Random fluctuations that affect the data and lead to sudden spikes or depressions.
- **Stationarity** :Stationarity means that the statistical properties of a process generating a time series do not change over time. It does not mean that the series does not change over time, just that the way it changes does not itself change over time.

Once we have made a note of the different components that generally are present in a time series data, let us delve a little deeper into observing our plotted data:

- It is evident from the plot that there is a presence of an overall increasing trend in the number of passengers over the year, as should be the case with the modernisation of the airlines as well as increase in the quality of life over the years.But it must be noted that the trend is not exactly a linear one, as it represents itself more like a quadratic one.
- We can also identify some seasonality from our graph, that repeats itself every 12 months or so.This seasonality may be attributable to the beginning and end of holidays, when everyone starts leaving their homes and workplacesfor vacation or vice-versa.
- Some irregular features may present itself along the way, as we see spikes in places of depressions and depressions in places of spikes at certain points, that do not comply to the overall pattern of the data.
- Lastly, the time-series data that we have here is not stationary and hence we would need to transform it into a stationary series, for ease of analysis and model fitting.

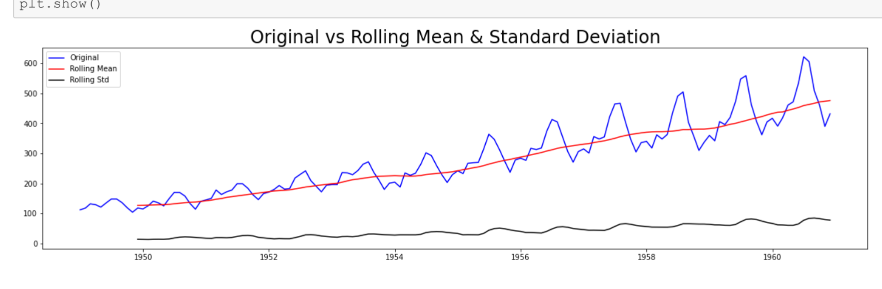
In [5]:
```
df.drop(['year','month'],axis=1,inplace=True)
```

3.Rolling Mean and Rolling SD

Let us now move on to working on the stationarity of the dataset by computing the Rolling Mean and Rolling SD along with the original data.

In [6]:
```
rolmean = df.rolling(window=12).mean() #window size 12 denotes 12 months
rolstd = df.rolling(window=12).std()

plt.figure(figsize=(20,5))
original = plt.plot(df, color='blue', label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Original vs Rolling Mean & Standard Deviation',size=24)
plt.show()
```



The graph that we have obtained corroborates with our claim : AirPassengers dataset has non-stationarity present in it. This can be identified from the fact that the rolling mean itself has a trend component even though rolling standard deviation is fairly constant with time. For our time series to be stationary, we need to ensure that both our rolling statistics: mean and standard deviation are invariant with time.Thus, the curves for both of them have to be parallel to the x-axis, which in our case is not.

To further augment our hypothesis that the time series is not stationary, let us perform the ADCF Test.

4.Augmented Dickey-Fuller Test: Investigating Stationarity.

Augmented Dickey Fuller test (ADF Test) is a common statistical test used to test whether a given Time series is stationary or not. It is one of the most commonly used statistical test when it comes to analyzing the stationary of a series.

The inference about the stationarity of the time series is decided from the p-value as well as the significance of coefficient from the results.

In [7]:
```
sts.adfuller(df)
```

Out[7]:
```
(0.8153688792060423,
 0.9918802434376409,
 13,
 130,
 {'1%': -3.4816817173418295,
  '5%': -2.8840418343195267,
  '10%': -2.578770059171598},
 996.6929308390189)
```

Note that the p-value is significantly higher even at the 10% level of significance, let alone 5%.This clearly confirms that our hypothesis regarding the stationarity of our time series data was true and the data isn't stationary at all.

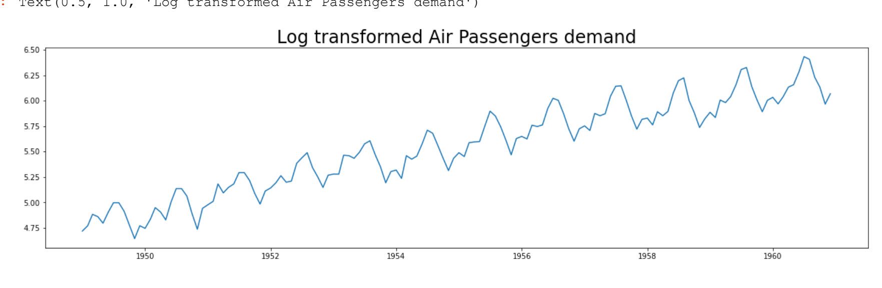
5.Transformation of Data: Logarithmic Transformation

Unless our time series is stationary, we cannot build a time series model. In cases where the stationary criteria are violated, it requires to first stationarize the time series and then try stochastic models for predictions. There are multiple ways to achieve this stationary, however concerning our series, we would use a combination of two techniques:

- We need to remove unequal variances. This can be done by taking the log of the values.
- We need to address the trend component. This can be done by taking the time-shifting of the log series.

In [8]:
```
df_log = np.log(df)
plt.figure(figsize=(20,5))
plt.plot(df_log)
plt.title('Log transformed Air Passengers demand',size=24)
```

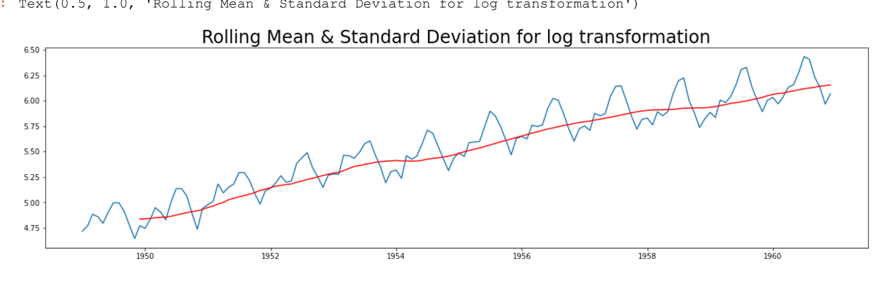
Out[8]:
```
Text(0.5, 1.0, 'Log transformed Air Passengers demand')
```



In this moving average approach, we take average of 'k' consecutive values depending on the frequency of time series. Here we can take the average over the past 1 year, i.e. last 12 values. Pandas has specific functions defined for determining rolling statistics.

In [9]:
```
movingAverage = df_log.rolling(window=12).mean()
movingSTD = df_log.rolling(window=12).std()
plt.figure(figsize=(20,5))
plt.plot(df_log)
plt.plot(movingAverage, color='red')
plt.title('Rolling Mean & Standard Deviation for log transformation',size=24)
```

Out[9]:
```
Text(0.5, 1.0, 'Rolling Mean & Standard Deviation for log transformation')
```



The red line shows the rolling mean. Lets subtract this from the original series. Note that since we are taking average of last 12 values, rolling mean is not defined for first 11 values.

In [10]:
```
datasetLogScaleMinusMovingAverage = df_log - movingAverage
datasetLogScaleMinusMovingAverage.head(12)
datasetLogScaleMinusMovingAverage.dropna(inplace=True)
datasetLogScaleMinusMovingAverage.head(10)
```

Out[10]:

Month	#Passengers
1949-12-01	-0.065494
1950-01-01	-0.093449
1950-02-01	-0.007566
1950-03-01	0.099416
1950-04-01	0.052142
1950-05-01	-0.027529
1950-06-01	0.139881
1950-07-01	0.260184
1950-08-01	0.248635
1950-09-01	0.162937

6.Testing Stationarity of the Transformed Data

Once we have finished trasnformation, the dataset requires checking using the rolling statistics and the Augmented Dickey-Fuller Test to ensure that we have remove all traces of non-stationarity from our data.

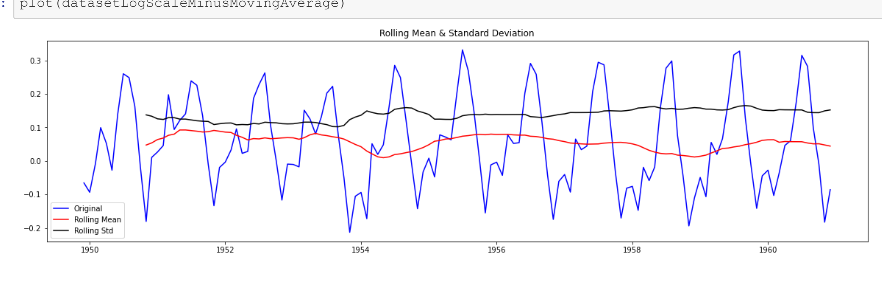
To prevent repeating the same chunks of code again and again in case we have to check statonarity of our series again, we build a function, callig which along wih our relevant time series data we'll be able to avail the rolling statistics plotted.

In [11]:
```
def plot(timeseries):

    #Determine rolling statistics
    movingAverage = timeseries.rolling(window=12).mean()
    movingSTD = timeseries.rolling(window=12).std()

    #Plot rolling statistics
    plt.figure(figsize=(20,5))
    orig = plt.plot(timeseries, color='blue', label='Original')
    mean = plt.plot(movingAverage, color='red', label='Rolling Mean')
    std = plt.plot(movingSTD, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()
```

In [12]:
```
plot(datasetLogScaleMinusMovingAverage)
```



Observing the rolling statistics from the plot , we can clearly see that we have managed to achieve stationarity of the time series based on the rolling statistics. But we need more of a statistical confirmation of our claim and we can get that from the augmented Dickey-Fuller test.

In [13]:
```
sts.adfuller(datasetLogScaleMinusMovingAverage)
```

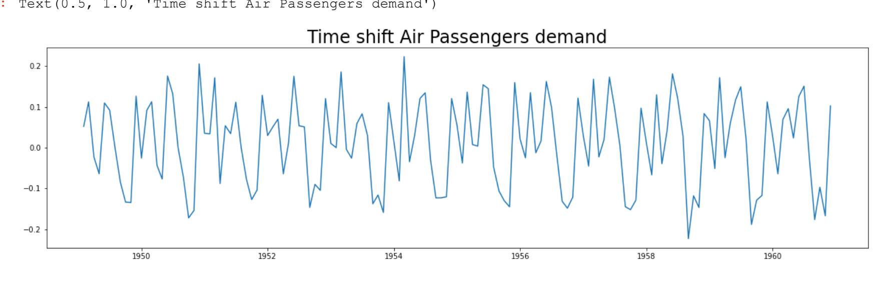
Out[13]:
```
(-3.162907991300858,
 0.022234630001243844,
 13,
 119,
 {'1%': -3.4865346059036564,
  '5%': -2.8861509858476264,
  '10%': -2.579896092790057},
 -436.63269481747125)
```

7.Transformation of Data: Time-Shift

Here, we do a time shift transformation to get the best model for prediction. So, we use the shift function to get the difference between log transformation and the shifted scale value.

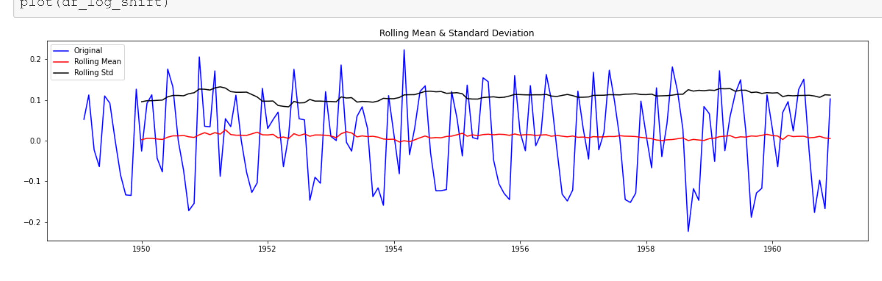
In [14]:
```
df_log_shift = df_log - df_log.shift()
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.title('Time shift Air Passengers demand',size=24)
```

Out[14]:
```
Text(0.5, 1.0, 'Time shift Air Passengers demand')
```



Finally, we plot the time shifted dataalong with rolling mean and rolling standard deviation and see both are constant.

In [15]:
```
df_log_shift.dropna(inplace=True)
plot(df_log_shift)
```



8.Seasonal Decomposition

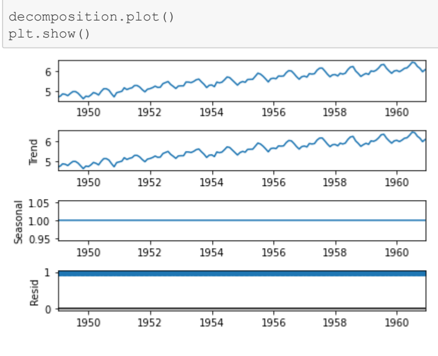
Decomposition is primarily used for time series analysis, and as an analysis tool it can be used to inform forecasting models on your problem.

It provides a structured way of thinking about a time series forecasting problem, both generally in terms of modeling complexity and specifically in terms of how to best capture each of these components in a given model.Seasonal Decomposition can be carried out by two basic model assumptions: Additive Model when there seems to be the presence of a basic linear trend and a multiplicative model, when such trends may be quadratic or exponential and have irregularities throughout the data.

Upon inspecting the line plot of our data, it suggests that there may be a linear trend, but it is hard to be sure from eye-balling. There is also seasonality, but the amplitude (height) of the cycles appears to be increasing, suggesting that it is multiplicative.

In [16]:
```
decomposition = seasonal_decompose(df_log, model='multiplicative',freq=1)

decomposition.plot()
plt.show()
```



Running the plots, we obtain a pictoral representation of the observed, trend, seasonal, and residual time series.

We can see that the trend and seasonality information extracted from the series does seem reasonable. The residuals are also interesting, showing periods of high variability in the early and later years of the series.

9. ACF and PACF plots

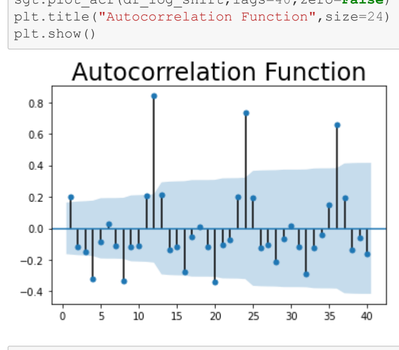
For understanding what should be the ideal no. of lags or residuals to be considered for fitting our best AR or MA values, we need significant help from the ACF and PACF plots.However before we proceed to plot our ACF and PACF plots, we need to understand what ACF and PACF means and how they are significant to our time-series data.

ACF stands for the Auto-Correlation Function, which provides us with values of auto-correlation of any series with its lag values.In simple terms, it describes how well the present value of the series is related with its past values. A time series can have components like trend, seasonality, cyclic and residual. ACF considers all these components while finding correlation between our data and the past lags.

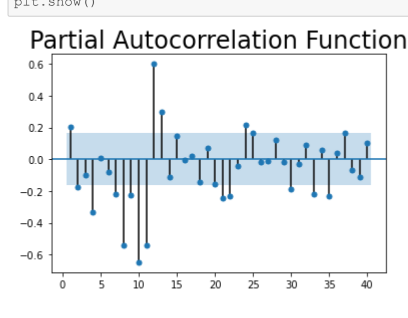
PACF stands for the Partial Auto-Correlation function.Basically instead of finding correlations of present with lags like ACF, it finds correlation of the residuals (which remains after removing the effects which are already explained by the earlier lag(s)) with the next lag value hence 'partial' and not 'complete' as we remove already found variations before we find the next correlation. So if there is any hidden information in the residual which can be modeled by the next lag, we might get a good correlation and we will keep that next lag as a feature while modeling.

The PACF plot provides us the value of the order of our Auto-Regressive(AR) Process, based on the optimum number of lags for our AR process.On the other hand, the ACF plot provides us the order of Moving Avergae(MA) Processes, depending on which time-point the correlation coefficients cross the upper confidence limit. Now that we can grasp the significance of the ACF and PACF plots, let's investigate the ACF and PACF plots for our dataset.

In [17]:
```
sgt.plot_acf(df_log_shift,lags=40,zero=False)
plt.title("Autocorrelation Function",size=24)
plt.show()
```



In [18]:
```
sgt.plot_pacf(df_log_shift,lags=40,alpha=0.05,zero=False,method=('ols'))
plt.title("Partial Autocorrelation Function",size=24)
plt.show()
```



We have our ACF and PACF plots ready. Now it's time to find out the orders of our time series processes.

- From the PACF plot , it is noticed that the first, second and fourth lags are significant, from wherein the significance of the lags greatly falter until the arrival of the seventh and the eighth lags. We could put the value of p in the AR(p) process to be p=1,2,4,7,8,.. and so on.
- From the ACF plot, we see that the first coefficient itself jumps above the upper confidence limit. We must take the value of q in our MA(q) process to be q=1, but there might arise situations where the 2nd residual still has some predicting power in deciding the best possible MA process. However more on that later, as we move on to the most interesting part of our project: **Fitting the models**.

10. AR Model

From observing the relevant lags from the PACF plot, we found out a number of possible values of p. But while checking out the model summaries, only two of the model arrived unscathed in terms of predicting power: AR(2) and AR(4). The summaries of both the models are provided below.

In [19]:
```
model_ar_2 = ARMA(df_log_shift, order=(2,0))
result_ar_2 = model_ar_2.fit()
result_ar_2.summary()
```

Out[19]:

ARMA Model Results

Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	ARMA(2, 0)	**Log Likelihood**	122.802
Method:	css-mle	**S.D. of innovations**	0.102
Date:	Fri, 02 Oct 2020	**AIC**	-237.605
Time:	10:22:05	**BIC**	-225.753
Sample:	02-01-1949	**HQIC**	-232.789
	- 12-01-1960		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0096	0.009	1.048	0.295	-0.008	0.028
ar.L1.#Passengers	0.2359	0.083	2.855	0.004	0.074	0.398
ar.L2.#Passengers	-0.1725	0.083	-2.070	0.038	-0.336	-0.009

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.6838	-2.3088j	2.4079	-0.2042
AR.2	0.6838	+2.3088j	2.4079	0.2042

In [20]:
```
model_ar_4 = ARMA(df_log_shift, order=(4,0))
result_ar_4 = model_ar_4.fit()
result_ar_4.summary()
```

Out[20]:

ARMA Model Results

Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	ARMA(4, 0)	**Log Likelihood**	131.367
Method:	css-mle	**S.D. of innovations**	0.096
Date:	Fri, 02 Oct 2020	**AIC**	-250.735
Time:	10:22:11	**BIC**	-232.958
Sample:	02-01-1949	**HQIC**	-243.511
	- 12-01-1960		

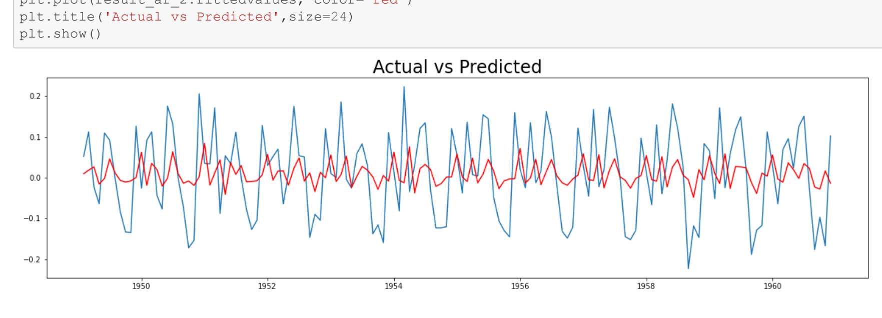
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0101	0.006	1.697	0.090	-0.002	0.022
ar.L1.#Passengers	0.1922	0.079	2.434	0.015	0.037	0.347
ar.L2.#Passengers	-0.2021	0.082	-2.466	0.014	-0.363	-0.041
ar.L3.#Passengers	-0.0265	0.081	-0.327	0.744	-0.186	0.133
ar.L4.#Passengers	-0.3290	0.080	-4.103	0.000	-0.486	-0.172

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.8314	-0.9232j	1.2423	-0.1333
AR.2	0.8314	+0.9232j	1.2423	0.1333
AR.3	-0.8717	-1.0999j	1.4034	-0.3567
AR.4	-0.8717	+1.0999j	1.4034	0.3567

By comparing both the AR(2) and AR(4) models we see AR(2) has a better predicting power and it has significant coefficients. So we plot an actual vs predicted graph for AR(2) model.

In [21]:
```
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(result_ar_2.fittedvalues, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



11. MA model

After examining the ACF of the time series we saw q=1 was more significant than others and hence we check different models and among them MA(1) and MA(4) have a better predicting power. There summaries are given below-

In [22]:
```
model_ma_1 = ARMA(df_log_shift, order=(0,1))
result_ma_1 = model_ma_1.fit()
result_ma_1.summary()
```

Out[22]:

ARMA Model Results

Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	ARMA(0, 1)	**Log Likelihood**	121.754
Method:	css-mle	**S.D. of innovations**	0.103
Date:	Fri, 02 Oct 2020	**AIC**	-237.507
Time:	10:22:27	**BIC**	-228.619
Sample:	02-01-1949	**HQIC**	-233.895
	- 12-01-1960		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0097	0.011	0.887	0.375	-0.012	0.031
ma.L1.#Passengers	0.2722	0.095	2.873	0.004	0.086	0.458

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	-3.6744	+0.0000j	3.6744	0.5000

In [23]:
```
model_ma_4 = ARMA(df_log_shift, order=(0,4))
result_ma_4 = model_ma_4.fit()
result_ma_4.summary()
```

Out[23]:

ARMA Model Results

Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	ARMA(0, 4)	**Log Likelihood**	144.693
Method:	css-mle	**S.D. of innovations**	0.087
Date:	Fri, 02 Oct 2020	**AIC**	-277.385
Time:	10:22:38	**BIC**	-259.608
Sample:	02-01-1949	**HQIC**	-270.161
	- 12-01-1960		

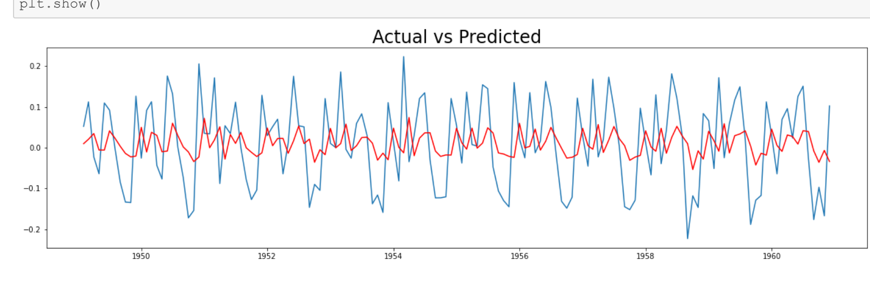
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0101	0.001	18.542	0.000	0.009	0.011
ma.L1.#Passengers	0.0417	0.081	0.512	0.609	-0.118	0.201
ma.L2.#Passengers	-0.3383	0.085	-3.988	0.000	-0.505	-0.172
ma.L3.#Passengers	-0.2381	0.074	-3.208	0.001	-0.384	-0.093
ma.L4.#Passengers	-0.4651	0.083	-5.581	0.000	-0.628	-0.302

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.0000	-0.0000j	1.0000	-0.0000
MA.2	-1.1693	-0.0000j	1.1693	-0.5000
MA.3	-0.1714	-1.3451j	1.3560	-0.2702
MA.4	-0.1714	+1.3451j	1.3560	0.2702

From the above summary table we see for MA(1) the predicting power is better as it has significant coefficients. Hence we plot the actual vs predicted plot for MA(1).

In [24]:
```
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(result_ma_1.fittedvalues, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



12. ARIMA models

We want our time series model to be parsimonious, so we tend to choose the simple models rather than the complicated ones for prediction. Hence, here we choose AR(2) and MA(1) as best predicting model. Now we have to find the best ARIMA model.

In [25]:
```
def rmsecheck(timeseries,fitted):
    list_actual = []
    list_predict = []
    list_actual = timeseries['#Passengers']
    list_predict = fitted
    mse = mean_squared_error(list_actual,list_predict)
    rmse = sqrt(mse)
    print('RMSE: %.6f' % rmse)
```

Here, we take d=0 as it is a stationary time series. After checking a multiple models we see ARMA(2,0,2) and ARMA(1,0,1) has more predicting power. The summary of these two models are given below along with there actual vs predicted plot and RMSE value.

In [26]:
```python
model_ar_2_ma_2 = ARIMA(df_log_shift, order=(2,0,2))
result_ar_2_ma_2 = model_ar_2_ma_2.fit()
result_ar_2_ma_2.summary()
```

Out[26]:

ARMA Model Results

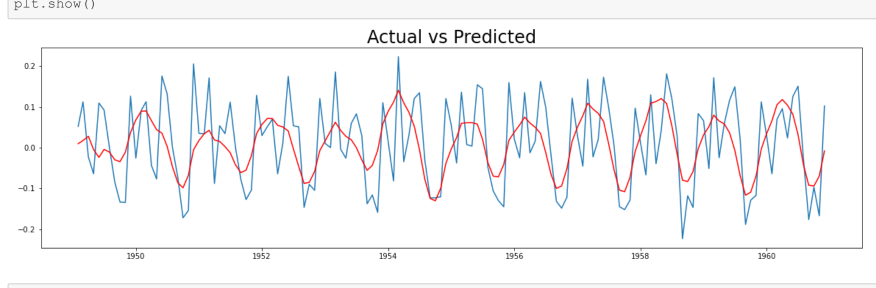
Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	ARMA(2, 2)	**Log Likelihood**	149.640
Method:	css-mle	**S.D. of innovations**	0.084
Date:	Fri, 02 Oct 2020	**AIC**	-287.281
Time:	10:22:58	**BIC**	-269.504
Sample:	02-01-1949	**HQIC**	-280.057
	- 12-01-1960		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0096	0.003	3.697	0.000	0.005	0.015
ar.L1.#Passengers	1.6293	0.039	41.868	0.000	1.553	1.706
ar.L2.#Passengers	-0.8946	0.039	-23.127	0.000	-0.970	-0.819
ma.L1.#Passengers	-1.8270	0.036	-51.303	0.000	-1.897	-1.757
ma.L2.#Passengers	0.9245	0.036	25.568	0.000	0.854	0.995

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.9106	-0.5372j	1.0573	-0.0848
AR.2	0.9106	+0.5372j	1.0573	0.0848
MA.1	0.9881	-0.3245j	1.0400	-0.0505
MA.2	0.9881	+0.3245j	1.0400	0.0505

In [27]:
```python
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(result_ar_2_ma_2.fittedvalues, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



In [28]:
```python
rmsecheck(df_log_shift,result_ar_2_ma_2.fittedvalues)
```

```
RMSE:  0.084834
```

In [29]:
```python
model_ar_1_ma_1 = ARIMA(df_log_shift, order=(1,0,1))
result_ar_1_ma_1 = model_ar_1_ma_1.fit()
result_ar_1_ma_1.summary()
```

Out[29]:

ARMA Model Results

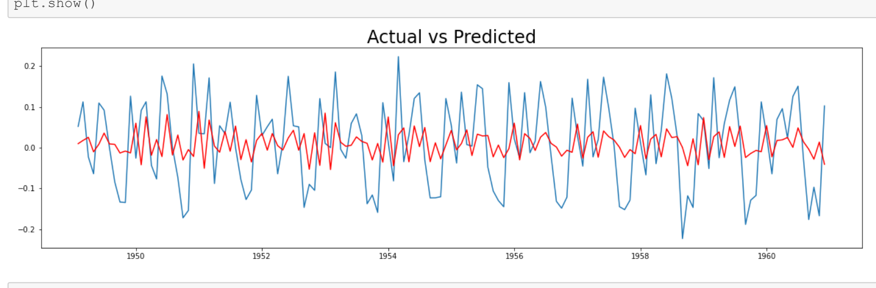
Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	ARMA(1, 1)	**Log Likelihood**	124.804
Method:	css-mle	**S.D. of innovations**	0.101
Date:	Fri, 02 Oct 2020	**AIC**	-241.608
Time:	10:23:16	**BIC**	-229.756
Sample:	02-01-1949	**HQIC**	-236.792
	- 12-01-1960		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0098	0.010	0.993	0.321	-0.010	0.029
ar.L1.#Passengers	-0.5826	0.128	-4.536	0.000	-0.834	-0.331
ma.L1.#Passengers	0.8502	0.086	9.936	0.000	0.682	1.018

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-1.7165	+0.0000j	1.7165	0.5000
MA.1	-1.1762	+0.0000j	1.1762	0.5000

In [30]:
```python
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(result_ar_1_ma_1.fittedvalues, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



In [31]:
```python
rmsecheck(df_log_shift,result_ar_1_ma_1.fittedvalues)
```

```
RMSE:  0.101034
```

Here the ARIMA(2,0,2) has a RMSE value of 0.084834 which is lower than the RMSE value of ARIMA(1,0,1) that is 0.101034 Hence, ARIMA(2,0,2) will give a better fit.

13.Auto_Arima method : SARIMAX model

From the previous discussions, we have seen that seasonality is present in the data set and hence we try to incorporate that in out best fitting model. We use the auto_arima model of statsmodels library to find a best fitting sarimax model.

In [32]:
```python
model_auto_sar = auto_arima(df_log_shift, m = 12, max_p = 7, max_q = 7, max_d = 2, max_P = 4, max_Q = 4
,max_D =2,
                             maxiter = 50, alpha = 0.05, n_jobs = -1, trend = 'ct', information_criterion =
'oob',
                            out_of_sample = int(len(df_log_shift)*0.2))
```

In [33]:
```python
model_auto_sar.summary()
```

Out[33]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	143
Model:	SARIMAX(0, 0, 1)x(2, 1, [], 12)	**Log Likelihood**	243.006
Date:	Fri, 02 Oct 2020	**AIC**	-474.013
Time:	10:25:33	**BIC**	-456.762
Sample:	0	**HQIC**	-467.003
	- 143		
Covariance Type:	opg		

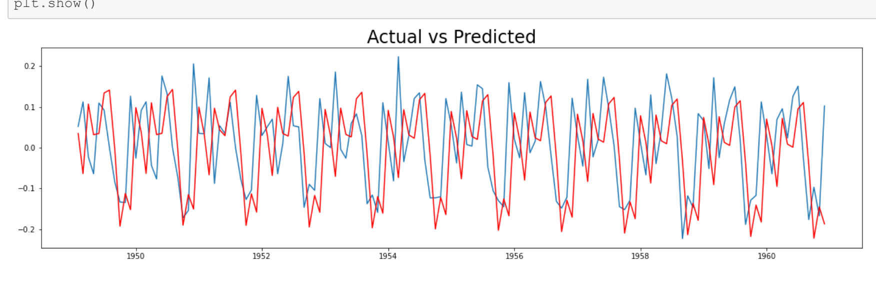
	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0036	0.005	0.674	0.500	-0.007	0.014
drift	-3.889e-05	6.58e-05	-0.591	0.555	-0.000	9.01e-05
ma.L1	-0.3526	0.077	-4.600	0.000	-0.503	-0.202
ar.S.L12	-0.4842	0.098	-4.961	0.000	-0.676	-0.293
ar.S.L24	-0.2333	0.114	-2.051	0.040	-0.456	-0.010
sigma2	0.0014	0.000	8.742	0.000	0.001	0.002

Ljung-Box (Q):	43.81	**Jarque-Bera (JB):**	1.33
Prob(Q):	0.31	**Prob(JB):**	0.51
Heteroskedasticity (H):	0.55	**Skew:**	0.02
Prob(H) (two-sided):	0.05	**Kurtosis:**	3.49

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [34]:
```python
start_date = '1949-01-01'
end_date = '1960-12-01'
df_auto_pred_sar = pd.DataFrame(model_auto_sar.predict(n_periods = len(df_log_shift[start_date:end_date
])),
                                index = df_log_shift[start_date:end_date].index )
```

In [35]:
```python
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(df_auto_pred_sar, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



The above actual vs predicted plot is not the best fit model. So we further check some SARIMAX models to enhance the predicting power.

14. Best SARIMAX model

After checking different SARIMAX models we see SARIMAX(1,0,1)(2,1,0,12) and SARIMAX(2,0,2)(2,1,0,12)has the best predicting power. The summary table, actual vs predicted plot and RMSE values are given below.

In [36]:
```python
model_sarimax_11 = SARIMAX(df_log_shift, order = (1,0,1), seasonal_order = (2,1,0,12))
results_sarimax_11 = model_sarimax_11.fit()
results_sarimax_11.summary()
```

Out[36]:

SARIMAX Results

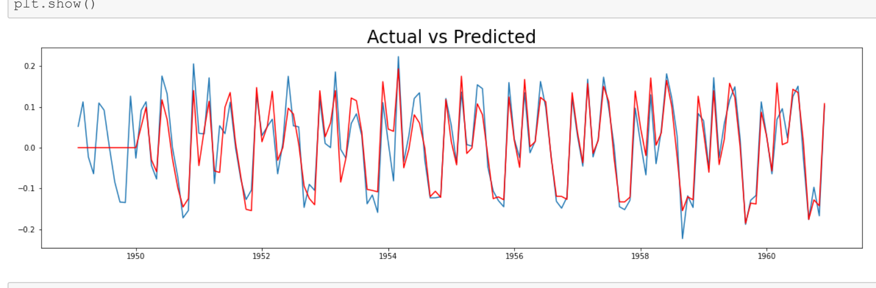
Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	SARIMAX(1, 0, 1)x(2, 1, [], 12)	**Log Likelihood**	243.829
Date:	Fri, 02 Oct 2020	**AIC**	-477.657
Time:	10:25:54	**BIC**	-463.281
Sample:	02-01-1949	**HQIC**	-471.816
	- 12-01-1960		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.1037	0.211	0.491	0.623	-0.310	0.517
ma.L1	-0.5132	0.187	-2.748	0.006	-0.879	-0.147
ar.S.L12	-0.5569	0.100	-5.556	0.000	-0.753	-0.360
ar.S.L24	-0.2035	0.113	-1.802	0.072	-0.425	0.018
sigma2	0.0014	0.000	8.429	0.000	0.001	0.002

Ljung-Box (Q):	40.25	**Jarque-Bera (JB):**	2.00
Prob(Q):	0.46	**Prob(JB):**	0.37
Heteroskedasticity (H):	0.60	**Skew:**	0.03
Prob(H) (two-sided):	0.09	**Kurtosis:**	3.60

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [37]:
```python
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(results_sarimax_11.fittedvalues, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



In [38]:
```python
rmsecheck(df_log_shift,results_sarimax_11.fittedvalues)
```

```
RMSE:  0.044768
```

In [39]:
```python
model_sarimax_22 = SARIMAX(df_log_shift, order = (2,0,2), seasonal_order = (2,1,0,12))
results_sarimax_22 = model_sarimax_22.fit()
results_sarimax_22.summary()
```

Out[39]:

SARIMAX Results

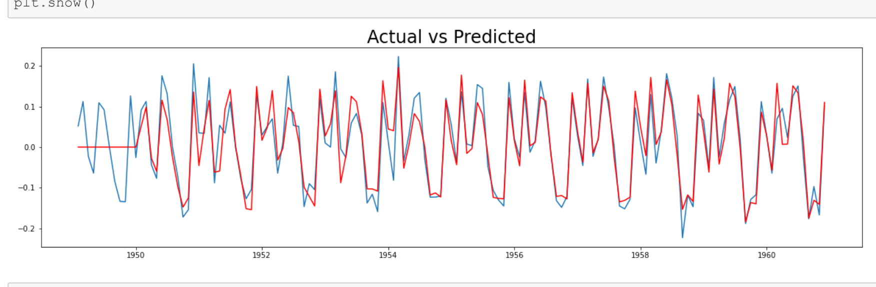
Dep. Variable:	#Passengers	**No. Observations:**	143
Model:	SARIMAX(2, 0, 2)x(2, 1, [], 12)	**Log Likelihood**	244.289
Date:	Fri, 02 Oct 2020	**AIC**	-474.577
Time:	10:26:17	**BIC**	-454.451
Sample:	02-01-1949	**HQIC**	-466.399
	- 12-01-1960		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0330	1.347	-0.025	0.980	-2.673	2.607
ar.L2	0.1719	0.264	0.652	0.514	-0.345	0.689
ma.L1	-0.3836	1.371	-0.280	0.780	-3.072	2.304
ma.L2	-0.1734	0.715	-0.242	0.808	-1.575	1.229
ar.S.L12	-0.5491	0.107	-5.112	0.000	-0.760	-0.339
ar.S.L24	-0.1983	0.115	-1.722	0.085	-0.424	0.027
sigma2	0.0014	0.000	8.017	0.000	0.001	0.002

Ljung-Box (Q):	43.13	**Jarque-Bera (JB):**	2.13
Prob(Q):	0.34	**Prob(JB):**	0.35
Heteroskedasticity (H):	0.60	**Skew:**	0.03
Prob(H) (two-sided):	0.10	**Kurtosis:**	3.62

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [40]:
```python
plt.figure(figsize=(20,5))
plt.plot(df_log_shift)
plt.plot(results_sarimax_22.fittedvalues, color='red')
plt.title('Actual vs Predicted',size=24)
plt.show()
```



In [41]:
```python
rmsecheck(df_log_shift,results_sarimax_22.fittedvalues)
```

```
RMSE:  0.044681
```

From the above comparative analysis of two SARIMAX plots: SARIMAX(1,0,1)x(2,1,0,12) and the SARIMAX(2,0,2)x(2,1,0,12) we observe a few crucial points:

- The Predicted vs Actual plots for both the SARIMAX models display similar and quite accurate fits.
- The Log-Likehood Function is greater for the SARIMAX(2,0,2)x(2,1,0,12) than the SARIMAX(2,0,2)x(2,1,0,12), and the AIC AND BIC parameters are lower for the former model hinting at greater predicting power for the SARIMAX(2,0,2)x(2,1,0,12)
- However inspite of extremely close Root Mean Square Errors, the SARIMAX(2,0,2)x(2,1,0,12) has 3 insignificant coefficients even at the 10% level of significance and 4 insiginificant coefficients at the 5% level of significance. However, the SARIMAX(1,0,1)x(2,1,0,12) has all coefficients significant at the 10% level of significance. This hints at the notion that the simpler SARIMAX Model might actually suffice for our time-series data rather than the more complex model.

Thus , the SARIMAX model that provides the best fit to our data is the **SARIMAX(1,0,1)x(2,1,0,12)**.

RMSE or the Root Mean Square Error is often used as the best measure of accuracy for fitted models or value estimators. For our SARIMA model, the RMSE was found out to be around 4.47%. This means that the measured deviation of our fitted values have just 4.47% variability from the original values of the air passengers. This corroborates with the observations from the plot, that the SARIMA Model chosen was an excellent fit to the available data and explains the patterns in the data quite successfully.

15. Taking our Forecast back to the original scale

Since the combined model gave best result, lets scale it back to the original values and see how well it performs there. Firstly we store the predicted results as a separate series and observe it.

In [42]:
```python
predictions_sarima_diff = pd.Series(results_sarimax_11.fittedvalues, copy=True)
print(predictions_sarima_diff.head())
```

```
Month
1949-02-01    0.000000e+00
1949-03-01   -3.721786e-11
1949-04-01   -8.382050e-11
1949-05-01    7.700925e-12
1949-06-01    4.645796e-11
dtype: float64
```

Notice that these start from '1949-02-01' and not the first month.This is because we took a lag by 1 and first element doesn't have anything before it to subtract from. The way to convert the differencing to log scale is to add these differences consecutively to the base number. An easy way to do it is to first determine the cumulative sum at index and then add it to the base number.So, we perform the cumulative sum.

In [43]:
```python
predictions_sarima_diff_cumsum = predictions_sarima_diff.cumsum()
print(predictions_sarima_diff_cumsum.head())
```

```
Month
1949-02-01    0.000000e+00
1949-03-01   -3.721786e-11
1949-04-01   -1.210384e-10
1949-05-01   -1.133374e-10
1949-06-01   -6.687947e-11
dtype: float64
```

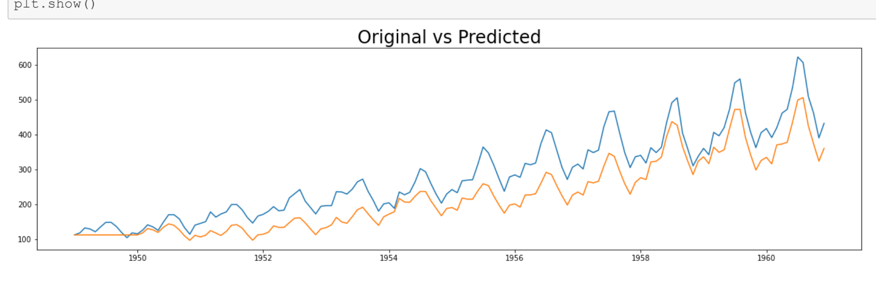
Next we've to add them to base number. For this lets create a series with all values as base number and add the differences to it.

In [44]:
```python
predictions_sarima_log = pd.Series(df_log['#Passengers'][0], index=df_log.index)
predictions_sarima_log = predictions_sarima_log.add(predictions_sarima_diff_cumsum,fill_value=0)
predictions_sarima_log.head()
```

Out[44]:
```
Month
1949-01-01    4.718499
1949-02-01    4.718499
1949-03-01    4.718499
1949-04-01    4.718499
1949-05-01    4.718499
dtype: float64
```

Here the first element is base number itself and from thereon the values cumulatively added. Last step is to take the exponent and compare with the original series.Thus we plot an actual vs predicted to compare the it.

In [45]:
```python
predictions_ARIMA = np.exp(predictions_sarima_log)
plt.figure(figsize=(20,5))
plt.plot(df)
plt.plot(predictions_ARIMA)
plt.title("Original vs Predicted",size=24)
plt.show()
```



16.Conclusion

The model can be further improved upon using popular methods such as the outlier analysis to remove or replace such data that are extremely deviant from the rest of the data.

This project was a unique experience at getting a first-hand experience of how the different models of Time Series Analysis are applied to real-life data and how the best model explains the data successfully.Thankyou for going through our project and have a nice day!