Headline: "Analyzing Air Passenger Trends: A Deep Dive into Historical Patterns and Future Projections"

Description: This comprehensive time series analysis delves into the extensive historical dataset of air passenger numbers, uncovering underlying trends, seasonality, and anomalies. The study explores the cyclic nature of air travel demand, examining the impact of various factors on passenger volume over time. Leveraging advanced statistical methods and predictive modeling, the analysis aims to forecast future passenger trends, aiding in strategic decision-making for airlines. This in-depth investigation can provide insights crucial for adapting to market fluctuations, optimizing resources, and preparing for the evolving landscape of air transportation.

A time series is a sequence of data points or observations collected and recorded at specific, equally spaced time intervals. These intervals can be regular (e.g., hourly, daily, monthly) or irregular (e.g., timestamps based on events). Time series data is used to track changes or variations in a particular phenomenon over time.

Key Components of Time Series:

1. **Temporal Order:** The data is recorded in chronological order, where the sequence of observations is crucial. Each data point is associated with a specific time or timestamp.
2. **Trend:** Long-term movement or direction in the data, whether it's an upward or downward pattern over an extended period.
3. **Seasonality:** Regular patterns or fluctuations that occur at specific intervals within a time frame. For instance, sales might rise during the holiday season or dip during off-peak times.
4. **Cyclic Patterns:** Repeating up-and-down movements that aren't fixed to a specific frequency, unlike seasonality. These cycles might occur over long periods, like economic cycles.
5. **Irregular or Random Variations:** Unpredictable fluctuations or noise within the data that doesn't follow a particular pattern. This could be due to unexpected events, random occurrences, or measurement errors.

Characteristics and Applications of Time Series:

- **Forecasting:** Time series analysis is commonly used for making predictions about future trends based on historical patterns.
- **Monitoring and Control:** It's utilized in monitoring various processes, such as stock prices, weather patterns, economic indicators, and more. For instance, control systems might use time series to detect deviations from expected behavior.
- **Statistical Analysis:** Time series analysis involves statistical methods to understand the underlying structure, dependencies, and patterns within the data.
- **Machine Learning and Predictive Modeling:** Time series data is used in machine learning models to make predictions or classifications based on historical trends and patterns.

Time Series Analysis Methods:

1. **Descriptive Statistics:** Calculating mean, median, variance, and other statistical measures to understand the properties of the data.
2. **Visualization:** Creating plots and charts such as line graphs, scatter plots, and histograms to explore trends and patterns visually.
3. **Modeling Techniques:** Implementing models like ARIMA (AutoRegressive Integrated Moving Average) or SARIMA (Seasonality ARIMA)
4. **Forecasting:** Making predictions about future values based on historical observations and model outputs.

Time series analysis plays a crucial role in understanding, interpreting, and predicting trends in various fields, including finance, economics, weather forecasting, retail sales, and more. It's fundamental for making informed decisions and planning based on historical data patterns.

Step 1: Import libraries

In [1]:
```
#import basic libraries first for data collection, cleaning, processing, visualizing and
#later stage will import time-series related libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Data Collection

In [2]:
```
data = pd.read_csv("AirPassengers.csv")
data.head()
```

Out[2]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

Step 3: Understand the data and prepare it for the process

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Month        144 non-null    object
 1   #Passengers  144 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 2.4+ KB
```

In [4]:
```
#lets convert month dtype from object to datetime using pandas
data['Month'] = pd.to_datetime(data['Month'])
```

In [5]:
```
data.info()
```

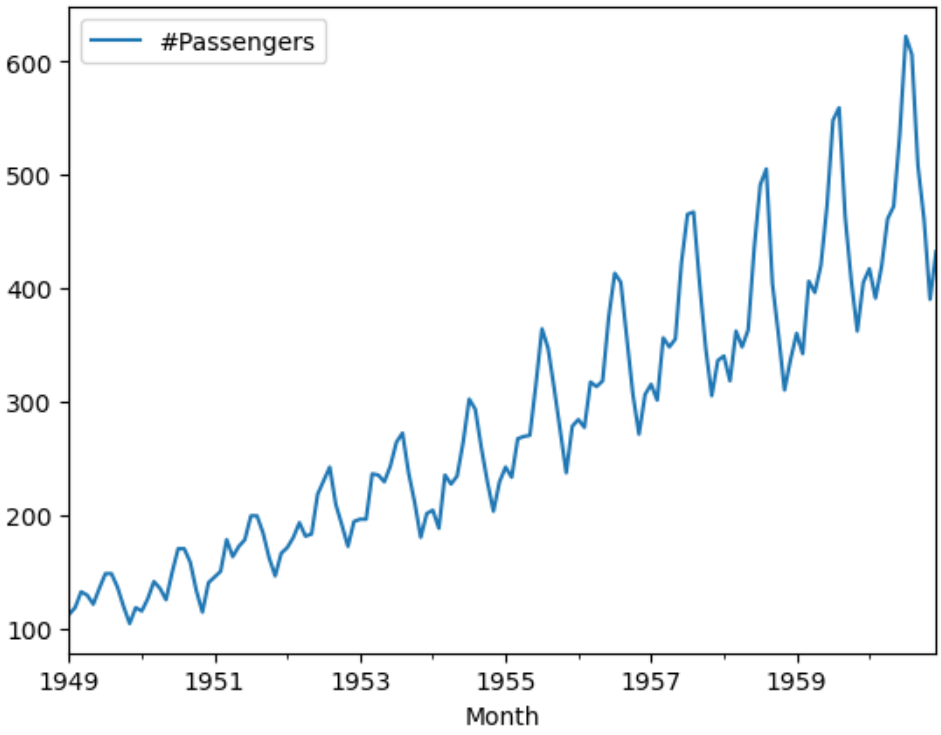
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   Month        144 non-null    datetime64[ns]
 1   #Passengers  144 non-null    int64         
dtypes: datetime64[ns](1), int64(1)
memory usage: 2.4 KB
```

In [6]:
```
#lets set the month column as index
data = data.set_index(['Month'])
data.head()
```

Out[6]:

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

In [7]:
```
#lets check the trend of our data wrt month(which is now our index)
data.plot()
plt.show()
```



In [8]:
```
#lets check whether the data is stationary or not
#for this will first import adfuller from statsmodels

from statsmodels.tsa.stattools import adfuller

#The adfuller function is part of the statsmodels library in Python and is used to conduct t
#which is a hypothesis test to determine whether a unit root is present in a time series dat
#a time series is non-stationary.

# This code snippet imports the necessary modules, applies the adfuller function to your tim
#and prints out the ADF statistic, p-value, and critical values. It then provides a basic in
#data is stationary or non-stationary based on the p-value (with a common significance leve
```

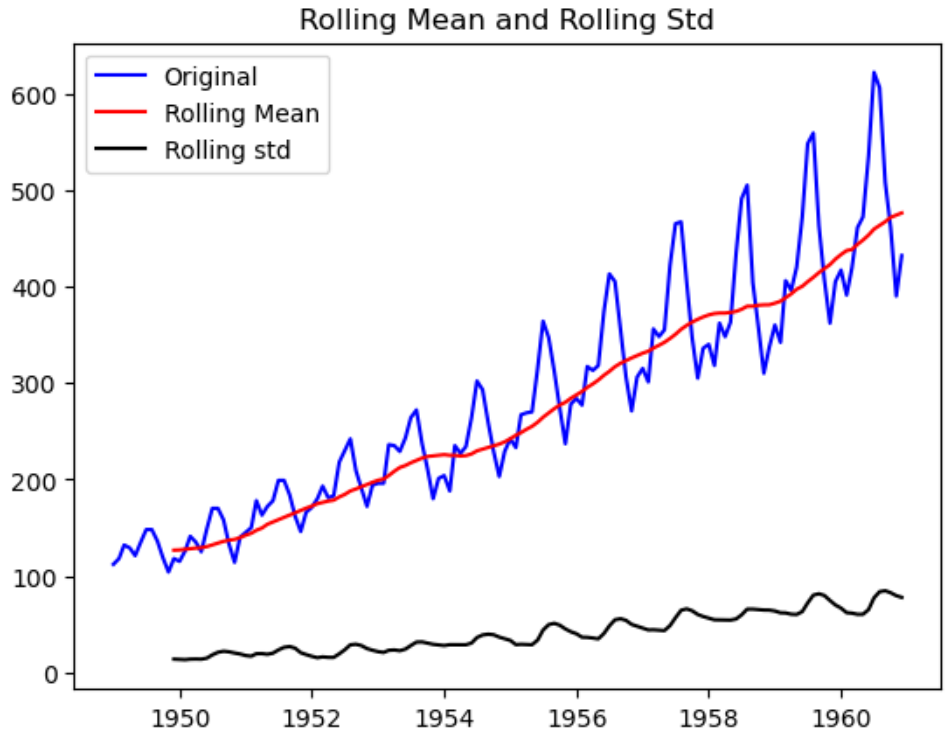
In [9]:
```
#ADF Test - if the p-value < 0.05 - Data is stationary

result = adfuller(data) #gives 3 results ADF statistic, p-value, and critical values

#lets print in the below format for better understanding

print(f'ADF Statistic,{result[0]}') #some default value based upon the critial values
print(f'p-value={result[1]}') #p-value
print(f'n_lags,{result[2]}') #previous no.of observations used for prediction


if(result[1]>0.05):
    print("The series is not stationary")
else:
    print("series is stationary")
```

```
ADF Statistic,0.8153688792060543
p-value=0.9918802434376411
n_lags,13
The series is not stationary
```

#to make the data stationary one can perform -differencing -log transforamtion -square transformation

In [10]:
```
#The rolling mean is a simple yet powerful tool in time series analysis. It provides an enh
#assists in decision-making regarding data transformations or model selection, and helps to
#the time series data.

#Rolling stats
mean_log=data.rolling(window=12).mean()
std_log=data.rolling(window=12).std()
```

In [11]:
```
#Lets plot the rolling mean and rolling std
plt.plot(data,color='blue',label='Original')
plt.plot(mean_log,color='red',label='Rolling Mean')
plt.plot(std_log,color='black',label='Rolling std')

plt.legend(loc='best')
plt.title('Rolling Mean and Rolling Std')
plt.show()
```



Plotting the rolling mean serves several significant purposes in time series analysis:

1. Visualizing Trends:

The rolling mean, also known as the moving average, helps smooth out short-term fluctuations or noise in the data, making underlying trends more apparent. By calculating the mean of a subset of data points within a moving window and plotting it against time, you can better identify the general direction of the series.

2. Detecting Trends and Seasonality:

It assists in identifying trends and seasonality that might not be immediately evident in the original time series plot. For instance, if the rolling mean exhibits a consistent increase or decrease over time, it suggests the presence of a trend.

3. Identifying Stationarity:

Stationarity is a critical assumption in time series analysis. The rolling mean can visually indicate whether the mean of the series is constant over time. If the rolling mean appears relatively constant, it suggests that the data might be stationary or exhibit stationarity after a certain transformation.

4. Spotting Anomalies:

Sudden spikes or drops in the rolling mean might signify irregular events or outliers within the time series. This can be crucial in understanding unusual occurrences within the data.

5. Model Selection and Validation:

When fitting time series models, the rolling mean helps in understanding the series' behavior. It's useful in selecting appropriate models for forecasting and also serves as a validation tool to check model accuracy and performance.

6. Adjusting Volatility and Noise:

It's instrumental in managing volatility and noise within the time series data, particularly when the original data is noisy or has a lot of short-term variability. Smoothing the data using rolling mean aids in better understanding the long-term behavior.

7. Assessing Stability:

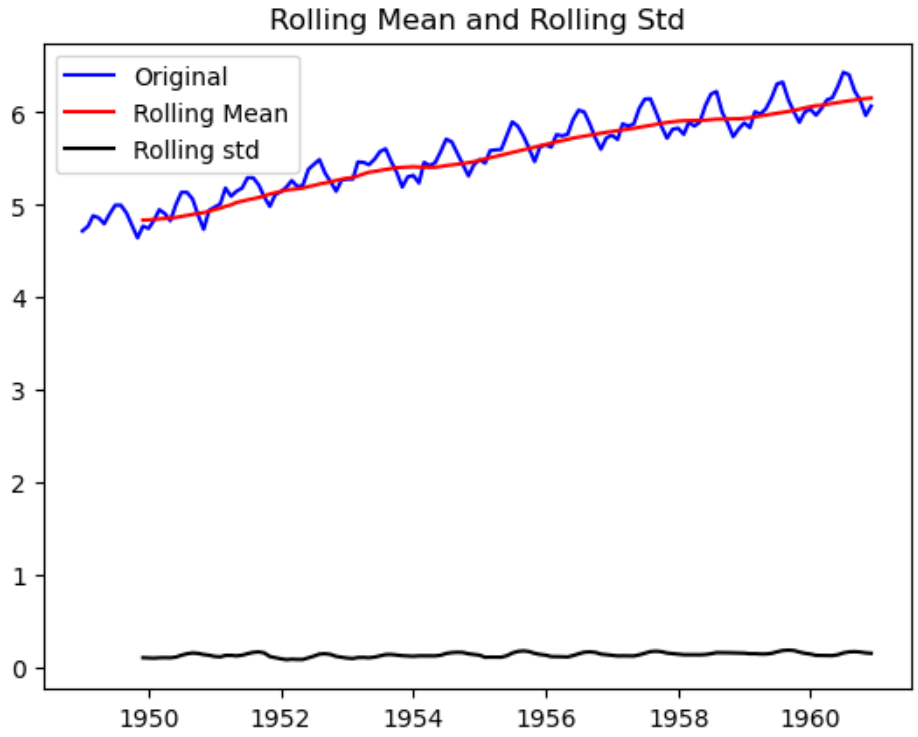
In some cases, examining the rolling mean over different time spans or comparing it with the standard deviation can help assess the stability of the series. It provides insights into whether the data is exhibiting consistent behavior over different periods.

In [12]:
```
#Transformations
first_log=np.log(data)
```

In [13]:
```
first_log = first_log.dropna()

# However, the np.log() function may produce NaN values if any of the original data points c
# By calling dropna() on first_log, any resulting NaN values are removed from the dataset,
# ensuring that only valid data remains for subsequent analysis.
```

In [14]:
```
#Lets plot again
#Rolling stats
mean_log=first_log.rolling(window=12).mean()
std_log=first_log.rolling(window=12).std()
```

In [15]:
```
#Lets plot the rolling mean and rolling std
plt.plot(first_log,color='blue',label='Original')
plt.plot(mean_log,color='red',label='Rolling Mean')
plt.plot(std_log,color='black',label='Rolling std')

plt.legend(loc='best')
plt.title('Rolling Mean and Rolling Std')
plt.show()
```



The objective of these operations seems to be creating a new dataset (new_data) by subtracting the mean value of the log-transformed dataset from the original log-transformed dataset and then removing any resulting missing values in the new dataset. This process might be part of data normalization, anomaly detection, or feature engineering in a time series analysis context.

In [16]:
```
new_data = first_log - mean_log
new_data = new_data.dropna()
new_data.head()
```

Out[16]:

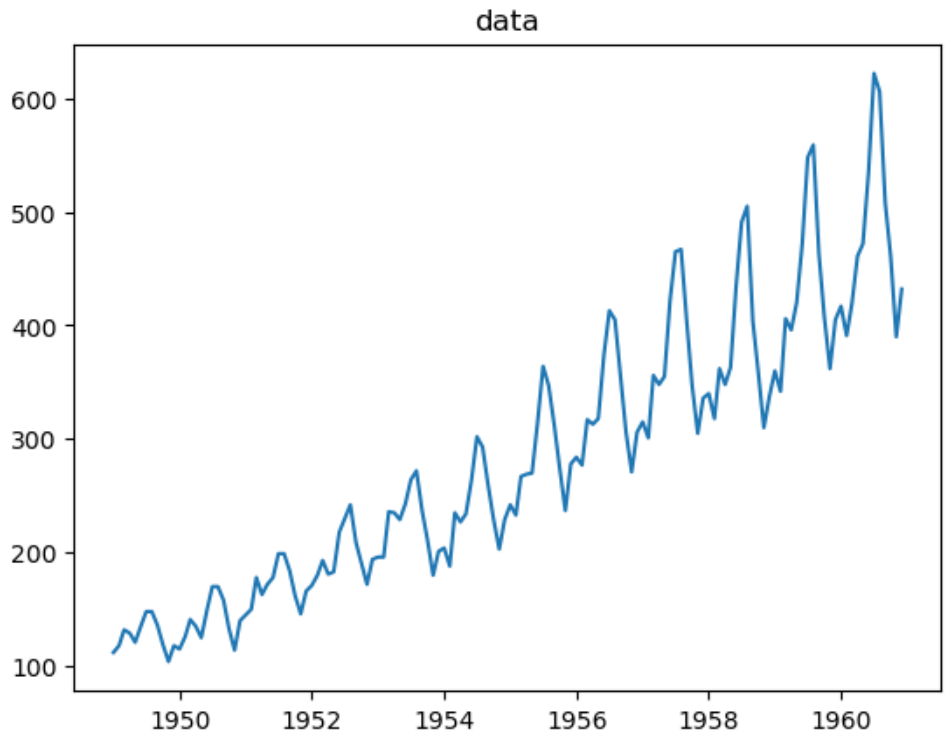
Month	#Passengers
1949-12-01	-0.065494
1950-01-01	-0.093449
1950-02-01	-0.007566
1950-03-01	0.099416
1950-04-01	0.052142

In [17]:
```
#as the transformation is completed lets check if the data is stationary or not now

#ADF Test - if the p-value < 0.05 - Data is stationary
result = adfuller(new_data)

print(f'ADF Statistic,{result[0]}') #some default value based upon the critial values
print(f'p-value={result[1]}')
print(f'n_lags,{result[2]}') #previous no.of observations used for prediction

#print(f'Result: The series is {"not " if result[1]>0.05 else ""}stationary')

if(result[1]>0.05):
    print("The series is not stationary")
else:
    print("series is stationary")
```

```
ADF Statistic,-3.1629079913008784
p-value=0.022234630001242536
n_lags,13
series is stationary
```

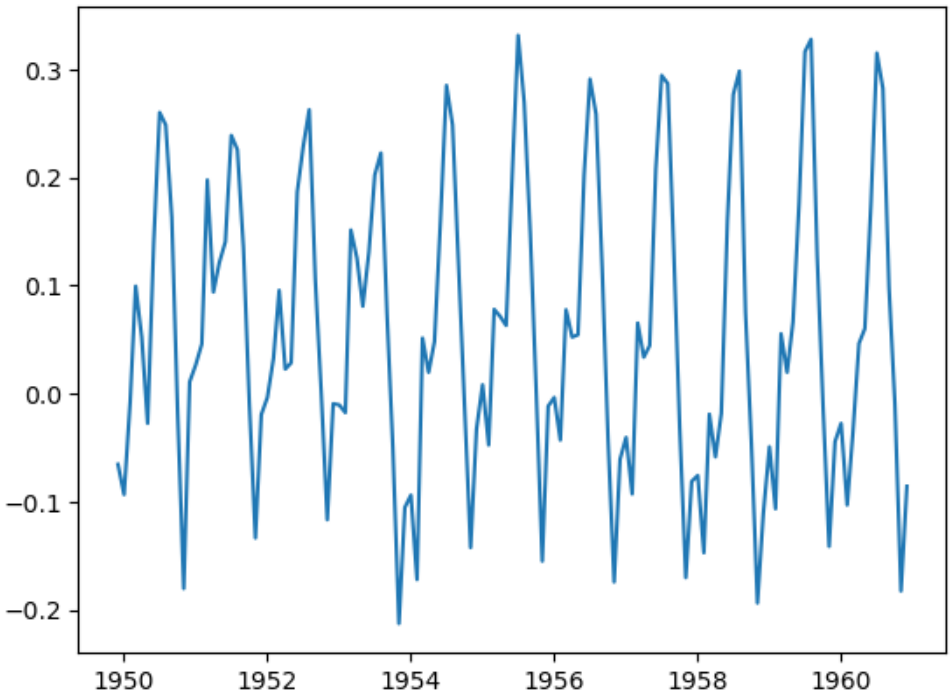
In [18]:
```
#old data which was non stationary
plt.plot(data)
plt.title('data')
```

Out[18]:
```
Text(0.5, 1.0, 'data')
```



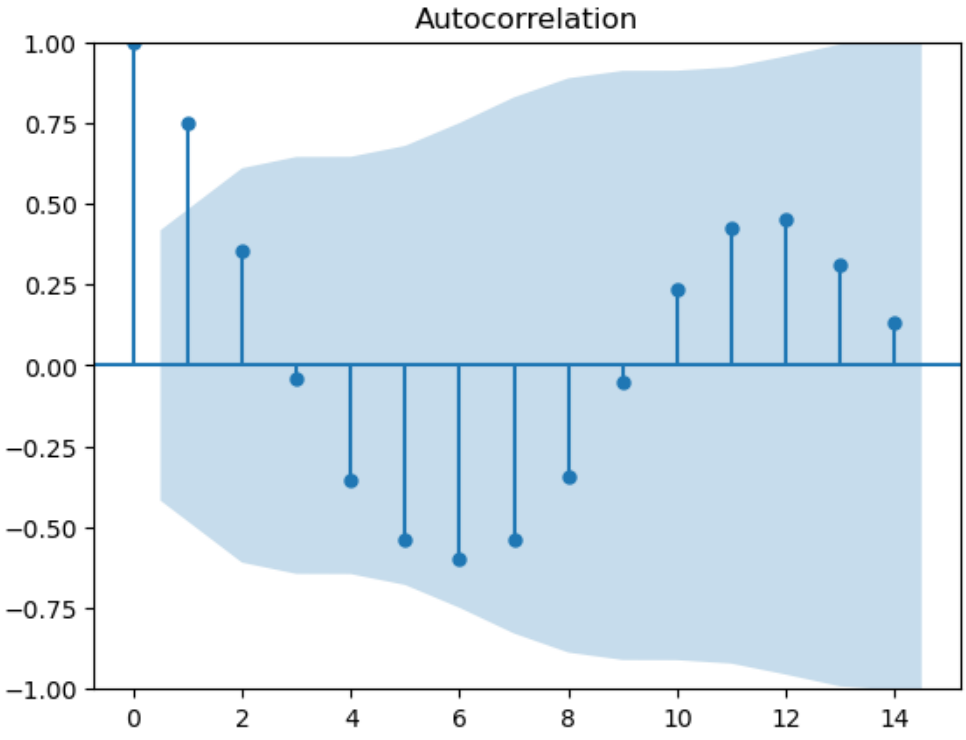
In [19]:
```
#new data which is stationary and after the transforamtions
plt.plot(new_data)
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x16a69859ea0>]
```



In [20]:
```
#Model Building
#ARIMA- auto regressive integrated moving average

from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf
```

In [21]:
```
acf_plot=acf(new_data.dropna()) #the data point or lag where there is a sudden shut-off -q=
plot_acf(acf_plot);
#use to calculate q value
```



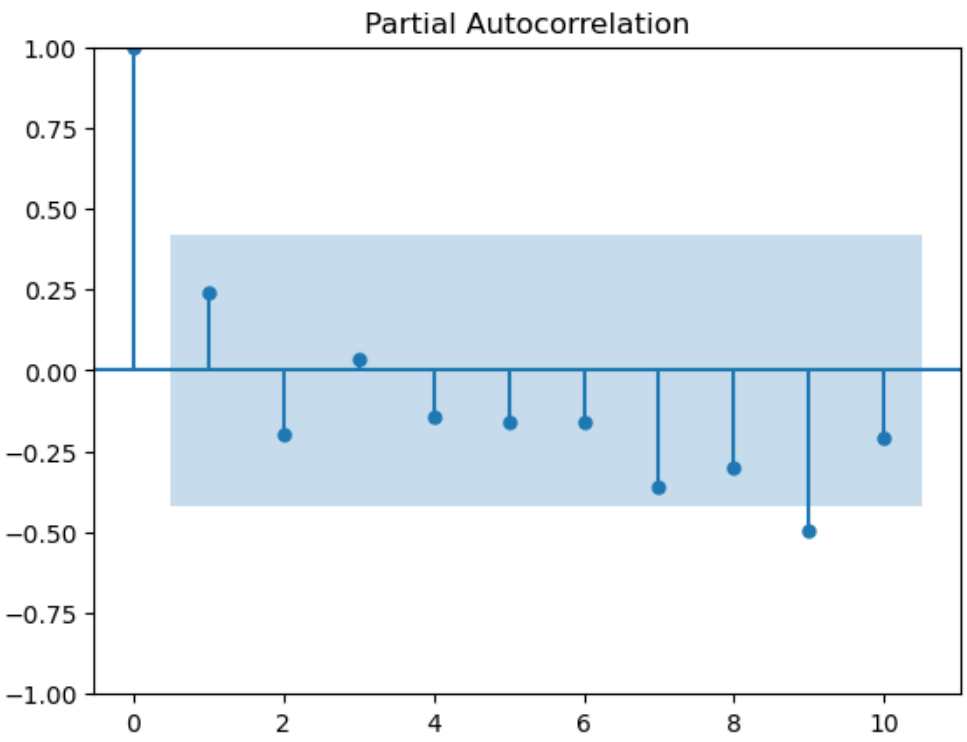
The purpose of plotting the Autocorrelation Function (ACF) is to visualize the correlation between a time series and its lagged values. Each point on the ACF plot represents the correlation between the time series at time t and the series at a lag of 'k' time periods before (k is the x-axis). It's useful for understanding the patterns of correlation, periodicity, and potential seasonality within the dataset.

Interpretation of ACF plot: Decay of Correlation: If the ACF plot shows a gradual decrease or exponential decay as lag increases, it indicates a time series that can be more easily modeled and forecasted. Significance at Lag Points: Significant spikes or patterns that deviate from the expected range (often indicated by dashed lines on the plot) can suggest potential seasonality or patterns in the data. Identification of Lag Order for Models: The ACF plot assists in determining the order for autoregressive (AR) terms in ARIMA models or other time series modeling. The ACF plot helps analysts identify the correlation structure within the time series data, providing insights into the potential lagged relationships between observations. This information is crucial for selecting appropriate models and understanding the underlying behavior of the time series.

In [22]:
```
from statsmodels.tsa.stattools import pacf
from statsmodels.graphics.tsaplots import plot_pacf
```

In [23]:
```
pacf_plot=pacf(new_data.dropna()) #Gradual decrease and previuous point to that -p= 1
plot_pacf(pacf_plot, lags=10);

#to calculate p value
```

```
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348: FutureWa
rning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. Aft
er 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can use this method
now by setting method='ywm'.
  warnings.warn(
```



The Partial Autocorrelation Function (PACF) plot displays the partial correlation of a time series with its own lagged values while controlling for the effect of previous lags. This plot is helpful for identifying the direct relationship between specific lags without the influence of intermediary lags.

Sudden Shut-Off: In the PACF plot, a significant, sudden "shut-off" after a certain lag indicates the direct effect of that lag on the current observation, with the effect tailing off rapidly for subsequent lags.

Identification of Lag Order for Models: The point where the PACF plot crosses the significance threshold (often indicated by dashed lines) can be crucial for determining the order of the autoregressive ((AR) term in models like ARIMA.

The PACF plot aids in understanding the direct impact of specific lagged values on the current observation. It provides insights into the optimal lag order for autoregressive terms in time series modeling, contributing to the selection and fine-tuning of appropriate models for the dataset.

In [24]:
```
train=new_data.iloc[:120]['#Passengers']
test=new_data.iloc[120:]['#Passengers']
```

In [25]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [26]:
```
model = ARIMA(train,order=(1,1,2)) #(p,d,q)
model_fit=model.fit()
```

```
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarn
ing: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

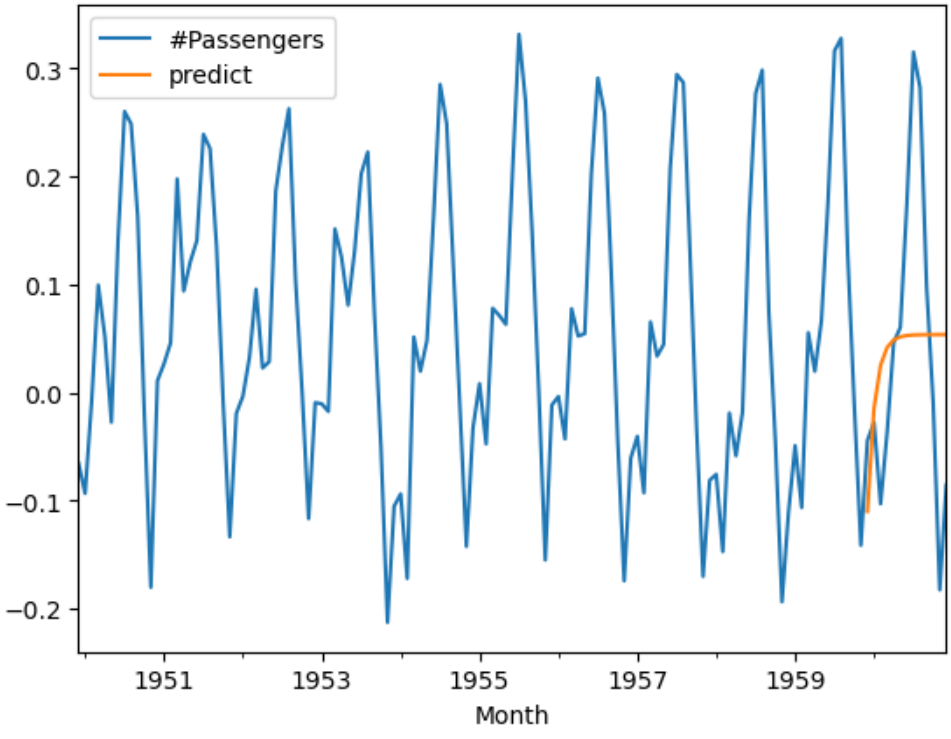
p: Represents the order of the AutoRegressive (AR) term. It's the number of lag observations included in the model.

d: Represents the degree of differencing, or the number of times the data has been differenced to achieve stationarity.

q: Denotes the order of the Moving Average (MA) term, which is the number of lagged forecast errors included in the model.

In [27]:
```
new_data['predict']=model_fit.predict(start=len(train),end=len(train)+len(test)-1, dynamic=
new_data[['#Passengers','predict']].plot()
```

Out[27]: <Axes: xlabel='Month'>



#as we have seasonality data, SARIMA will fit better check below

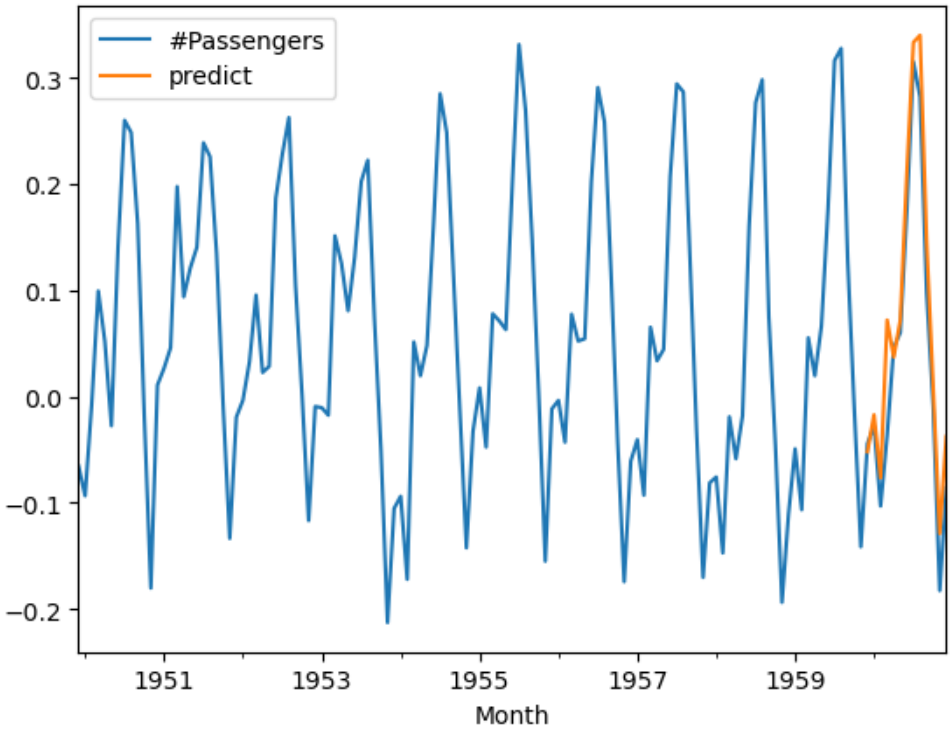
In [28]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [29]:
```
model=SARIMAX(train,order=(1,1,2),seasonal_order=(1,1,2,12))#p,d,q, time frame in which dat
model=model.fit()
```

```
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:1009: Us
erWarning: Non-invertible starting seasonal moving average Using zeros as starting paramet
ers.
  warn('Non-invertible starting seasonal moving average'
C:\Users\SIMRAN\anaconda3\lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarn
ing: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [30]:
```
new_data['predict']=model.predict(start=len(train), end=len(train)+len(test)-1,dynamic=True
new_data[['#Passengers','predict']].plot()
```

Out[30]: <Axes: xlabel='Month'>



In [ ]: