1. **Introduction**  
   The code forecasts a univariate time series using the ARIMA model. It follows these main steps:
   * Prepares and analyzes historical time series data.
   * Grid search of optimal ARIMA parameters
   * Estimate and evaluate an ARIMA model.
   * Predicts future values and displays the results.
   * It performs residual diagnostics on the model.
2. **Exploratory Data Analysis (EDA)**  
   Key Steps:  
   Import necessary libraries:

We make use of libraries like pandas, numpy, matplotlib, and statsmodels to manipulate, visualize and model the data.

Step 1: Load and preprocess the dataset  
The dataset is read with

pandas.read\_csv()

 and renamed for clarity.  
The time column is taken as a DatetimeIndex, with fixed daily frequency.

data.index = pd.date\_range(start='2024-01-01', periods=len(data), freq='D')

First differencing:

Stationarity is a key requirement of the whole basis of ARIMA, and differencing helps in achieving that by removing the trends.  
Preparing differenced Data (

diff\_data

) for ACF/PACF analysis.

diff\_data = data.diff().dropna()

Plotting ACF and PACF:

* + ACF finds correlation of a time series with its lagged versions (used for estimating q).
  + PACF isolates individual lagged effects (which we use to estimate p).

plot\_acf(diff\_data, lags=20)

plot\_pacf(diff\_data, lags=20)

1. **Data Preparation**  
   Key Steps:  
   Search through various combinations of the entire p, d, and q parameters in ARIMA:  
   The code first defines a function

grid\_search\_arima()

 to try out possible values of p, d, and q, fitting models and comparing their AIC values (lower is better).

for p, d, q in itertools.product(p\_values, d\_values, q\_values)

In this case, when you set your order parameters, you would do something like this:

results = model.fit(disp=False)

Best\_order is the optimal parameters selected per AIC.

Parameter ranges:

**python**

**Copy code**

p\_values = range(0, 3)

d\_values = range(0, 1)

q\_values = range(0, 3)

1. **Modeling**  
   Key Steps:  
   Fit the ARIMA model:  
   This refits the ARIMA model with the best parameters:

model = SARIMAX(data['Value'], order=best\_order, …)

results = model.fit(disp=False)

results.summary()

 gives a detailed output of coefficients, AIC, and diagnostics.

1. **Interpretation**  
   Key Steps:  
   Forecast future values:  
   The output of the model is the prediction

forecast\_steps

 steps ahead with confidence intervals.

**python**

**Copy code**

forecast = results.forecast(steps=forecast\_steps)

forecast\_ci = forecast.conf\_int()

Here is a plot of observed vs. predicted values:  
The observed series, forecasted mean, and confidence intervals are visualized.

**python**

**Copy code**

plt.plot(data['Value'], label='Observed', color='blue')

plt.plot(forecast\_index, forecast.predicted\_mean, name='Forecast', color='orange')

plt.fill\_between(…, color='orange', alpha=0.2)

1. **Diagnostics**  
   Key Steps:  
   Residual analysis:  
   Residuals (the difference between observed and fitted values) are plotted and tested for random distribution.

**python**

**Copy code**

residuals = results.resid

plt. # Residuals over time

plot(residuals)

plt.hist(residuals) # Residual histogram

Business – Statistical properties of residuals:  
Residuals have mean and variance computed for the residuals to make sure mean is approximately zero and variance is constant.

**python**

**Copy code**

print("Mean of residuals:", np.mean(residuals))

print("Variance of residuals:", np.var(residuals))

1. **Evaluation**  
   Key Steps:  
   Fit the ARIMA model manually:  
   We explicitly fit an ARIMA(1,1,1) model for further scrutiny.

**python**

**Copy code**

ARIMAXmodel = ARIMAX(data, order=(0, 1, 1), seasonal\_order=(0, 0, 1, 12))

arima\_result = arima\_model.fit()

Comparing original vs. fitted values:  
The original time series with fitted values overlaid.

**python**

**Copy code**

data['Fitted'] = arima\_result.fittedvalues

plt.plot(data['Value'], label='Original Data', alpha=0.7)

plt.plot(data['Fitted'], label='Fitted Values', linestyle='--')

Residual normality:  
Histogram and Q-Q plots confirm residual normal distribution.

**python**

**Copy code**

sns.histplot(residuals, kde=True, bins=30) # Let's see the residuals

stats.probplot(residuals, dist="norm", plot=plt)

Shapiro-Wilk test:

**python**

**Copy code**

shapiro\_test = stats.shapiro(residuals)

print(f'Shapiro-Wilk Test: W={shapiro\_test.statistic:.4f}, p-value={shapiro\_test.pvalue:.4e}')

1. **Final Outputs**
   * Optimal ARIMA Order: Found using grid search and confirmed with AIC.
   * Model Diagnostics — Checking assumptions using residual analysis
   * Save Time: View future trends with intervals of uncertainty.
2. **Potential Enhancements**
   * Automated differencing with the pmdarima library (auto\_arima).
   * Evaluate against reference models such as Naïve or Seasonal Naïve.
   * Use SARIMA to explicitly control for seasonality if relevant.