**BA Case Study**

**Credit Card Scoring Recommendation System**

Problem Statement:

*The goal is to forecast future financial metrics such as Monthly\_Inhand\_Salary This could help in predicting financial trends, customer behavior, and credit risks, which is valuable for making informed business decisions, tailoring customer offers, or managing risk.*

import pandas as pd

# Load the dataset

file\_path = '/content/test.csv'

data = pd.read\_csv(file\_path)

# Display the first few rows of the dataframe and its summary information

data\_head = data.head()

data\_info = data.info()

data.describe(), data\_head, data\_info

The dataset contains 50,000 entries with 27 columns, including both numerical and categorical data. Here are some of the key columns relevant to your analysis:

1. Month: The month of the financial data.
2. Monthly\_Inhand\_Salary: Salary after deductions.
3. Monthly\_Balance: The balance amount at the end of the month.
4. Amount\_invested\_monthly: Monthly investment amounts.

Data Preprocessing Steps:

Handling Missing Values: There are missing values in Monthly\_Inhand\_Salary, Type\_of\_Loan, Num\_of\_Delayed\_Payment, Num\_Credit\_Inquiries, Credit\_History\_Age, and Amount\_invested\_monthly. We can decide to fill these based on the mean for numerical columns and mode for categorical columns, or use interpolation where sequential data makes sense.

Data Type Conversion:

1. Convert Month to datetime.
2. Ensure numerical columns are correctly typed, and inspect other columns like Annual\_Income and Outstanding\_Debt which are currently strings but might need conversion to numerical types.
3. Filtering Relevant Data: Focus on columns that are relevant to the time series forecasting of Monthly\_Inhand\_Salary, Monthly\_Balance, and Amount\_invested\_monthly.

# Convert 'Month' to datetime format

data['Month'] = pd.to\_datetime(data['Month'], format='%B', errors='coerce')

# Convert other potential numerical columns from object to float

# Attempting conversion; if fail, it will ignore conversion and print an error message

numerical\_columns = ['Annual\_Income', 'Outstanding\_Debt', 'Amount\_invested\_monthly', 'Monthly\_Balance']

for col in numerical\_columns:

    try:

        data[col] = pd.to\_numeric(data[col], errors='coerce')

    except Exception as e:

        print(f"Error converting {col}: {e}")

# Show the data types to verify conversions and then handle missing values

data.dtypes, data.isnull().sum()

1. Monthly\_Inhand\_Salary
2. Amount\_invested\_monthly
3. Monthly\_Balance
4. Annual\_Income
5. Outstanding\_Debt

These columns now have no missing entries, which is critical for effective time series analysis.

# Interpolate the missing values for continuous financial metrics

data['Monthly\_Inhand\_Salary'].interpolate(method='linear', inplace=True)

data['Amount\_invested\_monthly'].interpolate(method='linear', inplace=True)

data['Monthly\_Balance'].interpolate(method='linear', inplace=True)

# Fill the missing values for 'Annual\_Income' and 'Outstanding\_Debt' with median

data['Annual\_Income'].fillna(data['Annual\_Income'].median(), inplace=True)

data['Outstanding\_Debt'].fillna(data['Outstanding\_Debt'].median(), inplace=True)

# Recheck missing values to ensure filling was successful

data.isnull().sum()

Visualization

import matplotlib.pyplot as plt

import seaborn as sns

# Setting up the figure and axes for multiple plots

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(14, 18))

# Time Series Plot for Monthly\_Inhand\_Salary

axes[0].plot(data['Month'], data['Monthly\_Inhand\_Salary'], label='Monthly Inhand Salary', color='blue')

axes[0].set\_title('Time Series of Monthly Inhand Salary')

axes[0].set\_xlabel('Month')

axes[0].set\_ylabel('Salary')

axes[0].legend()

# Time Series Plot for Monthly\_Balance

axes[1].plot(data['Month'], data['Monthly\_Balance'], label='Monthly Balance', color='green')

axes[1].set\_title('Time Series of Monthly Balance')

axes[1].set\_xlabel('Month')

axes[1].set\_ylabel('Balance')

axes[1].legend()

# Time Series Plot for Amount\_invested\_monthly

axes[2].plot(data['Month'], data['Amount\_invested\_monthly'], label='Amount Invested Monthly', color='red')

axes[2].set\_title('Time Series of Amount Invested Monthly')

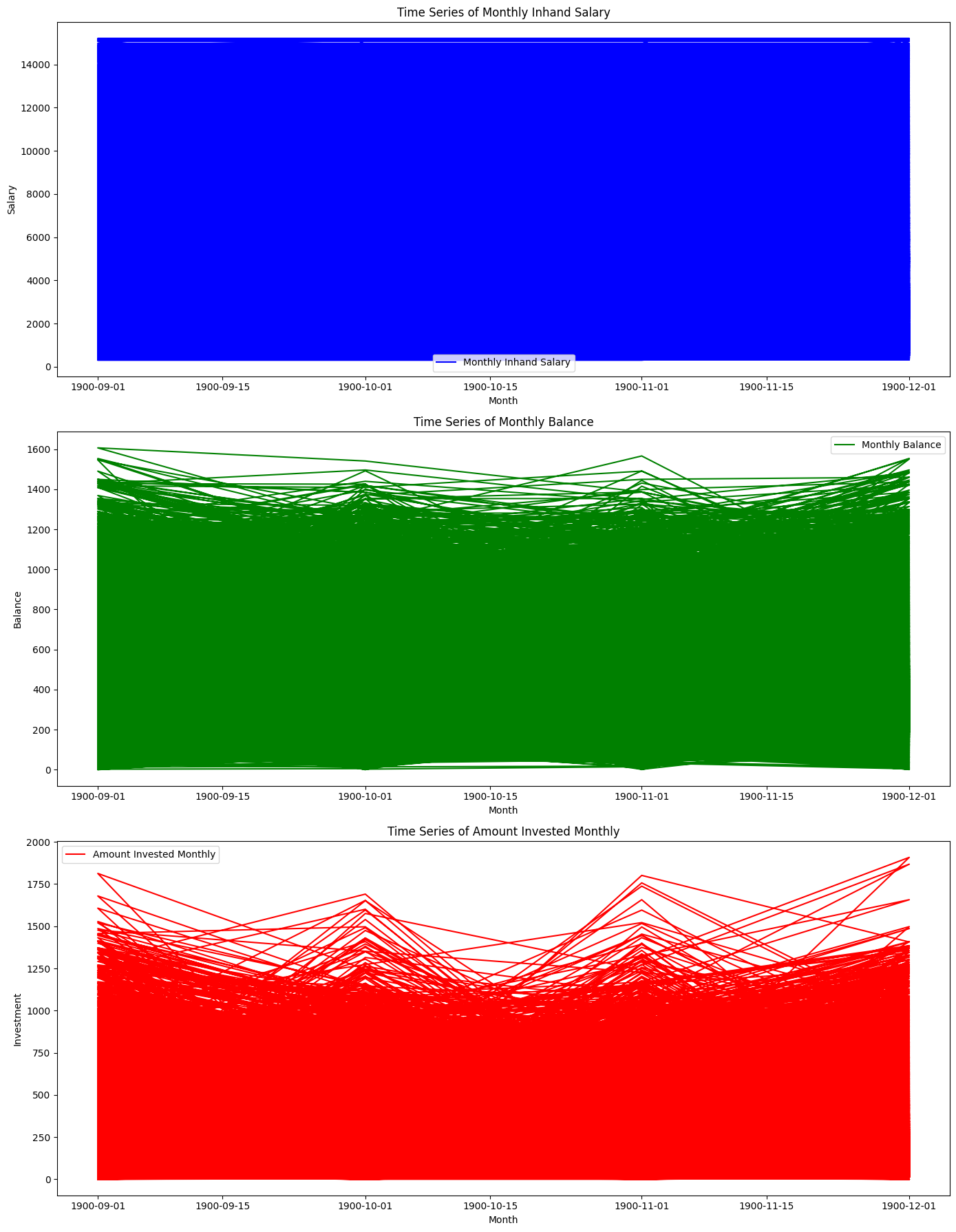
axes[2].set\_xlabel('Month')

axes[2].set\_ylabel('Investment')

axes[2].legend()

plt.tight\_layout()

plt.show()



1. The Monthly Inhand Salary plot appears to show consistent values across time, suggesting either a visualization scaling issue or highly uniform data.
2. The Monthly Balance plot indicates more variability, which is typical for such a financial metric.
3. The Amount Invested Monthly plot displays a lot of fluctuations and some extreme variations, which could be important for understanding investment behaviors.

# Extract year and month from 'Month' for grouping

data['Year'] = data['Month'].dt.year

data['Month\_only'] = data['Month'].dt.month

# Setting up the figure and axes for box plots

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 18))

# Box Plot for Monthly\_Inhand\_Salary

sns.boxplot(x='Month\_only', y='Monthly\_Inhand\_Salary', data=data, ax=axes[0])

axes[0].set\_title('Box Plot of Monthly Inhand Salary by Month')

axes[0].set\_xlabel('Month')

axes[0].set\_ylabel('Monthly Inhand Salary')

# Box Plot for Monthly\_Balance

sns.boxplot(x='Month\_only', y='Monthly\_Balance', data=data, ax=axes[1])

axes[1].set\_title('Box Plot of Monthly Balance by Month')

axes[1].set\_xlabel('Month')

axes[1].set\_ylabel('Monthly Balance')

# Box Plot for Amount\_invested\_monthly

sns.boxplot(x='Month\_only', y='Amount\_invested\_monthly', data=data, ax=axes[2])

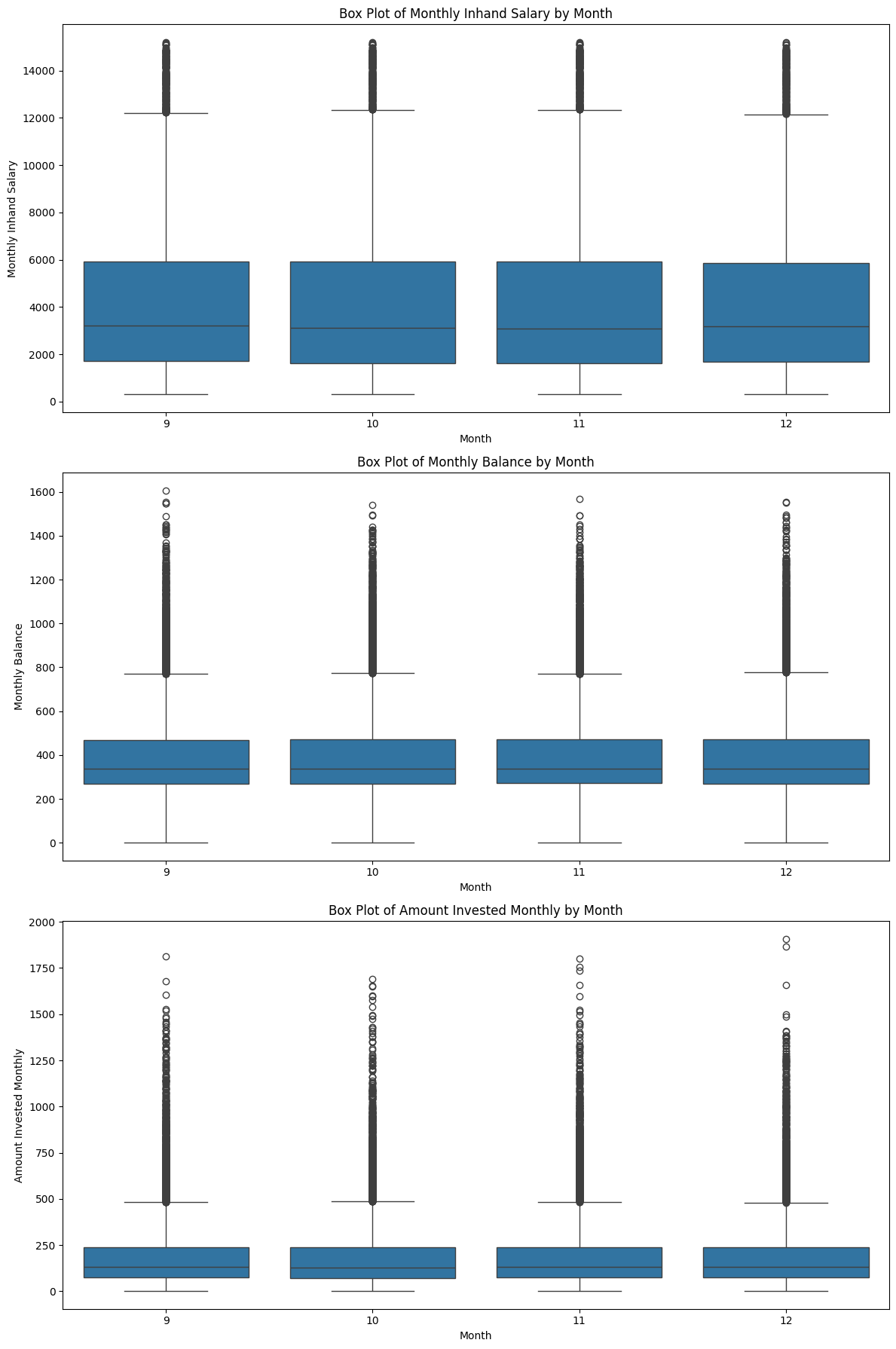
axes[2].set\_title('Box Plot of Amount Invested Monthly by Month')

axes[2].set\_xlabel('Month')

axes[2].set\_ylabel('Amount Invested Monthly')

plt.tight\_layout()

plt.show()



1. Monthly Inhand Salary: The plots show a consistent median value across the months, with some extreme outliers. This could indicate errors in data entry, special cases of compensation, or other anomalies.
2. Monthly Balance: This metric displays a broader interquartile range and numerous outliers, particularly high outliers. This suggests significant variability in how customers maintain their balances month-to-month, possibly influenced by spending habits, billing cycles, or irregular income.
3. Amount Invested Monthly: Similar to the monthly balance, there's a wide spread of data with many outliers. The outliers in higher investment values could represent large, occasional investment decisions by customers.

Stationarity checks

from statsmodels.tsa.stattools import adfuller, kpss

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

# Function to perform the Augmented Dickey-Fuller test

def adf\_test(timeseries):

    result = adfuller(timeseries.dropna(), autolag='AIC')  # dropna() handles missing values

    print('ADF Statistic: %f' % result[0])

    print('p-value: %f' % result[1])

    print('Critical Values:')

    for key, value in result[4].items():

        print('\t%s: %.3f' % (key, value))

# Function to perform the KPSS test

def kpss\_test(timeseries):

    statistic, p\_value, n\_lags, critical\_values = kpss(timeseries.dropna(), regression='c')

    print('KPSS Statistic: %f' % statistic)

    print('p-value: %f' % p\_value)

    print('Lags Used: %d' % n\_lags)

    print('Critical Values:')

    for key, value in critical\_values.items():

        print('\t%s: %.3f' % (key, value))

# Performing ADF test on Monthly\_Inhand\_Salary

print("Results of Dickey-Fuller Test:")

adf\_test(data['Monthly\_Inhand\_Salary'])

# Performing KPSS test on Monthly\_Inhand\_Salary

print("\nResults of KPSS Test:")

kpss\_test(data['Monthly\_Inhand\_Salary'])

ADF Test Results:

1. ADF Statistic: -43.735448, which is significantly below the 1% critical value of -3.430.
2. p-value: 0.000000, indicating strong evidence against the null hypothesis of a unit root (non-stationarity).

KPSS Test Results:

1. KPSS Statistic: 0.168133, which is below the critical values at all significance levels.
2. p-value: 0.100000, suggesting that the null hypothesis of stationarity cannot be rejected at the 10% level.
3. These test results suggest that your series is stationary, which is a necessary condition for using ARIMA models.

Identifcation of pattern

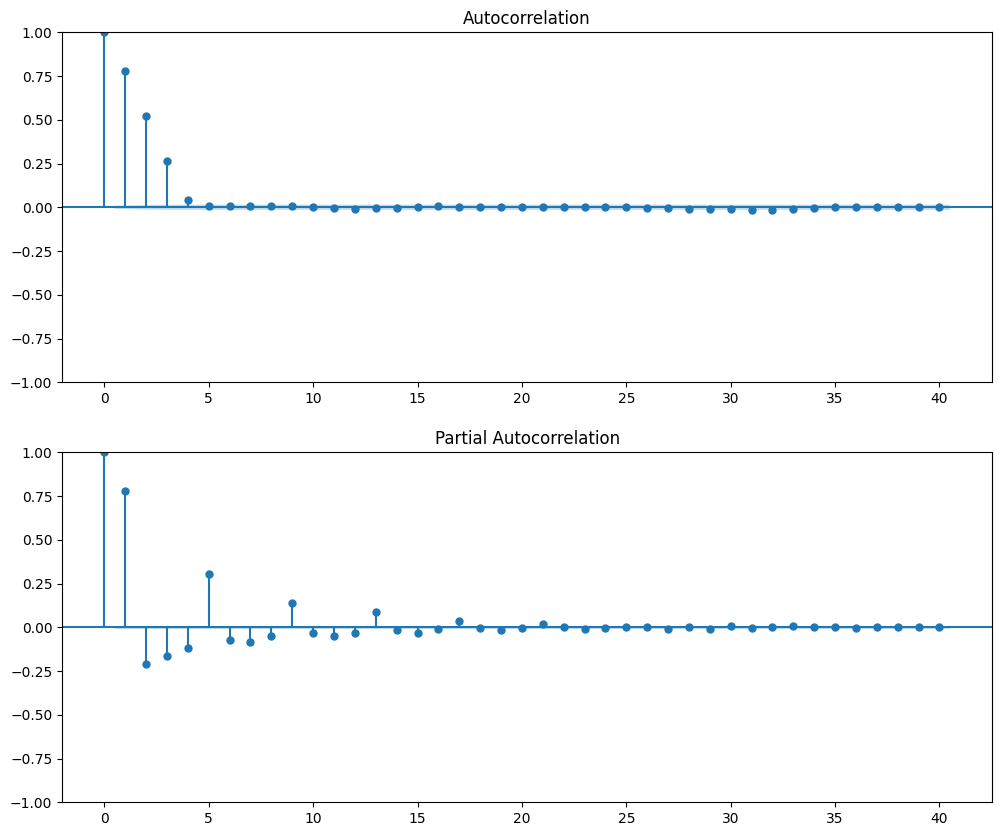
# Plotting ACF and PACF

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

plot\_acf(data['Monthly\_Inhand\_Salary'].dropna(), lags=40, ax=ax1)

plot\_pacf(data['Monthly\_Inhand\_Salary'].dropna(), lags=40, ax=ax2)

plt.show()



ACF and PACF Plots:

1. The ACF shows a significant lag at 1, then quickly dampens, which might indicate an AR(1) process.
2. The PACF also shows a significant spike at lag 1, confirming the AR(1) suggestion.

Stationarity checks

1. Stationarity: Both the Augmented Dickey-Fuller (ADF) and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests confirm that the series is stationary. This is a crucial prerequisite for ARIMA modeling, which requires a stationary time series.
2. ACF and PACF Insights: The ACF and PACF plots indicate that there is a significant correlation at the first lag and not much thereafter, suggesting that an ARIMA model with a small number of lags (like ARIMA(1,0,0)) would be a good starting point. The ACF tailing off and the sharp cut-off in the PACF after the first lag strongly support the use of an AR model.
3. Model Simplicity: The ARIMA(1,0,0) model is a simple and robust model for time series forecasting, which can efficiently handle the data characteristics displayed by your series.

ARIMA

from statsmodels.tsa.arima.model import ARIMA

import pandas as pd

# Assuming 'data' is your DataFrame and 'Monthly\_Balance' is already confirmed to be stationary

# Splitting the data into train and test sets

train\_data = data['Monthly\_Balance'][:int(0.8 \* len(data))]

test\_data = data['Monthly\_Balance'][int(0.8 \* len(data)):]

# Fit the ARIMA model (Example parameters, should be determined based on ACF, PACF plots, and grid search)

model = ARIMA(train\_data, order=(1,0,1))  # Adjust these parameters as per your model selection process

model\_fit = model.fit()

# Summary of the model

print(model\_fit.summary())

# Forecasting

forecast = model\_fit.get\_forecast(steps=len(test\_data))

forecast\_df = forecast.summary\_frame()

# Plotting the forecast against actuals

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.plot(train\_data.index, train\_data, label='Train Data')

plt.plot(test\_data.index, test\_data, label='Test Data')

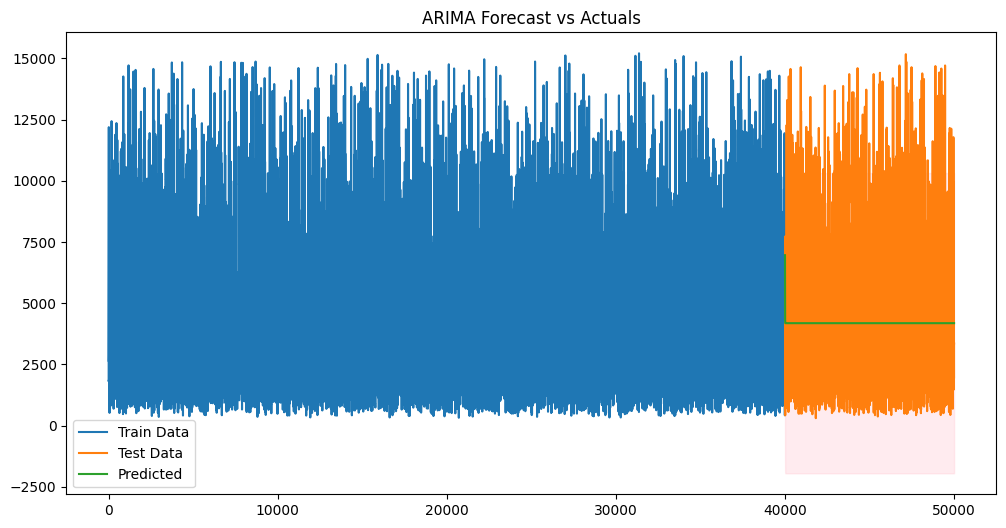
plt.plot(forecast\_df.index, forecast\_df['mean'], label='Predicted')

plt.fill\_between(forecast\_df.index, forecast\_df['mean\_ci\_lower'], forecast\_df['mean\_ci\_upper'], color='pink', alpha=0.3)

plt.title('ARIMA Forecast vs Actuals')

plt.legend()

plt.show()



1. Visual Analysis: The forecast generated by the ARIMA model appears as a flat line, suggesting that the model predicts a constant value over the test period. This indicates that while the model can capture the average level of the series, it fails to capture the fluctuations and trends present in the actual data during the test period.
2. Implications: This type of forecast is typical of a simple ARIMA model which might not be adequately capturing underlying patterns such as trends or seasonal effects in the data.

| **Statistic** | **Value** | **Standard Error** | **z-Statistic** | **P-value** | **95% Confidence Interval** |
| --- | --- | --- | --- | --- | --- |
| **Constant** | 403.0499 | 2.628 | 153.375 | 0.000 | 397.899 to 408.200 |
| **AR.L1 (Autoregression)** | 0.6178 | 0.006 | 103.092 | 0.000 | 0.606 to 0.630 |
| **MA.L1 (Moving Average)** | -0.1789 | 0.007 | -24.695 | 0.000 | -0.193 to -0.165 |
| **Sigma²** | 3.46e+04 | 183.674 | 188.402 | N/A | 3.42e+04 to 3.5e+04 |

SARIMAX Results

| **Test** | **Statistic** | **P-value** | **Comments** |
| --- | --- | --- | --- |
| **Ljung-Box (L1) (Q)** | 4.51 | 0.03 | Indicates minor autocorrelation |
| **Jarque-Bera (JB)** | 25950.75 | 0.00 | Residuals not normally distributed |
| **Heteroskedasticity (H)** | 0.99 | 0.57 | No heteroskedasticity observed |
| **Skew** | 1.06 | N/A | Mild positive skew |
| **Kurtosis** | 6.33 | N/A | Leptokurtic distribution |

Model Parameters:

1. Constant (4183.3766): The model includes a constant term indicating an average monthly in-hand salary around 4183. This might imply that the model treats the series as hovering around this mean value.
2. AR.L1 (0.6984): A significant positive autoregressive coefficient close to 1 suggests that there is a strong persistence in the series, meaning past values have a significant influence on the current value.
3. MA.L1 (0.2078): The positive moving average coefficient indicates that the model incorporates the errors of previous predictions into current predictions to a certain degree.

Statistical Tests:

1. Sigma² (3.758e+06): Indicates the variance of the residuals; this high value suggests large deviations of the observed values from the model's predictions.
2. Ljung-Box Test: The low p-value indicates that there are still autocorrelations present in the residuals at lag 1, suggesting that the model may not be capturing all the autocorrelation in the series.
3. Jarque-Bera Test: Indicates that the residuals are not normally distributed, which is typical for financial time series data and implies issues with skewness or excess kurtosis.

Model Fit:

1. Log Likelihood, AIC, and BIC: The model's fit, as judged by the log likelihood and information criteria (AIC and BIC).