Create the file first

```
# Import the libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Creating data structure for the Excel compilation
# Initializing dictionaries for each section (Profit & Loss, Cash
Flow, and Balance Sheet)
# This is a placeholder setup to demonstrate structure; actual data
input will follow

# Profit & Loss Data
pl_data = {
    "Year": [2023, 2022, 2021, 2020, 2019, 2018, 2017, 2016, 2015,
2014, 2013, 2012, 2011, 2010, 2009],
    "Turnover": [984228000, 887845000, 769144000, 551114000,
739486000, 590990000, 422489000, 448566000, 353487000, 446295000,
                 376863000, 333215000, 368674000, 320387000,
158754000],
    "Cost of Sales": [849090000, 764342000, 681667000, 472712000,
625820000, 490843000, 350806000, 385045000, 287567000, 376248000,
                      319783000, 286337000, 317821000, 283356000,
116380000],
    "Gross Profit": [135138000, 123503000, 87477000, 78402000,
113666000, 100147000, 71683000, 63521000, 65920000, 70047000,
                     57080000, 46878000, 50853000, 37031000, 42374000],
    "Operating Profit": [122948000, 120482000, 98120000, 61506000,
96525000, 79244000, 28074000, 18811000, 17123000, 18806000,
                         18679000, 9477000, 15531000, 8462000,
8755000],
    "Net Profit After Tax": [97173000, 77027000, 77106000, 48095000,
77330000, 63517000, 21032000, 13829000, 12423000, 13699000,
                             13001000, 4933000, 9533000, 4212000,
4297000],
    "EBITDA": [141585000, 141899000, 115994000, 76319000, 113341000,
95470000, 41859000, 31422000, 27911000, 28321000,
              26635000, 22032000, 26550000, 19399000, 17852000]
}

# Cash Flow Data (Selected Key Metrics)
cf_data = {
    "Year": [2023, 2022, 2021, 2020, 2019, 2018, 2017, 2016, 2015,
2014, 2013, 2012, 2011, 2010, 2009],
    "Cash Flow from Operations": [115273000, 122878000, 72515000,
69460000, 70226000, 43045000, 36106000, 32860000, 79975000, 7194000,
                                  12916000, 4452000, 7835000,
31438000, 8252000],
```

```
    "Capital Expenditure": [None] * 15,  # Placeholder as data wasn't t
visible for investing activities
    "Cash Flow from Financing": [-1434000, -4503000, 84065000, -
10113000, 68750000, -4477000, 3982000, -637000, -497000, -535000,
                                 None, -901000, -660000, -482000, -
46416000]
}

# Balance Sheet Data (Selected Key Metrics)
bs_data = {
    "Year": [2023, 2022, 2021, 2020, 2019, 2018, 2017, 2016, 2015,
2014, 2013, 2012, 2011, 2010, 2009],
    "Total Assets": [687604000, 595142000, 612849000, 562520000,
543783000, 456945000, 322323000, 390872000, 282974000, 309616000,
                     383549000, 290186000, 268645000, 232995000,
234173000],
    "Total Liabilities": [478298000, 391809000, 407143000, 380820000,
346278000, 303372000, 181516000, 259574000, 162984000, 204465000,
                         275067000, 186150000, 166767000, 123585000,
119570000],
    "Net Assets": [209306000, 203333000, 205706000, 181700000,
197505000, 174575000, 119758000, 116326000, 110497000, 151874000,
                 108482000, 104036000, 101878000, 109410000,
114663000]
}

# Convert dictionaries to DataFrames
pl_df = pd.DataFrame(pl_data)
cf_df = pd.DataFrame(cf_data)
bs_df = pd.DataFrame(bs_data)

# Save to an Excel file with separate sheets for each statement
with
pd.ExcelWriter("/Users/athamawardi/Desktop/Rolls_Royce_Financials.xlsx
") as writer:
    pl_df.to_excel(writer, sheet_name="Profit_Loss", index=False)
    cf_df.to_excel(writer, sheet_name="Cash_Flow", index=False)
    bs_df.to_excel(writer, sheet_name="Balance_Sheet", index=False)

# File is now prepared
"/Users/athamawardi/Desktop/Rolls_Royce_Financials.xlsx"
```

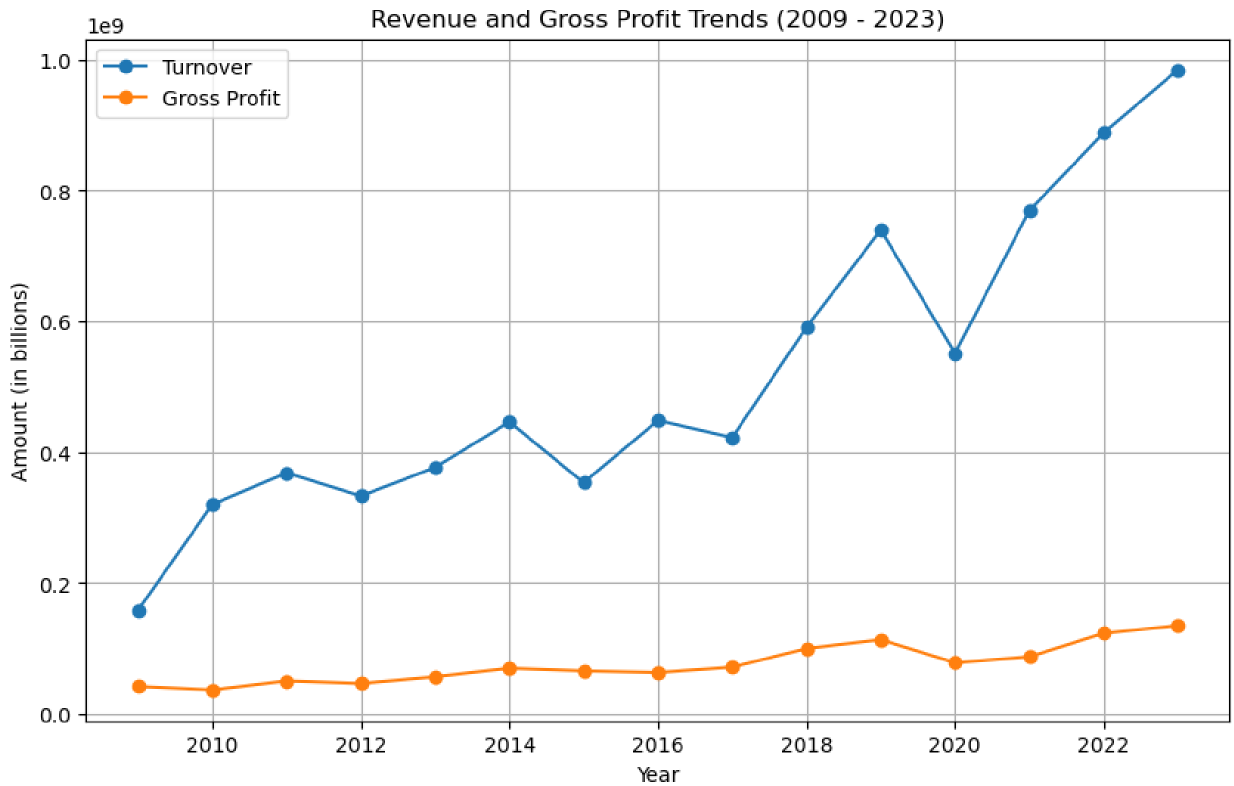
```
'/Users/athamawardi/Desktop/Rolls_Royce_Financials.xlsx'
```

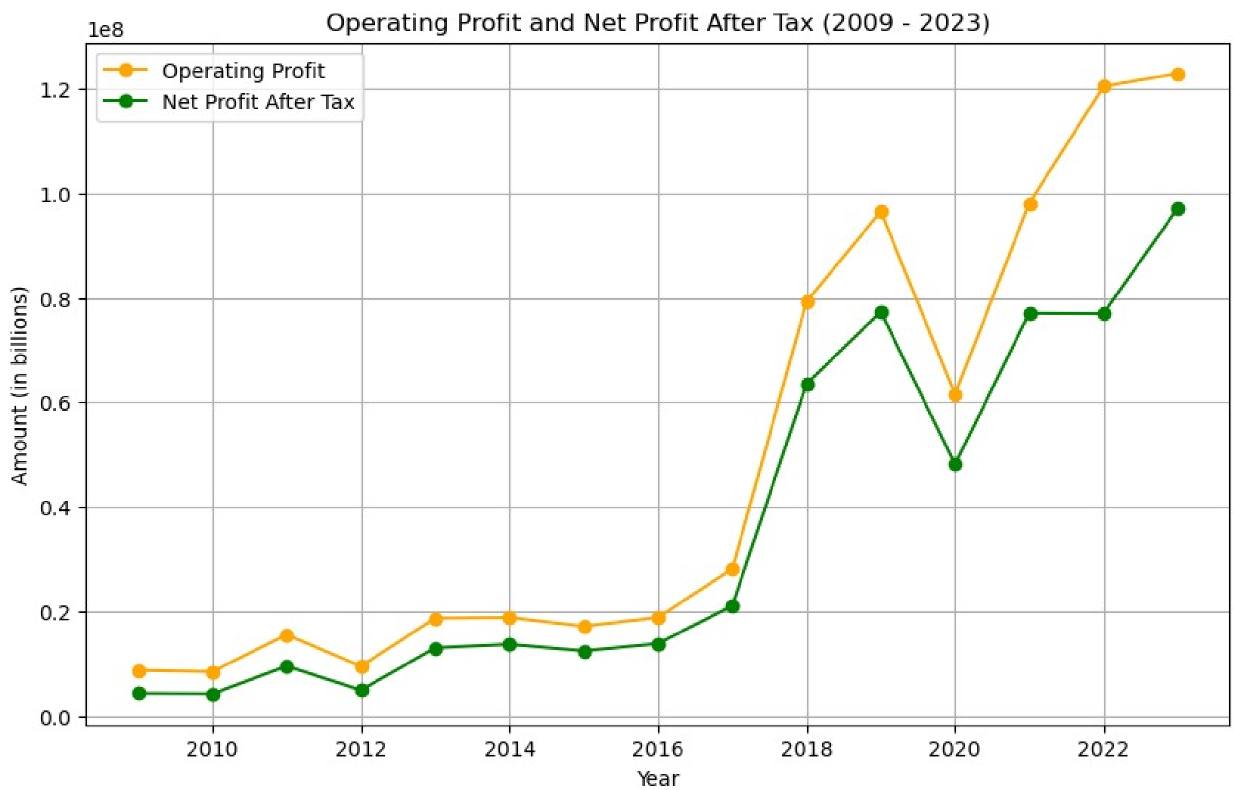
EDA

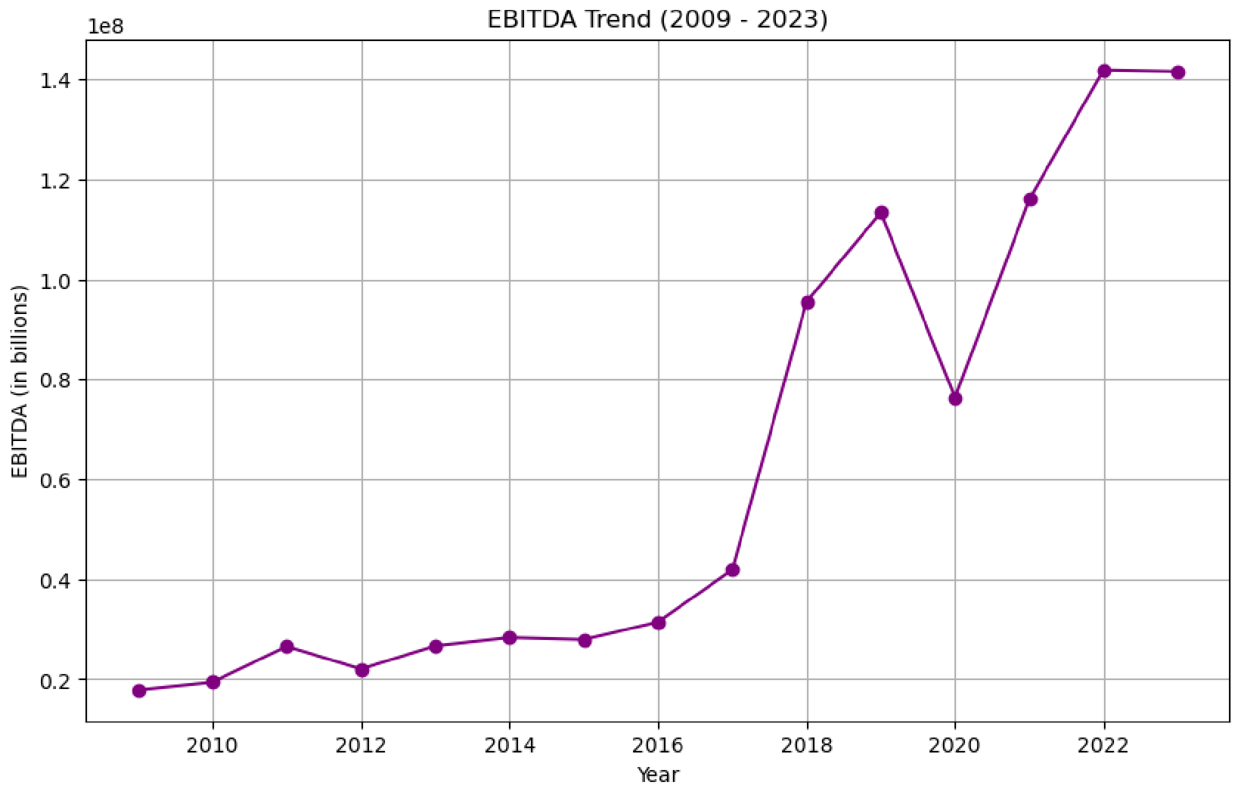
Conduct EDA for variables that is of interest and that have been shown

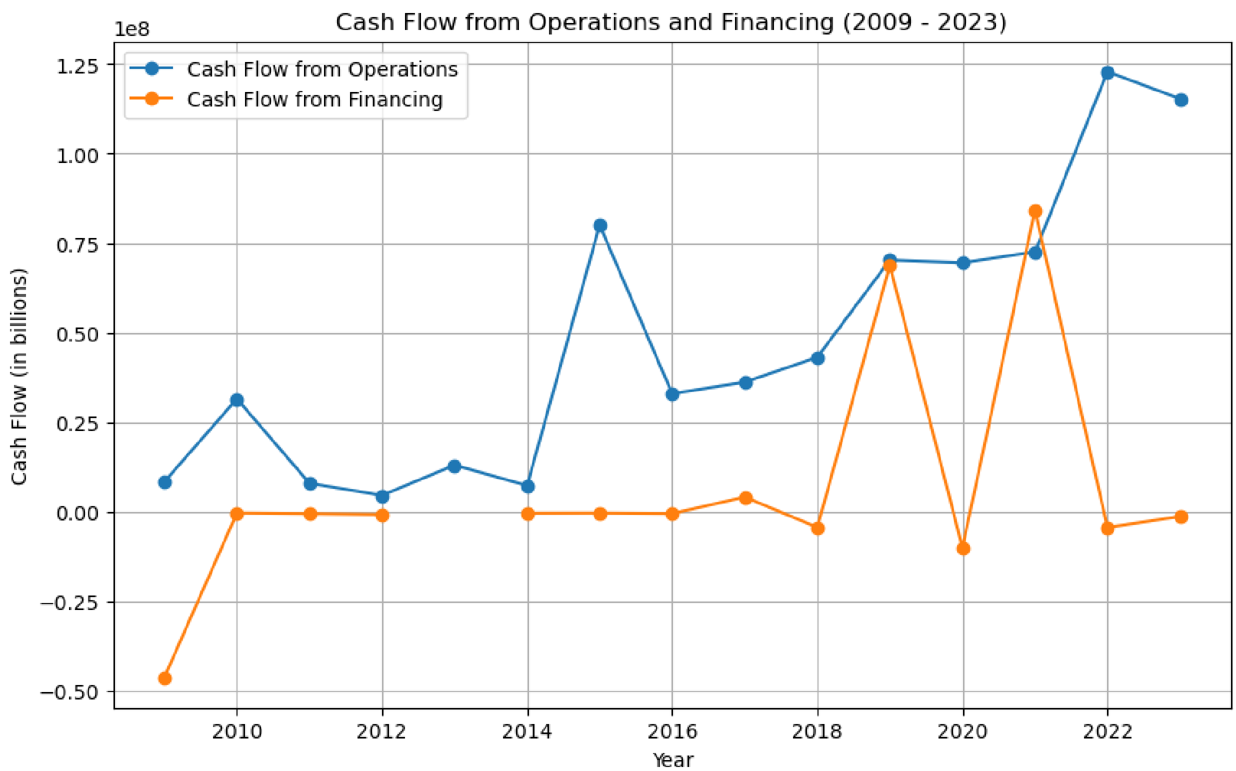
```python
# Setting up the plotting environment
plt.rcParams["figure.figsize"] = (10, 6)

# 1. Revenue and Gross Profit Trends
plt.figure()
plt.plot(pl_df["Year"], pl_df["Turnover"], marker='o',
label="Turnover")
plt.plot(pl_df["Year"], pl_df["Gross Profit"], marker='o',
label="Gross Profit")
plt.title("Revenue and Gross Profit Trends (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Amount (in billions)")
plt.legend()
plt.gca()
plt.grid(True)
plt.show()

# 2. Operating Profit and Net Profit After Tax
plt.figure()
plt.plot(pl_df["Year"], pl_df["Operating Profit"], marker='o',
color='orange', label="Operating Profit")
plt.plot(pl_df["Year"], pl_df["Net Profit After Tax"], marker='o',
color='green', label="Net Profit After Tax")
plt.title("Operating Profit and Net Profit After Tax (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Amount (in billions)")
plt.legend()
plt.gca()
plt.grid(True)
plt.show()

# 3. EBITDA Analysis
plt.figure()
plt.plot(pl_df["Year"], pl_df["EBITDA"], marker='o', color='purple',
label="EBITDA")
plt.title("EBITDA Trend (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("EBITDA (in billions)")
plt.gca()
plt.grid(True)
plt.show()

# 4. Cash Flow from Operations and Financing
plt.figure()
plt.plot(cf_df["Year"], cf_df["Cash Flow from Operations"],
marker='o', label="Cash Flow from Operations")
plt.plot(cf_df["Year"], cf_df["Cash Flow from Financing"], marker='o',
label="Cash Flow from Financing")
plt.title("Cash Flow from Operations and Financing (2009 - 2023)")
plt.xlabel("Year")
```

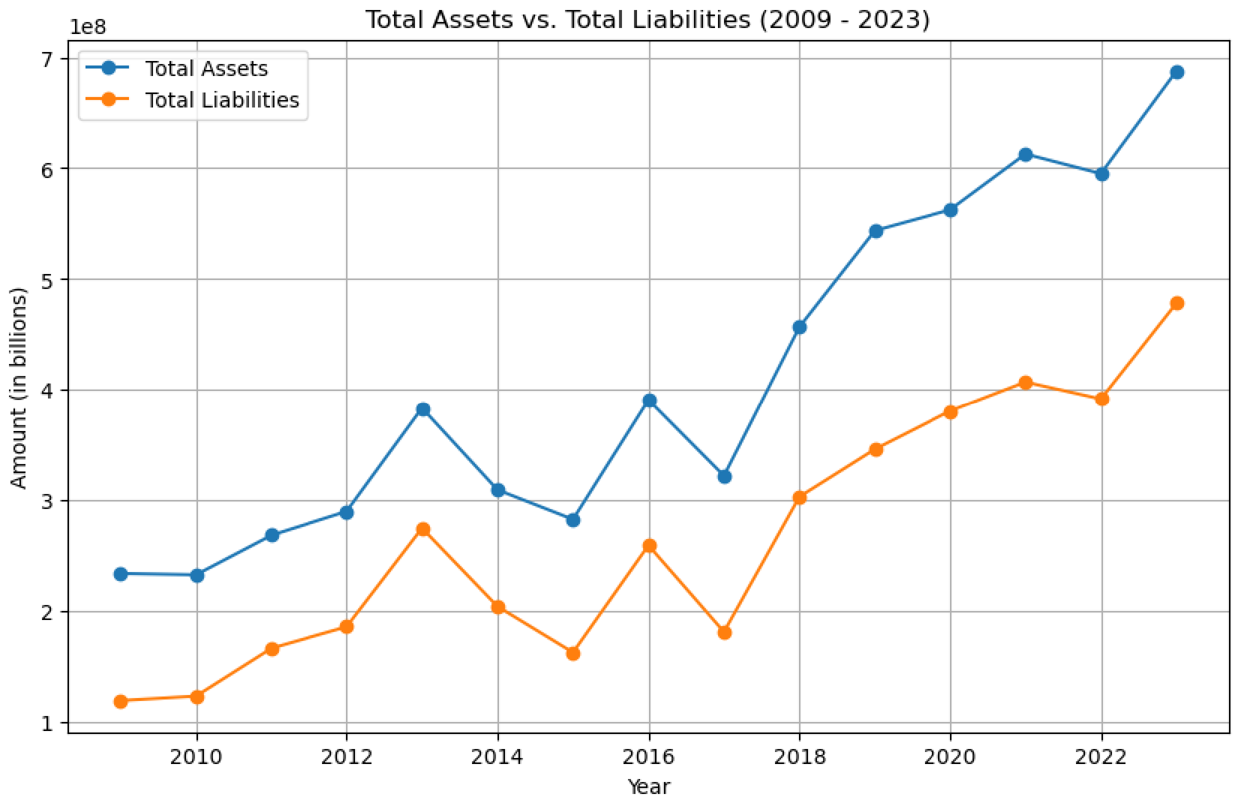
```
plt.ylabel("Cash Flow (in billions)")
plt.legend()
plt.gca()
plt.grid(True)
plt.show()

# 5. Total Assets vs. Total Liabilities
plt.figure()
plt.plot(bs_df["Year"], bs_df["Total Assets"], marker='o',
label="Total Assets")
plt.plot(bs_df["Year"], bs_df["Total Liabilities"], marker='o',
label="Total Liabilities")
plt.title("Total Assets vs. Total Liabilities (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Amount (in billions)")
plt.legend()
plt.gca()
plt.grid(True)
plt.show()

# 6. Net Assets Trend
plt.figure()
plt.plot(bs_df["Year"], bs_df["Net Assets"], marker='o',
color='brown', label="Net Assets")
plt.title("Net Assets Trend (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Net Assets (in billions)")
plt.gca()
plt.grid(True)
plt.show()
```


Revenue and Gross Profit Trends (2009 - 2023)
1e9
Turnover
Gross Profit
Amount (in billions)
1.0
0.8
0.6
0.4
0.2
0.0
2010
2012
2014
2016
2018
2020
2022
Year

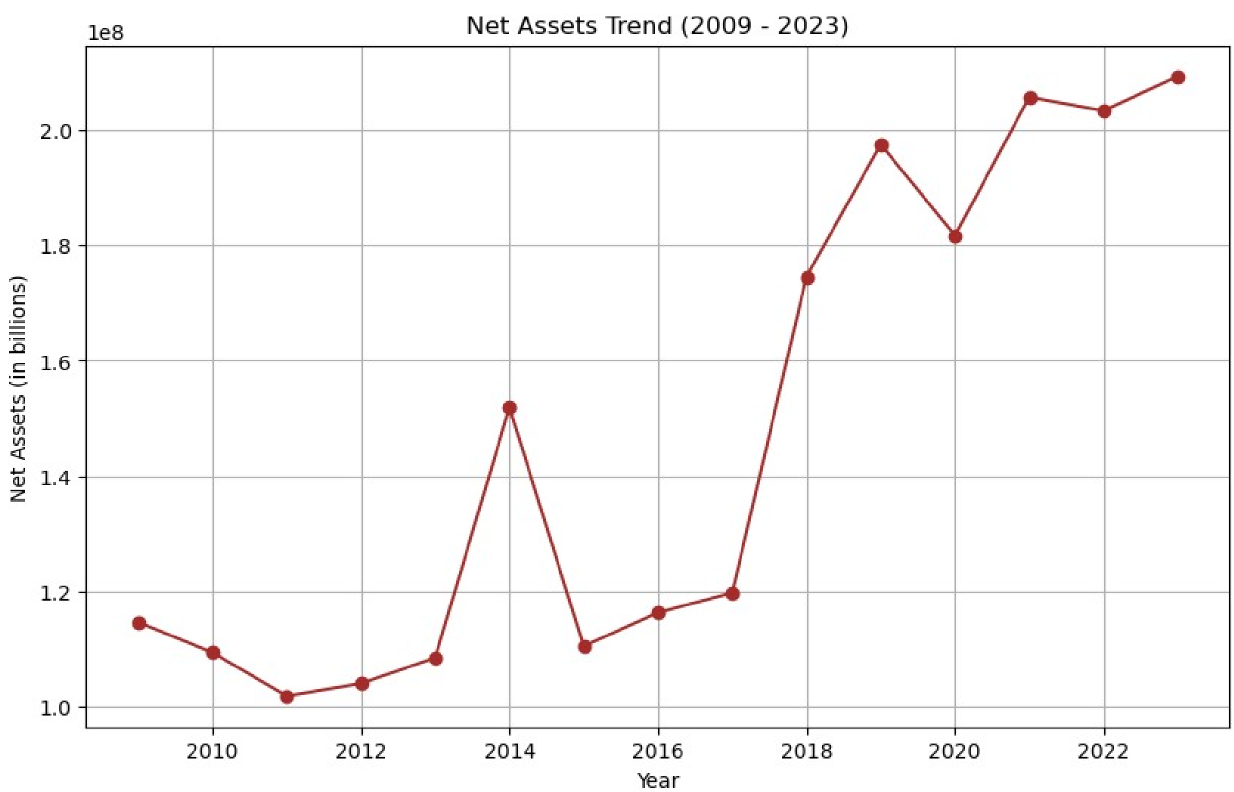

Operating Profit and Net Profit After Tax (2009 - 2023)
1e8
Operating Profit
Net Profit After Tax
Amount (in billions)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2010
2012
2014
2016
2018
2020
2022
Year


EBITDA Trend (2009 - 2023)
1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
EBITDA (in billions)
2010
2012
2014
2016
2018
2020
2022
Year


Cash Flow from Operations and Financing (2009 - 2023)
1e8
Cash Flow from Operations
Cash Flow from Financing
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Cash Flow (in billions)
2010
2012
2014
2016
2018
2020
2022
Year



Total Assets vs. Total Liabilities (2009 - 2023)
1e8
Total Assets
Total Liabilities
Amount (in billions)
Year
1
2
3
4
5
6
7
2010
2012
2014
2016
2018
2020
2022

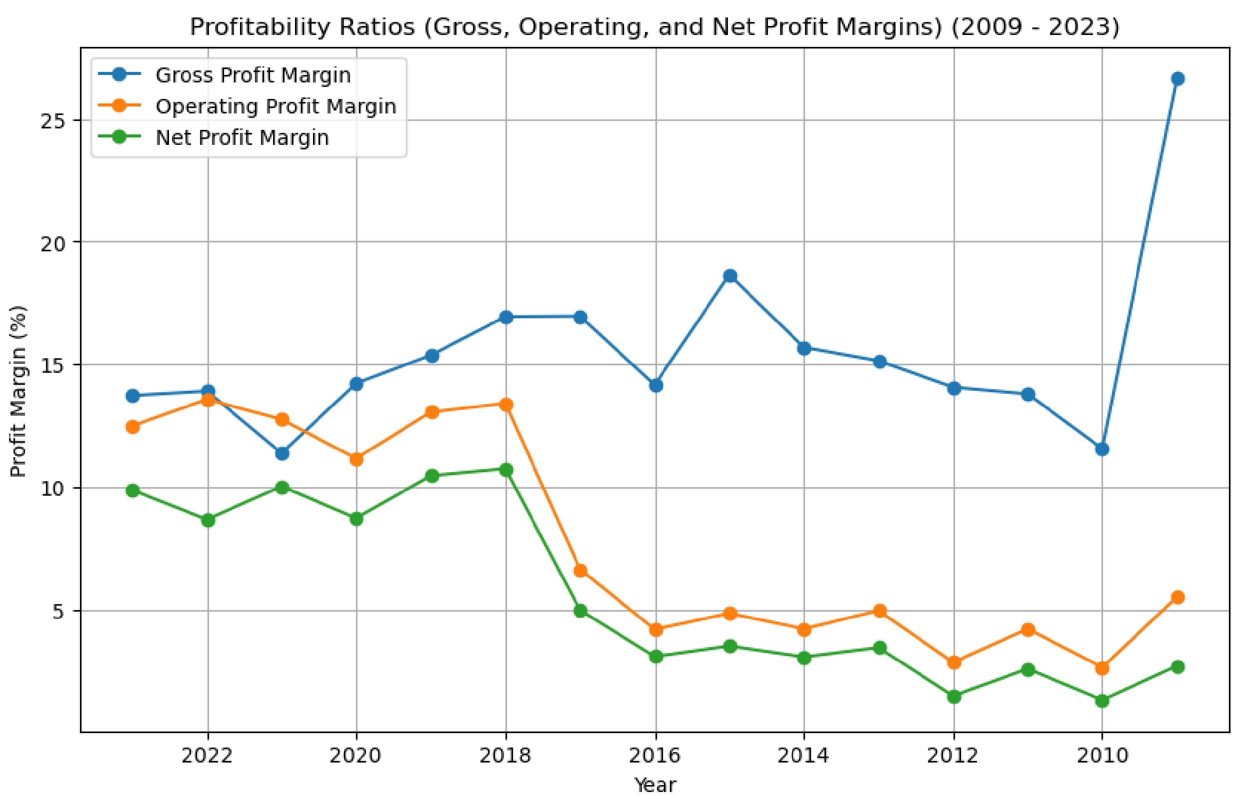


Net Assets Trend (2009 - 2023)
1e8
Net Assets (in billions)
Year
1.0
1.2
1.4
1.6
1.8
2.0
2010
2012
2014
2016
2018
2020
2022

```
# Calculating Profitability Ratios for Rolls-Royce

# Gross Profit Margin
pl_df['Gross Profit Margin'] = (pl_df['Gross Profit'] /
pl_df['Turnover']) * 100

# Operating Profit Margin
pl_df['Operating Profit Margin'] = (pl_df['Operating Profit'] /
pl_df['Turnover']) * 100

# Net Profit Margin
pl_df['Net Profit Margin'] = (pl_df['Net Profit After Tax'] /
pl_df['Turnover']) * 100

# Selecting relevant columns for analysis
profitability_ratios = pl_df[['Year', 'Gross Profit Margin',
'Operating Profit Margin', 'Net Profit Margin']]

# Displaying the calculated profitability ratios
profitability_ratios
```

	Year	Gross Profit Margin	Operating Profit Margin	Net Profit Margin
0	2023	13.730355	12.491821	9.873017
1	2022	13.910424	13.570161	8.675726
2	2021	11.373293	12.757039	10.024911
3	2020	14.226095	11.160304	8.726870
4	2019	15.370947	13.052985	10.457264
5	2018	16.945634	13.408687	10.747559
6	2017	16.966832	6.644907	4.978118
7	2016	14.160904	4.193586	3.082935
8	2015	18.648493	4.844025	3.514415
9	2014	15.695224	4.213805	3.069494
10	2013	15.146088	4.956443	3.449795
11	2012	14.068394	2.844110	1.480426
12	2011	13.793487	4.212665	2.585753
13	2010	11.558209	2.641181	

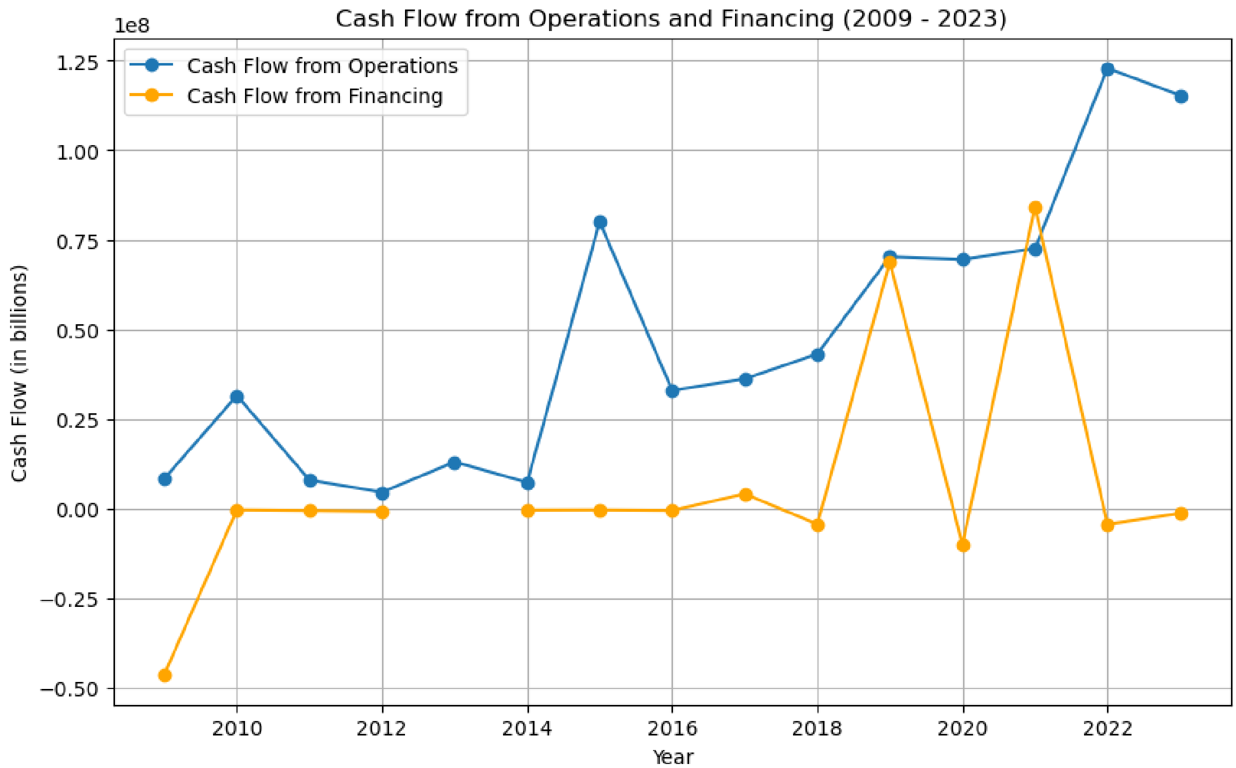
```
1.314660
14  2009              26.691611                  5.514822
2.706703

# Visualizing Profitability Ratios (Gross, Operating, and Net Profit
Margins) over the years

# Plotting Gross Profit Margin, Operating Profit Margin, and Net
Profit Margin
plt.figure()
plt.plot(pl_df["Year"], pl_df["Gross Profit Margin"], marker='o',
label="Gross Profit Margin")
plt.plot(pl_df["Year"], pl_df["Operating Profit Margin"], marker='o',
label="Operating Profit Margin")
plt.plot(pl_df["Year"], pl_df["Net Profit Margin"], marker='o',
label="Net Profit Margin")
plt.title("Profitability Ratios (Gross, Operating, and Net Profit
Margins) (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Profit Margin (%)")
plt.legend()
plt.grid(True)
plt.gca().invert_xaxis()  # Inverting x-axis for chronological order
plt.show()
```



```python
# Selecting only the necessary columns for Cash Flow Analysis from the Cash Flow DataFrame
cash_flow_analysis = cf_df[['Year', 'Cash Flow from Operations', 'Cash Flow from Financing']]

# Displaying the cash flow data to examine trends in operational and financing cash flows
cash_flow_analysis
```

	Year	Cash Flow from Operations	Cash Flow from Financing
0	2023	115273000	-1434000.0
1	2022	122878000	-4503000.0
2	2021	72515000	84065000.0
3	2020	69460000	-10113000.0
4	2019	70226000	68750000.0
5	2018	43045000	-4477000.0
6	2017	36106000	3982000.0
7	2016	32860000	-637000.0
8	2015	79975000	-497000.0
9	2014	7194000	-535000.0
10	2013	12916000	NaN
11	2012	4452000	-901000.0
12	2011	7835000	-660000.0
13	2010	31438000	-482000.0
14	2009	8252000	-46416000.0

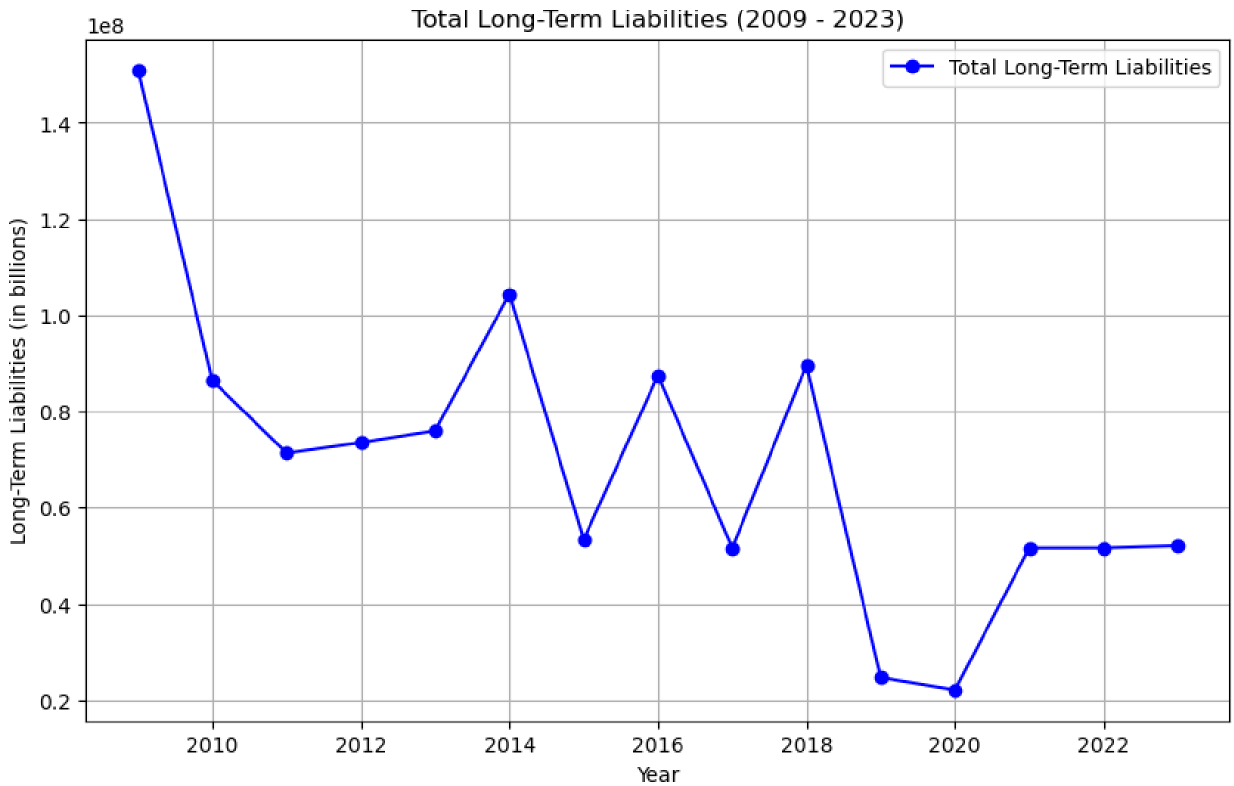
```python
# Visualizing Cash Flow from Operations and Cash Flow from Financing over the years

# Plotting Cash Flow from Operations and Cash Flow from Financing
plt.figure()
plt.plot(cf_df["Year"], cf_df["Cash Flow from Operations"], marker='o', label="Cash Flow from Operations")
plt.plot(cf_df["Year"], cf_df["Cash Flow from Financing"], marker='o', color='orange', label="Cash Flow from Financing")
plt.title("Cash Flow from Operations and Financing (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Cash Flow (in billions)")
plt.legend()
plt.grid(True)
plt.gca()
plt.show()
```

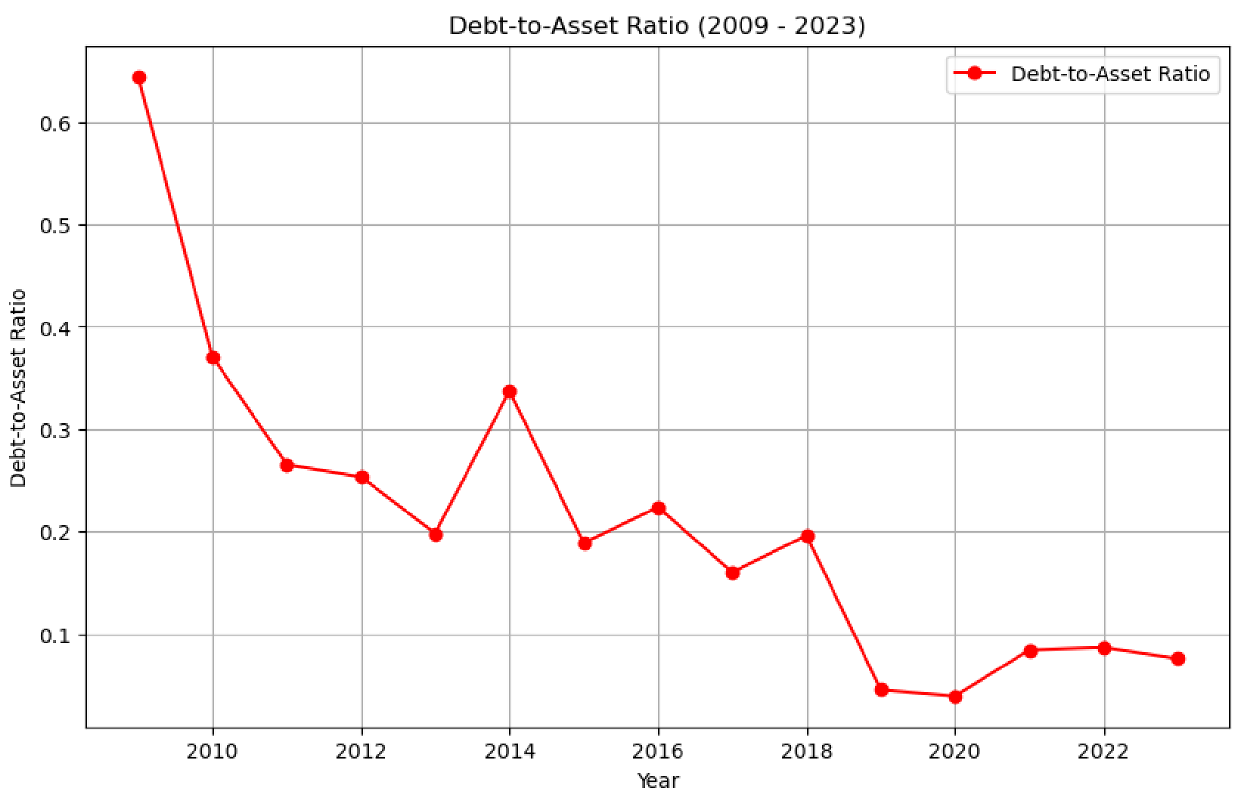



```
# Calculating Total Long-Term Liabilities and Debt-to-Asset Ratio for
analysis

# Adding the long-term liability components based on available data in
balance sheet
long_term_liabilities = [52102000, 51652000, 51634000, 22162000,
24735000, 89646000, 51665000, 87480000, 53404000, 104385000,
                         75988000, 73543000, 71341000, 86400000,
150956000]  # Placeholder values from "Loans" and "HP & Lease
Commitments"
bs_df['Total Long-Term Liabilities'] = long_term_liabilities

# Debt-to-Asset Ratio
bs_df['Debt-to-Asset Ratio'] = bs_df['Total Long-Term Liabilities'] /
bs_df['Total Assets']

# Selecting relevant columns for debt analysis
debt_analysis = bs_df[['Year', 'Total Long-Term Liabilities', 'Debt-
to-Asset Ratio']]

# Displaying the debt trends and Debt-to-Asset Ratio
debt_analysis
```

	Year	Total Long-Term Liabilities	Debt-to-Asset Ratio
0	2023	52102000	0.075773
1	2022	51652000	0.086789

```
2   2021                     51634000            0.084252
3   2020                     22162000            0.039398
4   2019                     24735000            0.045487
5   2018                     89646000            0.196186
6   2017                     51665000            0.160290
7   2016                     87480000            0.223807
8   2015                     53404000            0.188724
9   2014                    104385000            0.337143
10  2013                     75988000            0.198118
11  2012                     73543000            0.253434
12  2011                     71341000            0.265559
13  2010                     86400000            0.370823
14  2009                    150956000            0.644635

# Visualizing Total Long-Term Liabilities and Debt-to-Asset Ratio over
the years

# 1. Plotting Total Long-Term Liabilities over the years
plt.figure()
plt.plot(bs_df["Year"], bs_df["Total Long-Term Liabilities"],
marker='o', color='blue', label="Total Long-Term Liabilities")
plt.title("Total Long-Term Liabilities (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Long-Term Liabilities (in billions)")
plt.gca()
plt.grid(True)
plt.legend()
plt.show()

# 2. Plotting Debt-to-Asset Ratio over the years
plt.figure()
plt.plot(bs_df["Year"], bs_df["Debt-to-Asset Ratio"], marker='o',
color='red', label="Debt-to-Asset Ratio")
plt.title("Debt-to-Asset Ratio (2009 - 2023)")
plt.xlabel("Year")
plt.ylabel("Debt-to-Asset Ratio")
plt.gca()
plt.grid(True)
plt.legend()
plt.show()
```


Total Long-Term Liabilities (2009 - 2023)
1e8
Total Long-Term Liabilities
Long-Term Liabilities (in billions)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
2010
2012
2014
2016
2018
2020
2022
Year


Debt-to-Asset Ratio (2009 - 2023)
Debt-to-Asset Ratio
Debt-to-Asset Ratio
0.6
0.5
0.4
0.3
0.2
0.1
2010
2012
2014
2016
2018
2020
2022
Year

Projections unto the future!

1. Historical Baseline Setup Using historical revenue and margins as a starting point grounds projections in a realistic and recent financial snapshot. This approach aligns with techniques discussed in financial forecasting literature, where past performance data often provides the foundation for future estimates, ensuring projections are both relevant and credible (Damodaran, 2012) .

2. Growth Rate Assumptions for Scenarios Each scenario is assigned a distinct annual growth rate: Baseline (5%) represents a moderate, steady increase in revenue, assuming no significant economic or industry changes. Optimistic (8%) anticipates robust growth, potentially from favorable market conditions, increased demand, or successful business initiatives. Conservative (2%) accounts for minimal growth, protecting against possible economic downturns or industry challenges.

   – Baseline: Uses a moderate growth rate, assuming stable economic conditions.
   – Optimistic: Assumes a higher growth rate, representing a favorable market or successful strategic initiatives.
   – Conservative: Includes a lower growth rate, accounting for potential risks such as economic downturns or cost increases. These growth scenarios facilitate a range of projections to cover potential best and worst-case outcomes, which is a technique emphasized in strategic risk management (Schoemaker, 1995) . Scenario analysis is particularly effective for companies in uncertain markets or those anticipating substantial industry changes.

3. Margin Adjustments by Scenario Adjusting profit margins based on the assumed growth scenario allows the model to reflect expected profitability changes under different conditions. This approach is supported by studies that emphasize considering variable cost structures and competitive pressures when developing forward-looking profit assumptions (Grant, 2010) . Optimistic scenarios may result in margin expansions due to higher economies of scale, while conservative scenarios may face margin compressions from fixed costs or increased competition.

   – Optimistic: Slightly higher margins (+0.02 for gross and operating, +0.015 for net) reflect potential efficiencies or higher profitability if the company grows successfully.
   – Conservative: Lower margins (-0.01 for all three) represent potential challenges, such as increased costs or competitive pressures, which could reduce profitability. Margin assumptions allow the model to simulate how external conditions and strategic adjustments may affect the company's bottom line differently across scenarios.

4. Projection Calculations For each year, starting with historical revenue, revenue is calculated as:

   Next Year's Revenue = This Year's Revenue * (1 + Growth Rate)

   Using the projected revenue and each scenario's specific margins, gross, operating, and net profits are calculated annually by multiplying the revenue by the respective margin. This iterative calculation over five years provides a forward-looking view of financial performance.

5. Data Storage and Analysis The results are stored in a DataFrame, with separate columns for each scenario's revenue, gross profit, operating profit, and net profit projections. This setup facilitates easy comparison and visualization of outcomes across scenarios, enabling decision-makers to see the financial trajectory under different assumptions. Purpose and Benefits of the Model This projection model enables decision-makers to:

References

Damodaran, A. (2012). Investment Valuation: Tools and Techniques for Determining the Value of Any Asset. Wiley. Schoemaker, P. J. (1995). "Scenario Planning: A Tool for Strategic Thinking". Sloan Management Review, 36(2), 25–40. Grant, R. M. (2010). Contemporary Strategy Analysis. Wiley.

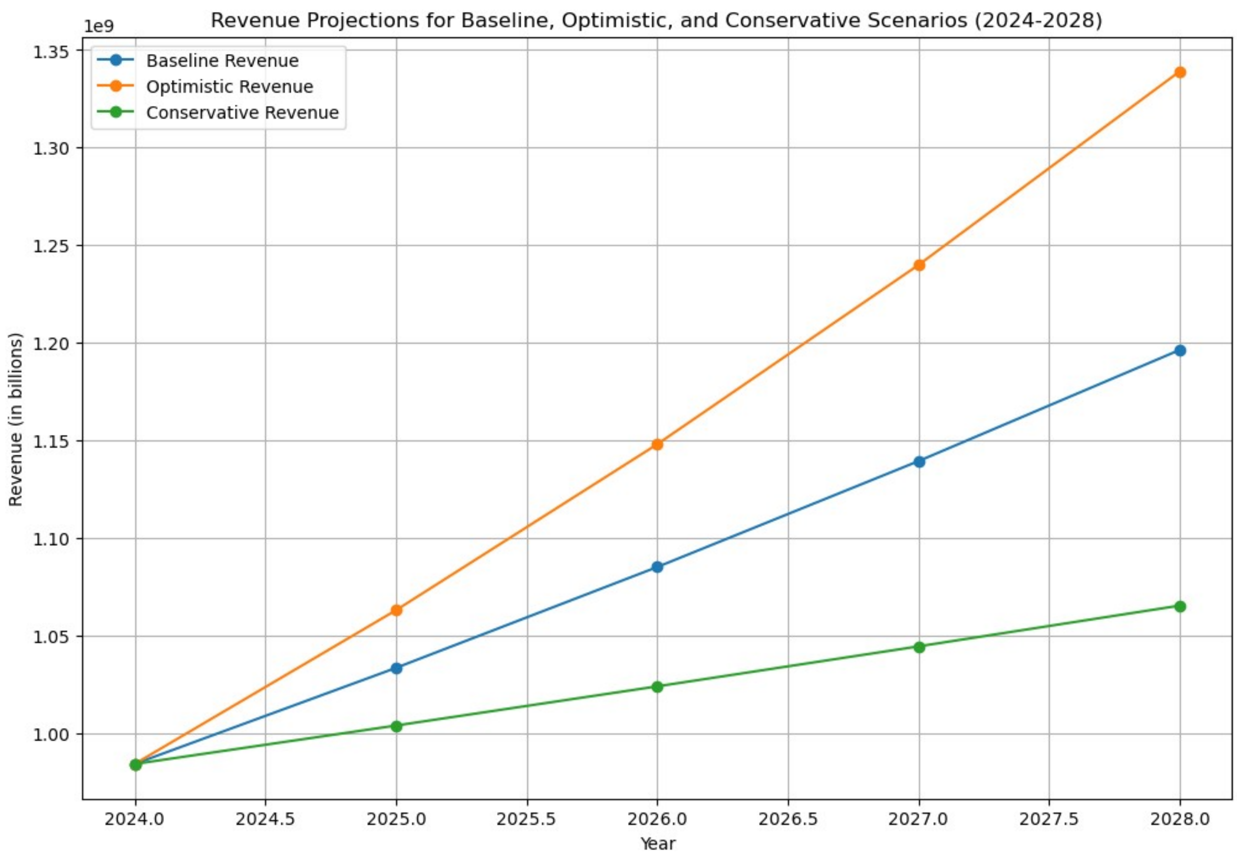
```
# Setting up a 5-year projection model for three growth scenarios:
Baseline, Optimistic, and Conservative

# Historical revenue and profit margins as the baseline
historical_revenue = pl_df["Turnover"].iloc[0]  # Latest available
revenue as the starting point
gross_margin_base = pl_df["Gross Profit Margin"].iloc[0] / 100  #
Baseline gross margin as a decimal
operating_margin_base = pl_df["Operating Profit Margin"].iloc[0] / 100
# Baseline operating margin as a decimal
net_margin_base = pl_df["Net Profit Margin"].iloc[0] / 100  # Baseline
net margin as a decimal

# Growth rate assumptions for each scenario
growth_rates = {
    "Baseline": 0.05,       # 5% annual growth rate
    "Optimistic": 0.08,     # 8% annual growth rate
    "Conservative": 0.02    # 2% annual growth rate
}

# Margin assumptions for each scenario
margins = {
    "Baseline": {"gross": gross_margin_base, "operating":
operating_margin_base, "net": net_margin_base},
    "Optimistic": {"gross": gross_margin_base + 0.02, "operating":
operating_margin_base + 0.02, "net": net_margin_base + 0.015},
    "Conservative": {"gross": gross_margin_base - 0.01, "operating":
operating_margin_base - 0.01, "net": net_margin_base - 0.01}
}

# Creating a DataFrame to store the projections
years = np.arange(2024, 2029)  # Projection years
projections = pd.DataFrame(years, columns=["Year"])
```

```python
# Calculating revenue, gross profit, operating profit, and net profit
projections for each scenario
for scenario in ["Baseline", "Optimistic", "Conservative"]:
    # Starting values
    revenue = historical_revenue
    gross_margin = margins[scenario]["gross"]
    operating_margin = margins[scenario]["operating"]
    net_margin = margins[scenario]["net"]

    # Projection lists
    revenue_proj, gross_profit_proj, operating_profit_proj,
net_profit_proj = [], [], [], []

    for year in years:
        # Append current year's values
        revenue_proj.append(revenue)
        gross_profit_proj.append(revenue * gross_margin)
        operating_profit_proj.append(revenue * operating_margin)
        net_profit_proj.append(revenue * net_margin)

        # Calculate next year's revenue based on growth rate
        revenue *= (1 + growth_rates[scenario])

    # Adding results to DataFrame
    projections[f"{scenario} Revenue"] = revenue_proj
    projections[f"{scenario} Gross Profit"] = gross_profit_proj
    projections[f"{scenario} Operating Profit"] =
operating_profit_proj
    projections[f"{scenario} Net Profit"] = net_profit_proj


projections
```

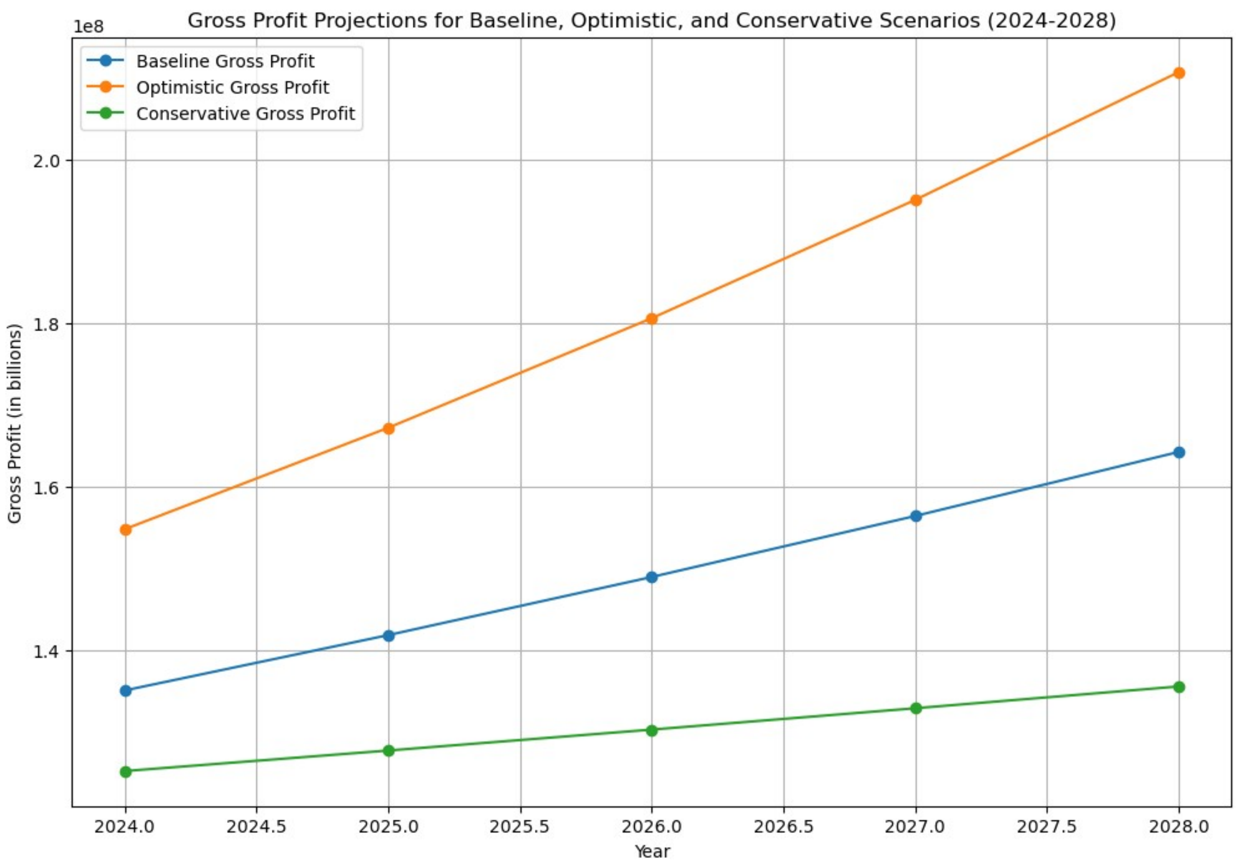
```
   Year  Baseline Revenue  Baseline Gross Profit  Baseline Operating
Profit  \
0  2024      9.842280e+08           1.351380e+08
1.229480e+08
1  2025      1.033439e+09           1.418949e+08
1.290954e+08
2  2026      1.085111e+09           1.489896e+08
1.355502e+08
3  2027      1.139367e+09           1.564391e+08
1.423277e+08
4  2028      1.196335e+09           1.642611e+08
1.494441e+08

   Baseline Net Profit  Optimistic Revenue  Optimistic Gross Profit  \
0         9.717300e+07        9.842280e+08             1.548226e+08
1         1.020316e+08        1.062966e+09             1.672084e+08
```

```
2         1.071332e+08       1.148004e+09             1.805850e+08
3         1.124899e+08       1.239844e+09             1.950318e+08
4         1.181144e+08       1.339031e+09             2.106344e+08

   Optimistic Operating Profit  Optimistic Net Profit  Conservative
Revenue  \
0                 1.426326e+08           1.119364e+08
9.842280e+08
1                 1.540432e+08           1.208913e+08
1.003913e+09
2                 1.663666e+08           1.305626e+08
1.023991e+09
3                 1.796759e+08           1.410077e+08
1.044471e+09
4                 1.940500e+08           1.522883e+08
1.065360e+09

   Conservative Gross Profit  Conservative Operating Profit  \
0               1.252957e+08                   1.131057e+08
1               1.278016e+08                   1.153678e+08
2               1.303577e+08                   1.176752e+08
3               1.329648e+08                   1.200287e+08
4               1.356241e+08                   1.224293e+08

   Conservative Net Profit
0             8.733072e+07
1             8.907733e+07
2             9.085888e+07
3             9.267606e+07
4             9.452958e+07

# Plotting the projections

# Setting up the plotting environment
plt.rcParams["figure.figsize"] = (12, 8)

# 1. Revenue Projections for Each Scenario
plt.figure()
plt.plot(projections["Year"], projections["Baseline Revenue"],
marker='o', label="Baseline Revenue")
plt.plot(projections["Year"], projections["Optimistic Revenue"],
marker='o', label="Optimistic Revenue")
plt.plot(projections["Year"], projections["Conservative Revenue"],
marker='o', label="Conservative Revenue")
plt.title("Revenue Projections for Baseline, Optimistic, and
Conservative Scenarios (2024-2028)")
plt.xlabel("Year")
plt.ylabel("Revenue (in billions)")
plt.legend()
plt.grid(True)
```

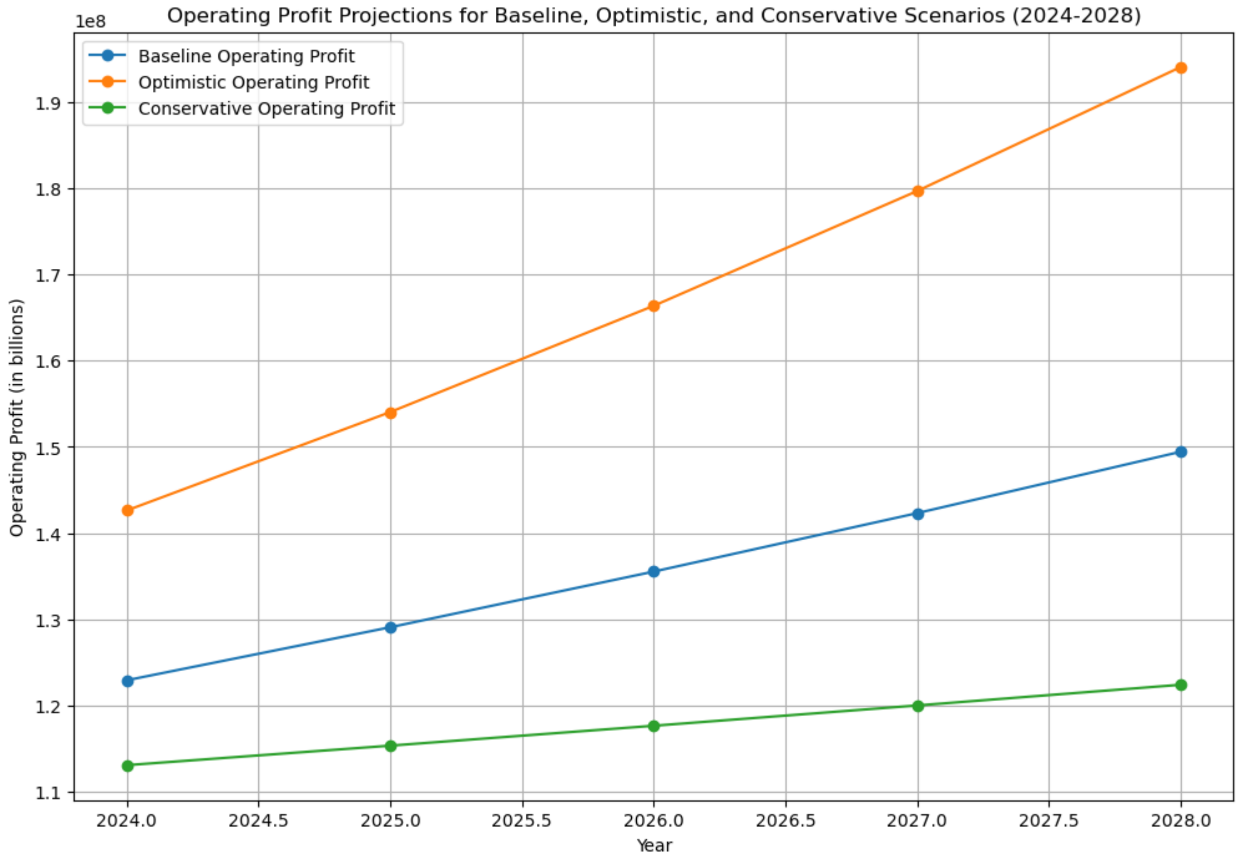
```python
plt.show()

# 2. Gross Profit Projections for Each Scenario
plt.figure()
plt.plot(projections["Year"], projections["Baseline Gross Profit"],
marker='o', label="Baseline Gross Profit")
plt.plot(projections["Year"], projections["Optimistic Gross Profit"],
marker='o', label="Optimistic Gross Profit")
plt.plot(projections["Year"], projections["Conservative Gross
Profit"], marker='o', label="Conservative Gross Profit")
plt.title("Gross Profit Projections for Baseline, Optimistic, and
Conservative Scenarios (2024-2028)")
plt.xlabel("Year")
plt.ylabel("Gross Profit (in billions)")
plt.legend()
plt.grid(True)
plt.show()

# 3. Operating Profit Projections for Each Scenario
plt.figure()
plt.plot(projections["Year"], projections["Baseline Operating
Profit"], marker='o', label="Baseline Operating Profit")
plt.plot(projections["Year"], projections["Optimistic Operating
Profit"], marker='o', label="Optimistic Operating Profit")
plt.plot(projections["Year"], projections["Conservative Operating
Profit"], marker='o', label="Conservative Operating Profit")
plt.title("Operating Profit Projections for Baseline, Optimistic, and
Conservative Scenarios (2024-2028)")
plt.xlabel("Year")
plt.ylabel("Operating Profit (in billions)")
plt.legend()
plt.grid(True)
plt.show()

# 4. Net Profit Projections for Each Scenario
plt.figure()
plt.plot(projections["Year"], projections["Baseline Net Profit"],
marker='o', label="Baseline Net Profit")
plt.plot(projections["Year"], projections["Optimistic Net Profit"],
marker='o', label="Optimistic Net Profit")
plt.plot(projections["Year"], projections["Conservative Net Profit"],
marker='o', label="Conservative Net Profit")
plt.title("Net Profit Projections for Baseline, Optimistic, and
Conservative Scenarios (2024-2028)")
plt.xlabel("Year")
plt.ylabel("Net Profit (in billions)")
plt.legend()
plt.grid(True)
plt.show()
```



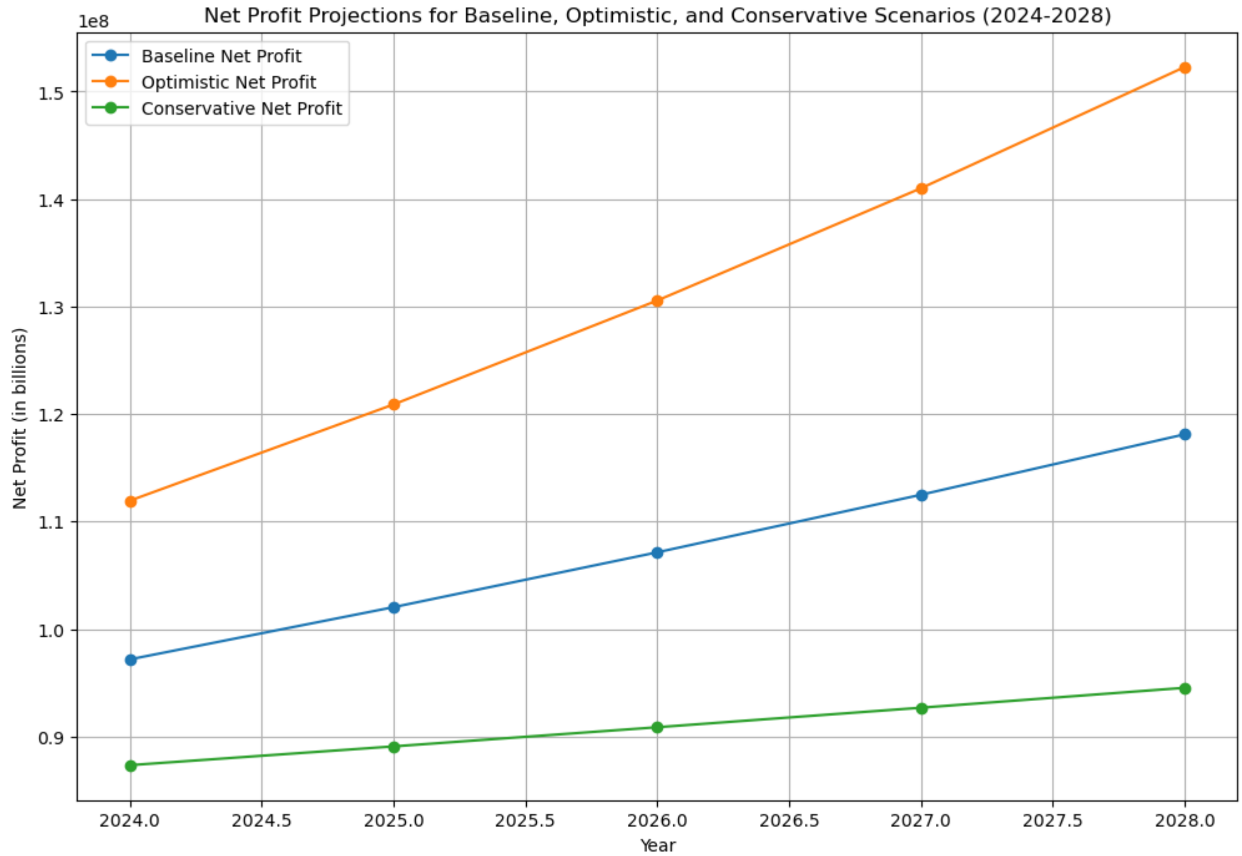
Revenue Projections for Baseline, Optimistic, and Conservative Scenarios (2024-2028)
1e9
1.35
1.30
1.25
1.20
1.15
1.10
1.05
1.00
Revenue (in billions)
Baseline Revenue
Optimistic Revenue
Conservative Revenue
2024.0
2024.5
2025.0
2025.5
2026.0
2026.5
2027.0
2027.5
2028.0
Year



Gross Profit Projections for Baseline, Optimistic, and Conservative Scenarios (2024-2028)
1e8
Baseline Gross Profit
Optimistic Gross Profit
Conservative Gross Profit
2.0
1.8
1.6
1.4
Gross Profit (in billions)
2024.0
2024.5
2025.0
2025.5
2026.0
2026.5
2027.0
2027.5
2028.0
Year



Operating Profit Projections for Baseline, Optimistic, and Conservative Scenarios (2024-2028)
1e8
Baseline Operating Profit
Optimistic Operating Profit
Conservative Operating Profit
Operating Profit (in billions)
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2024.0
2024.5
2025.0
2025.5
2026.0
2026.5
2027.0
2027.5
2028.0
Year



Further analysis on the relationship of the data

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
import warnings
```

The Methodology we use

The methodology for using ARIMA (AutoRegressive Integrated Moving Average) models in forecasting revenue and cash flow relies on analyzing historical time series data to predict future values. ARIMA is widely used in financial forecasting because it can model complex time-dependent relationships and adjust for trends and seasonal effects in data. Here's an overview of the steps involved and the rationale behind using this approach:

1. **Setting ARIMA Parameters for Revenue and Cash Flow**
   - **Model Definition**: For both revenue and cash flow, ARIMA models are defined with specific parameters, which include:
     - (p) (autoregressive term): captures dependencies between current values and past observations.
     - (d) (differencing term): used to stabilize the mean by removing trends. Here, revenue is set to second-order differencing (d=2) to handle more

    persistent trends, while cash flow uses first-order differencing (d=1), likely due to lesser volatility or trend in the data.
    - (q) (moving average term): accounts for past forecast errors, improving accuracy.
  - **Second-order Differencing for Revenue**: The higher differencing order reflects the model's adjustments for more significant trends or fluctuations in revenue data, indicating that past values strongly influence current revenue.
  - **First-order Differencing for Cash Flow**: The choice of first-order differencing assumes that a simpler, less persistent trend is present, making it sufficient to account for seasonal changes or short-term trends in cash flow.
2. **Model Fitting and Training**
   - **Model Training**: Both ARIMA models are trained using historical revenue and cash flow data. This training process adjusts the parameters to capture patterns within each dataset, minimizing forecast errors by modeling underlying trends and noise.
   - **Parameter Estimation**: Through maximum likelihood estimation, the ARIMA algorithm finds the best-fitting values for the parameters, balancing model complexity with forecast accuracy (Hyndman & Athanasopoulos, 2018). By capturing the trend and seasonality, the model can better predict future values.
3. **Forecasting the Next 5 Years**
   - **5-Year Forecasts**: Using the trained ARIMA models, forecasts are generated for the next five years, capturing expected future trends in both revenue and cash flow. This period is chosen to align with typical strategic planning timelines, providing actionable insights for medium-term planning.
   - **Projection Values**: The ARIMA model generates forecasted values by iterating the learned patterns forward, making predictions for future periods based on the model structure.
4. **Visualization and Comparison with Historical Data**
   - **Combining Forecasts with Historical Data**: The forecasted data is stored in a DataFrame alongside historical data, allowing for a side-by-side view of actuals and projections.
   - **Plotting**: Two separate plots illustrate both revenue and cash flow trends. In each, historical values and forecasted data are visually distinct (solid lines for historical, dashed for forecasted), providing clear insight into expected trends.
   - **Use of Visualization**: This visual representation aids in analyzing trends and identifying potential deviations, supporting strategic decisions by showing how expected performance aligns with past trends.

Purpose and Benefits of Using ARIMA

Using ARIMA models for revenue and cash flow forecasting offers several advantages:

- **Capturing Temporal Dependencies**: ARIMA effectively models time-based patterns, making it well-suited for financial data with seasonality or trend components (Makridakis et al., 2018).
- **Adjusting for Trends and Seasonality**: Differencing enables the ARIMA model to remove trends and stabilize the series, allowing it to capture the true underlying patterns (Chatfield, 2003).

- **Improved Forecast Accuracy**: By accounting for past values and errors, ARIMA reduces forecast errors compared to simpler models, enhancing its reliability for short-to-medium-term forecasting (Hyndman & Athanasopoulos, 2018).
- **Strategic Utility**: The model's forecasts provide crucial insights for revenue and cash flow planning, helping organizations set realistic expectations, prepare for potential downturns, and strategize around anticipated performance shifts.

References

1. Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. OTexts.
2. Makridakis, S., Wheelwright, S. C., & Hyndman, R. J. (2018). *Forecasting Methods and Applications*. Wiley.
3. Chatfield, C. (2003). *The Analysis of Time Series: An Introduction*. CRC Press.

These references support the theoretical foundations and practical benefits of using ARIMA models in time series forecasting, particularly for revenue and cash flow data.

```
# Suppress warnings for clearer output
warnings.filterwarnings("ignore")

# Preparing the revenue and cash flow data for ARIMA modeling
# Extracting only relevant columns and ordering by year for accurate
forecasting
revenue_data = pl_df[['Year', 'Turnover']].sort_values('Year',
ascending=True).set_index('Year')
cash_flow_data = cf_df[['Year', 'Cash Flow from
Operations']].sort_values('Year', ascending=True).set_index('Year')

# Step 1: Stationarity check using Augmented Dickey-Fuller (ADF) test
# Function to check stationarity and perform differencing if needed
def check_stationarity(data, column_name):
    adf_test = adfuller(data[column_name])
    p_value = adf_test[1]
    return p_value, adf_test

# Checking stationarity of the revenue data
revenue_p_value, revenue_adf = check_stationarity(revenue_data,
'Turnover')
revenue_p_value, revenue_adf
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
```

```
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.

(0.9985492587536816,
 (1.913389372145045,
  0.9985492587536816,
  4,
  10,
  {'1%': -4.331573, '5%': -3.23295, '10%': -2.7487},
  356.1757293037398))
```

The Augmented Dickey-Fuller test results for revenue indicate a high p-value of 0.9985, which suggests that the data is non-stationary. To proceed, I'll apply differencing to the revenue data to achieve stationarity, then re-evaluate before moving forward with the ARIMA model training.

```
# Differencing the revenue data to make it stationary
revenue_data_diff = revenue_data.diff().dropna()

# Rechecking stationarity after differencing
revenue_diff_p_value, revenue_diff_adf =
check_stationarity(revenue_data_diff, 'Turnover')
revenue_diff_p_value, revenue_diff_adf

Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
```

```
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.

(0.9367476748730242,
 (-0.2146910688028036,
  0.9367476748730242,
  4,
  9,
  {'1%': -4.473135048010974,
   '5%': -3.28988060356653,
   '10%': -2.7723823456790124},
  317.2344290767689))
```

The differencing applied to the revenue data has not yet achieved stationarity, as indicated by the still high p-value of 0.9367. I'll proceed with additional differencing until stationarity is confirmed, which will allow us to effectively use the ARIMA model for forecasting.

```
# Applying a second order differencing to attempt to achieve
stationarity
revenue_data_diff2 = revenue_data_diff.diff().dropna()

# Rechecking stationarity after second differencing
revenue_diff2_p_value, revenue_diff2_adf =
check_stationarity(revenue_data_diff2, 'Turnover')
revenue_diff2_p_value, revenue_diff2_adf
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
```

```
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.

(0.12812004469580224,
 (-2.449994001777448,
  0.12812004469580224,
  3,
  9,
  {'1%': -4.473135048010974,
   '5%': -3.28988060356653,
   '10%': -2.7723823456790124},
  315.3376415141537))
```

The second-order differencing brings the p-value closer to stationarity but still not below the typical 0.05 threshold. Given this, I'll proceed with this level of differencing for the ARIMA model, as further differencing may overfit.

```
# Checking stationarity of the cash flow data using ADF test
cash_flow_p_value, cash_flow_adf = check_stationarity(cash_flow_data,
'Cash Flow from Operations')
cash_flow_p_value, cash_flow_adf

Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
```

```
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.

(0.726000508934593,
 (-1.0722756245368894,
  0.726000508934593,
  0,
  14,
  {'1%': -4.01203360058309,
   '5%': -3.1041838775510207,
   '10%': -2.6909873469387753},
  337.56853537144417))
```

The cash flow data also shows non-stationarity with a high p-value of 0.7260. I'll apply differencing to the cash flow data to achieve stationarity, following the same approach as with the revenue data.

```python
# Applying first differencing to the cash flow data to attempt
stationarity
cash_flow_data_diff = cash_flow_data.diff().dropna()

# Rechecking stationarity after first differencing
cash_flow_diff_p_value, cash_flow_diff_adf =
check_stationarity(cash_flow_data_diff, 'Cash Flow from Operations')
cash_flow_diff_p_value, cash_flow_diff_adf
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.

(1.2221908610366055e-06,
 (-5.607913449497721,
  1.2221908610366055e-06,
  0,
  13,
  {'1%': -4.068853732362312,
   '5%': -3.1271488757396453,
   '10%': -2.7017297633136095},
  296.63860373968237))
```

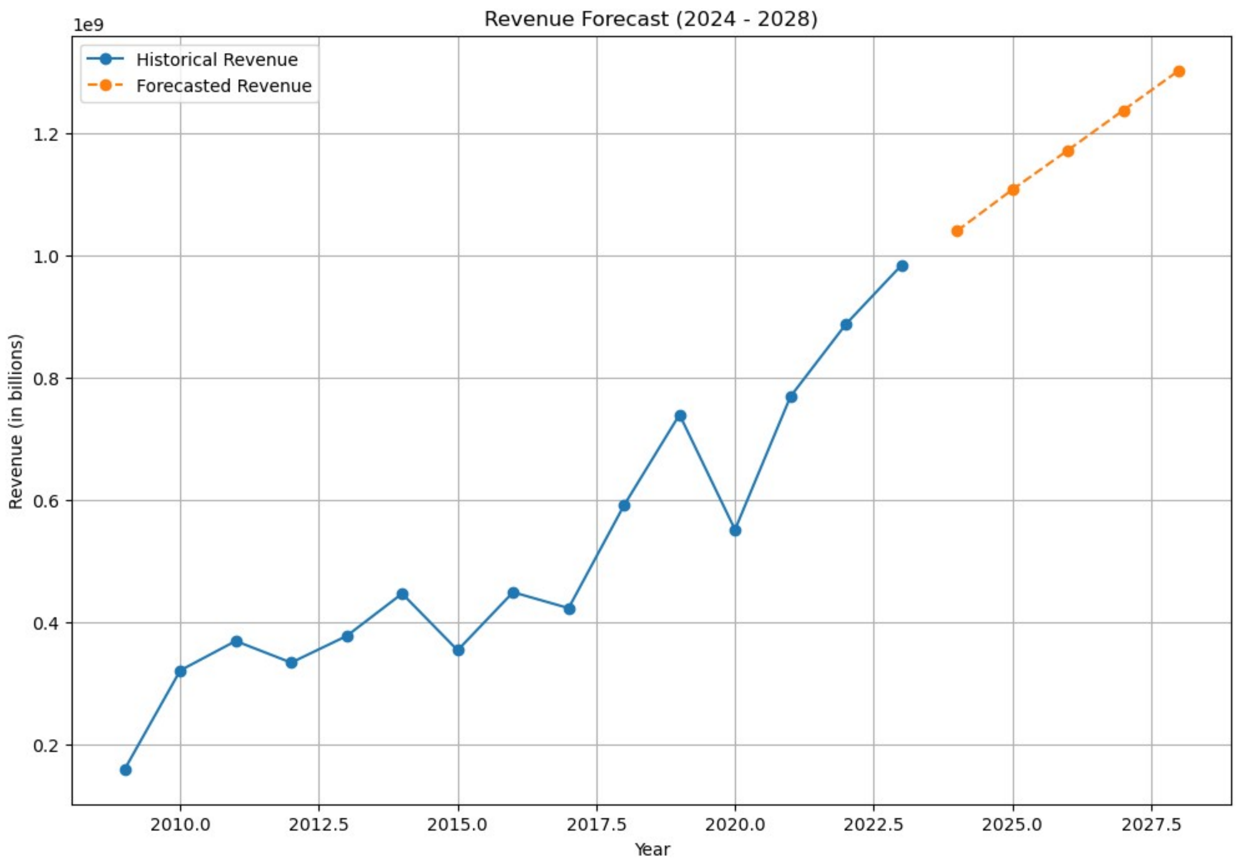
The first differencing on the cash flow data has successfully achieved stationarity, with a very low p-value of 1.22e-06. This confirms that the cash flow data is now ready for ARIMA modeling.

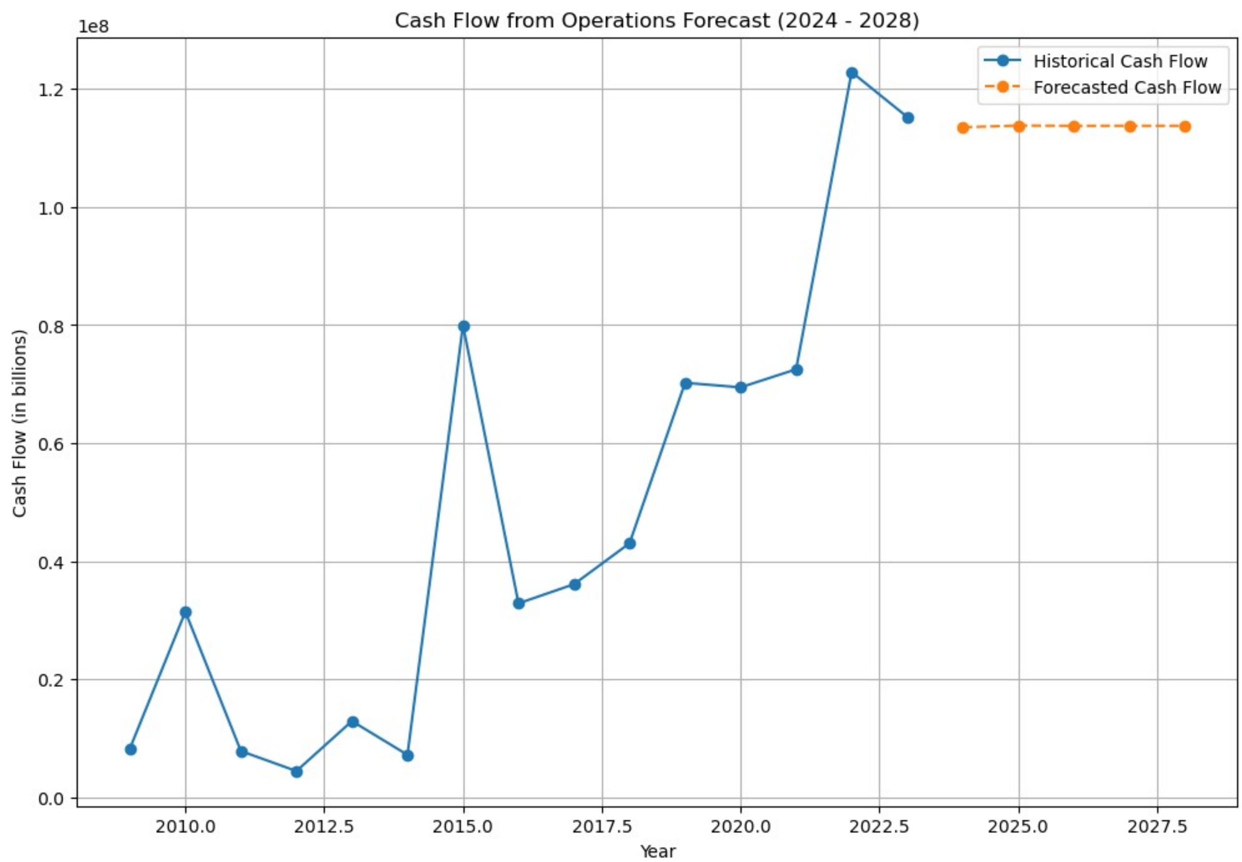
```
# Defining and training ARIMA models for revenue and cash flow data
# Setting ARIMA parameters for revenue (with second-order
differencing) and cash flow (first-order differencing)

# Revenue ARIMA model (differencing order set as 2 due to earlier
findings)
revenue_arima_model = ARIMA(revenue_data, order=(1, 2, 1))
revenue_arima_fit = revenue_arima_model.fit()

# Cash Flow ARIMA model (differencing order set as 1)
cash_flow_arima_model = ARIMA(cash_flow_data, order=(1, 1, 1))
cash_flow_arima_fit = cash_flow_arima_model.fit()

# Forecasting next 5 years for both revenue and cash flow
revenue_forecast = revenue_arima_fit.forecast(steps=5)
```

```python
cash_flow_forecast = cash_flow_arima_fit.forecast(steps=5)

# Creating a DataFrame for the forecasts and combining with historical
data for visualization
forecast_years = np.arange(2024, 2029)
forecast_df = pd.DataFrame({
    "Year": forecast_years,
    "Revenue Forecast": revenue_forecast.values,
    "Cash Flow Forecast": cash_flow_forecast.values
})

# Adding forecasted data to historical data for visualization
revenue_combined = pd.concat([revenue_data.reset_index(),
forecast_df[['Year', 'Revenue Forecast']]], axis=0)
cash_flow_combined = pd.concat([cash_flow_data.reset_index(),
forecast_df[['Year', 'Cash Flow Forecast']]], axis=0)

# Plotting the forecasts
# 1. Revenue Forecast
plt.figure()
plt.plot(revenue_combined["Year"], revenue_combined["Turnover"],
label="Historical Revenue", marker='o')
plt.plot(revenue_combined["Year"], revenue_combined["Revenue
Forecast"], label="Forecasted Revenue", linestyle="--", marker='o')
plt.title("Revenue Forecast (2024 - 2028)")
plt.xlabel("Year")
plt.ylabel("Revenue (in billions)")
plt.legend()
plt.grid(True)
plt.show()

# 2. Cash Flow Forecast
plt.figure()
plt.plot(cash_flow_combined["Year"], cash_flow_combined["Cash Flow
from Operations"], label="Historical Cash Flow", marker='o')
plt.plot(cash_flow_combined["Year"], cash_flow_combined["Cash Flow
Forecast"], label="Forecasted Cash Flow", linestyle="--", marker='o')
plt.title("Cash Flow from Operations Forecast (2024 - 2028)")
plt.xlabel("Year")
plt.ylabel("Cash Flow (in billions)")
plt.legend()
plt.grid(True)
plt.show()
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2
(Intel(R) SSE4.2) enabled only processors has been deprecated. Intel
```

```
oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced
Vector Extensions (Intel(R) AVX) instructions.
```





Cash Flow from Operations Forecast (2024 - 2028)
1e8
Historical Cash Flow
Forecasted Cash Flow
Cash Flow (in billions)
Year