Informative Analysis (Part 4)

Analytical Process 1: Monthly Revenue Analysis by Subscription Plan

Objective

The objective of this analytical process is to understand how monthly revenue is distributed across different subscription plans over time. By analyzing revenue at the plan level, this process helps identify which plans contribute most to overall revenue growth and how customer preferences evolve across months. This type of analysis is typical in OLAP systems for reporting and strategic decision-making.

Data Preparation

This analysis uses data stored in an OLAP-style schema consisting of a fact table and multiple dimension tables. The main data source is the Fact_Monthly_Revenue table, which records revenue transactions at the plan and date level. This fact table is joined with:

- Dim_Plan to obtain human-readable subscription plan names
- Dim_Date to convert surrogate date keys into actual calendar dates

Using Spark SQL (simulated here through SQL queries), the data is filtered to include records between December 2024 and December 2025. Revenue values are aggregated by month and subscription plan, which is a common OLAP operation (GROUP BY on time and product dimensions).

To ensure a consistent time axis for analysis and visualization, missing months are explicitly added to the dataset. This avoids gaps in the timeline and makes trends easier to interpret.

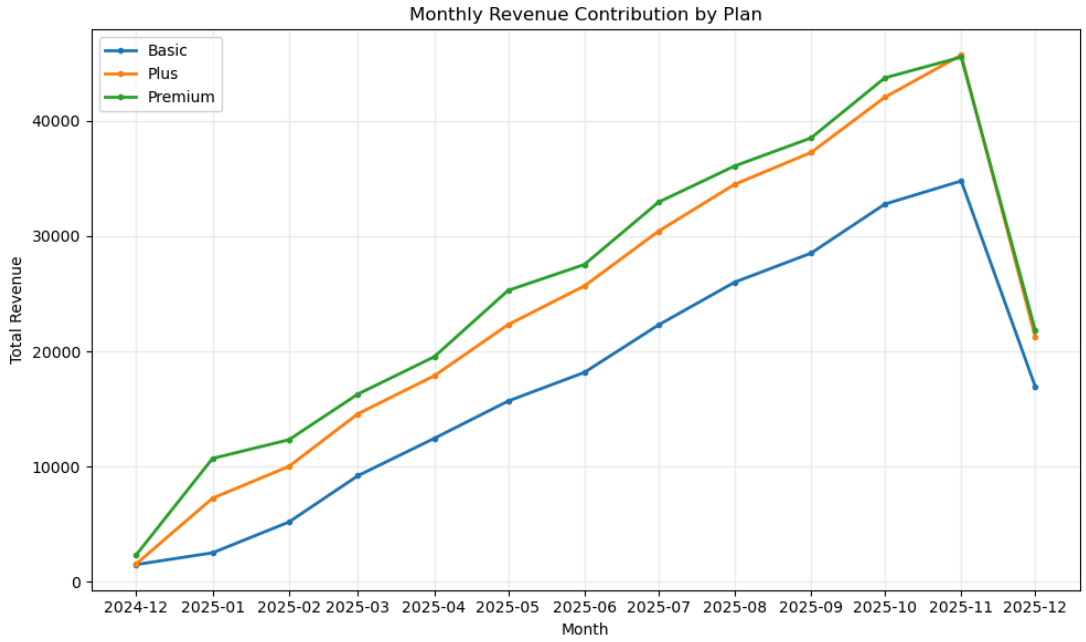
Analytical Workflow

The analytical workflow consists of the following steps:

1. Query monthly revenue data by joining fact and dimension tables
2. Aggregate total revenue by month and subscription plan

3. Standardize the monthly timeline to ensure continuity
4. Prepare the data for comparison across plans

Visualization



Analytical Process 2: Daily Utilization Rate Analysis and Forecasting

Objective

The objective of this analytical process is to analyze daily utilization rates over time and forecast short-term future utilization trends. Utilization rate reflects how efficiently resources (such as classes, facilities, or sessions) are being used. Understanding its temporal pattern helps support capacity planning and operational decision-making.

Data Preparation

This analysis is based on an OLAP-style fact table Fact_Class_Attendance, which records utilization metrics at the date level. The fact table is joined with the Dim_Date table to retrieve actual calendar dates from surrogate keys.

The dataset covers daily utilization rates from December 2024 to mid-December 2025. Since utilization data may contain missing dates and incomplete observations, several preprocessing steps are applied to ensure data quality and continuity:

- Multiple records for the same date are aggregated using the daily mean utilization rate
- Missing dates are explicitly added to create a continuous daily time series
- Missing utilization values are filled using time-based linear interpolation

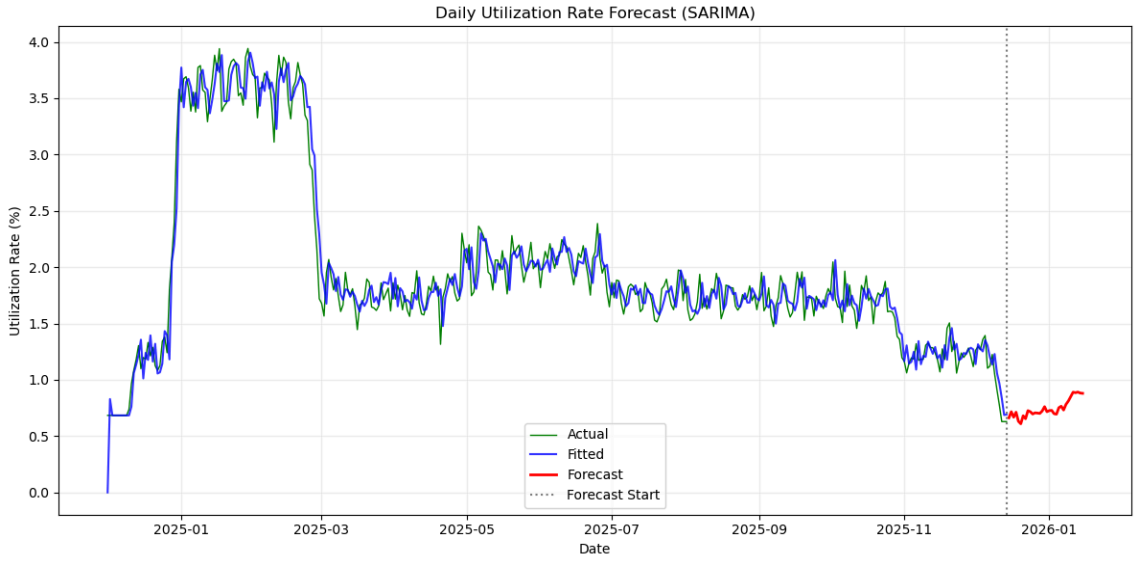
Analytical Workflow

The analytical workflow includes the following steps:

1. Retrieve daily utilization data using SQL joins between fact and date dimensions
2. Aggregate utilization rates at the daily level
3. Enforce continuity in the time dimension by filling missing dates
4. Apply time-based interpolation to handle missing values
5. Train a Seasonal ARIMA (SARIMA) model on the cleaned daily time series
6. Generate both in-sample fitted values and short-term forecasts

A SARIMA model is selected because it can capture both trend and seasonal patterns in time series data. A monthly seasonal component is included to reflect recurring utilization behavior across time.

Visualization



Analytical Process 3: Daily Revenue Trend Analysis and Forecasting

Objective

The objective of this analytical process is to analyze daily revenue patterns and forecast short-term future revenue trends. Daily revenue is a core business metric that reflects customer activity and overall financial performance. By modeling revenue as a time series, this process supports short-term financial planning and trend monitoring.

Data Preparation

This analysis is based on the Fact_Monthly_Revenue fact table, which records revenue information linked to date keys. The fact table is joined with the Dim_Date table to obtain actual calendar dates.

Although the original fact table records revenue at a transactional level, the analysis focuses on daily total revenue. Several preprocessing steps are applied to transform the raw OLAP data into a clean and continuous time series suitable for forecasting:

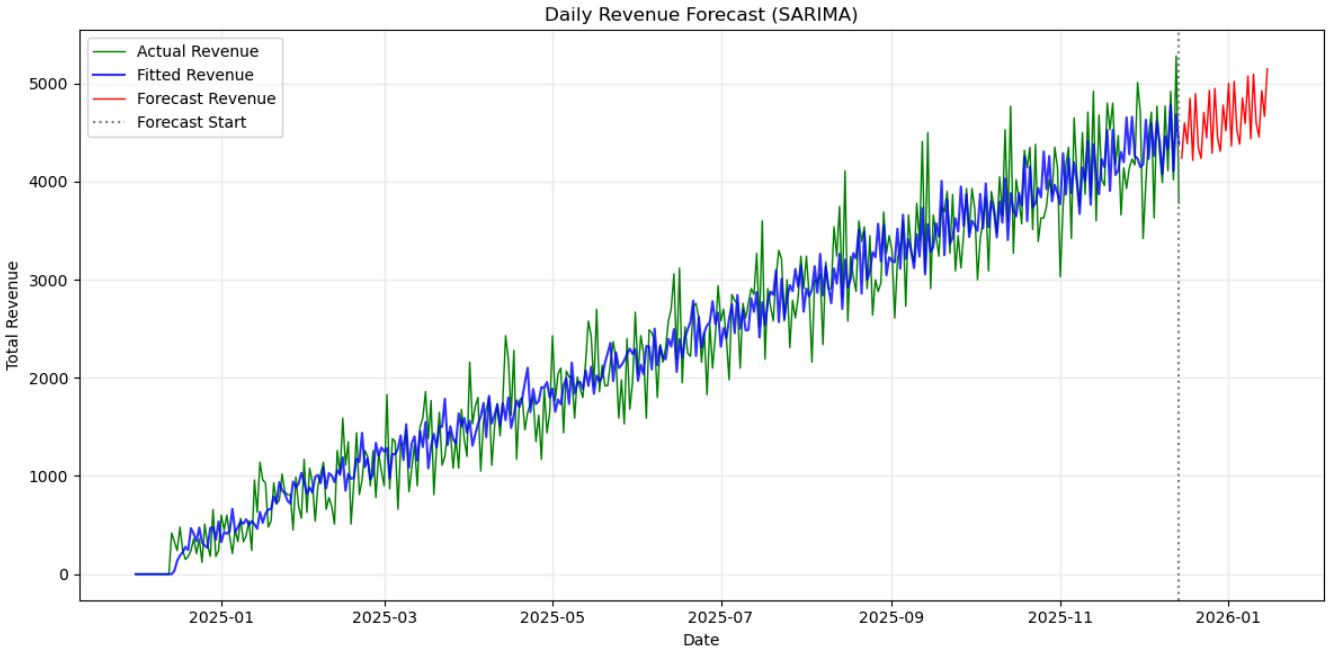
- Revenue values are aggregated by day using summation, which is appropriate for flow-based financial metrics
- Missing dates are added to ensure a continuous daily timeline
- Missing revenue values are filled using time-based linear interpolation

Analytical Workflow

The analytical workflow consists of the following steps:

1. Retrieve daily revenue data using SQL joins between fact and date dimensions
2. Aggregate revenue at the daily level
3. Enforce continuity in the time dimension by filling missing dates
4. Apply time-based interpolation to handle missing revenue values
5. Train a Seasonal ARIMA (SARIMA) model to capture trend and weekly seasonality
6. Generate in-sample fitted values and short-term revenue forecasts

Visualization


Daily Revenue Forecast (SARIMA)
Actual Revenue
Fitted Revenue
Forecast Revenue
Forecast Start
Total Revenue
0
1000
2000
3000
4000
5000
2025-01
2025-03
2025-05
2025-07
2025-09
2025-11
2026-01
Date

Appendix

Informative Analysis 1

```
import pandas as pd
import numpy as np
from sqlalchemy import create_engine
import urllib
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX



server = 'sql-fitflow-team8-2025.database.windows.net'
database = 'fitflow_olap'
username = 'fitflowadmin'
password = 'Kergerisgreat2025!'

params = urllib.parse.quote_plus(
   f'Driver={{ODBC Driver 18 for SQL Server}};'
   f'Server=tcp:{server},1433;Database={database};'
   f'Uid={username};Pwd={password};Encrypt=yes;TrustServerCertificate=no;'
)
engine = create_engine(f'mssql+pyodbc:///?odbc_connect={params}')

sql1 = """select CONVERT(char(7), full_date, 120) as Date, total_revenue, plan_namefrom
       Fact_Monthly_Revenue as f join Dim_Plan as p on f.plan_key = p.plan_key right join
       Dim_Date as d on f.date_key = d.date_key
      where full_date >= '2024-12-01' and full_date <= '2025-12-31' """
df1 = pd.read_sql(sql1, engine)
df1 = df1[~df1.isna()]
df1 = df1.dropna()
total_rev_per_mon = df1.groupby(['Date','plan_name'])['total_revenue'].sum().reset_index()
total_rev_per_mon['total_revenue'] = round(total_rev_per_mon['total_revenue'],2)
import matplotlib.pyplot as plt

total_rev_per_mon["Date"] = pd.to_datetime(total_rev_per_mon["Date"])
full_months = pd.date_range(
   start="2024-12-01",
   end="2025-12-01",
```

```
    freq="MS"
)
plans = total_rev_per_mon["plan_name"].unique()

full_df = (
    total_rev_per_mon
    .set_index(["Date", "plan_name"])
    .unstack("plan_name")
    .reindex(full_months)
    .stack("plan_name")
    .reset_index()
)

full_df.columns = ["Date", "plan_name", "total_revenue"]

plt.figure(figsize=(10, 6))

for plan in plans:
    tmp = full_df[full_df["plan_name"] == plan]
    plt.plot(
        tmp["Date"],
        tmp["total_revenue"],
        marker=".",
        linewidth=2,
        label=plan
    )

plt.xticks(full_months, [d.strftime("%Y-%m") for d in full_months])
plt.xlabel("Month")
plt.ylabel("Total Revenue")
plt.title("Monthly Revenue Contribution by Plan")
plt.legend()
plt.grid(alpha=0.2)
plt.tight_layout()
plt.show()
```

Informative Analysis 2

```
sql2 = """select full_date as Date, utilization_rate
from Fact_Class_Attendance f right join Dim_Date d on f.date_key = d.date_key
```

```
where full_date >= '2024-12-01' and full_date <= '2025-12-14' """
df2 = pd.read_sql(sql2, engine)
df2['utilization_rate'] = df2['utilization_rate']* 100

df2["Date"] = pd.to_datetime(df2["Date"])

df2 = (
   df2
   .groupby("Date", as_index=False)
   .mean()
)

df2 = df2.set_index("Date").sort_index()

full_dates = pd.date_range(
   start=df2.index.min(),
   end=df2.index.max(),
   freq="D"
)

df2 = df2.reindex(full_dates)

df2["utilization_rate"] = df2["utilization_rate"].interpolate(
   method="time"
)

df2["utilization_rate"] = df2["utilization_rate"].ffill().bfill().round(4)

df2 = df2.reset_index().rename(columns={"index": "Date"})

df2
# Ensure datetime and set index
df = df2.copy()
df["Date"] = pd.to_datetime(df["Date"])
df = df.set_index("Date").sort_index()

y = df["utilization_rate"]

model = SARIMAX(
   y,
   order=(1, 1, 1),            # ARIMA part
   seasonal_order=(1, 0, 1, 30),# Monthly seasonality
```

```
    enforce_stationarity=False,
    enforce_invertibility=False
)

result = model.fit(disp=False)

# In-sample fitted values
fitted = result.fittedvalues

# Forecast to 2025-12-31
forecast_horizon = pd.date_range(
    start="2025-12-15",
    end="2026-1-15",
    freq="D"
)

forecast = result.get_forecast(steps=len(forecast_horizon))
forecast_mean = forecast.predicted_mean

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

# Actual values
plt.plot(
    y.index, y,
    label="Actual",
    color="green",
    linewidth=1
)

# Fitted values
plt.plot(
    fitted.index, fitted,
    label="Fitted",
    color="blue",
    linestyle="solid",
    alpha=0.8
)

# Forecast values
plt.plot(
    forecast_horizon, forecast_mean,
    label="Forecast",
```

```
    color="red",
    linewidth=2
)

plt.axvline(
    pd.to_datetime("2025-12-14"),
    color="gray",
    linestyle=":",
    label="Forecast Start"
)

plt.title("Daily Utilization Rate Forecast (SARIMA)")
plt.xlabel("Date")
plt.ylabel("Utilization Rate (%)")
plt.legend()
plt.grid(alpha=0.2)
plt.tight_layout()
plt.show()
```

Informative Analysis 3

```
sql3 = """select full_date as Date, total_revenue
from Fact_Monthly_Revenue f right join Dim_Date d on f.date_key = d.date_key
where full_date >= '2024-12-01' and full_date <= '2025-12-14'"""

df_rev = pd.read_sql(sql3, engine)

# 1. Convert Date to datetime
df_rev["Date"] = pd.to_datetime(df_rev["Date"])

# 2. Aggregate revenue by day (Date must be unique)
df_rev = (
    df_rev
    .groupby("Date", as_index=False)
    .sum()    # revenue 用 sum 是业务上正确的
)

# 3. Set Date as index and sort
df_rev = df_rev.set_index("Date").sort_index()

# 4. Create full daily date range
```

```
full_dates = pd.date_range(
    start=df_rev.index.min(),
    end=df_rev.index.max(),
    freq="D"
)

# 5. Reindex to make dates continuous
df_rev = df_rev.reindex(full_dates)

# 6. Time-based linear interpolation
df_rev["total_revenue"] = df_rev["total_revenue"].interpolate(
    method="time"
)

# 7. Handle edge NaNs if any
df_rev["total_revenue"] = df_rev["total_revenue"].ffill().bfill()

# 8. Restore Date column
df_rev = df_rev.reset_index().rename(columns={"index": "Date"})

# Copy and prepare data
df = df_rev.copy()
df["Date"] = pd.to_datetime(df["Date"])
df = df.set_index("Date").sort_index()

y = df["total_revenue"]
# y_log = np.log1p(y)
model = SARIMAX(
    y,
    order=(2, 0, 2),
    seasonal_order=(1, 1, 1, 7),
    enforce_stationarity=False,
    enforce_invertibility=False
)

result = model.fit(disp=False)

# In-sample fitted values
fitted = result.fittedvalues

# Forecast horizon
forecast_horizon = pd.date_range(
```

```
    start="2025-12-15",
    end="2026-01-15",
    freq="D"
)

forecast = result.get_forecast(steps=len(forecast_horizon))
forecast_mean = forecast.predicted_mean

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

# Actual revenue
plt.plot(
    y.index, y,
    label="Actual Revenue",
    color="green",
    linewidth=1
)

# Fitted revenue
plt.plot(
    fitted.index, fitted,
    label="Fitted Revenue",
    color="blue",
    linestyle="solid",
    alpha=0.8
)

# Forecast revenue
plt.plot(
    forecast_horizon, forecast_mean,
    label="Forecast Revenue",
    color="red",
    linewidth=1
)

# Forecast start marker
plt.axvline(
    pd.to_datetime("2025-12-14"),
    color="gray",
    linestyle=":",
    label="Forecast Start"
)
```

```
plt.title("Daily Revenue Forecast (SARIMA)")
plt.xlabel("Date")
plt.ylabel("Total Revenue")
plt.legend()
plt.grid(alpha=0.2)
plt.tight_layout()
plt.show()
```