# Notebook summary (high-level steps)

1. **Environment & imports** — standard Kaggle/Python setup: `pandas`, `numpy`, `matplotlib`, `seaborn`, `fbprophet/prophet`, `xgboost`, `sklearn`, `pathlib`, `datetime`.

2. **File listing & paths** — sets `base_path = "/kaggle/input/"` and inspects files; reads CSVs into DataFrames (`claims`, `prescribers`, `market`, `calendar`).

3. **Initial EDA** — `df.head()`, `df.info()`, `df.describe()`, `isnull().sum()` checks. Ensures date columns parsed with `pd.to_datetime`.

4. **Preprocessing** — normalize column names, drop/rename columns, handle missing values, cast dtypes (e.g., `astype(str)`), fix NDC/drug mapping by merging with `market`.

5. **Claim timeline engineering** — sort by `(patient_id, start_date)`, compute `previous_end`, `gap_days` = difference between consecutive fills; `new_start` flagged when gap > 180.

6. **Discontinuation & switches** — compute `last_fill`, flag `discontinued` if last end_date > 60 days before `today`; detect molecule switches by shifting molecule per patient and comparing.

7. **Aggregation to quarterly** — create `year` + `fiscal_quarter` from calendar table, groupby `(patient_id, year, fiscal_quarter)` to produce features: `total_fills`, `avg_gap_days`, `fill_count`, `dos`, `adherence = days_supply/90`, `last_med_days_count`, etc.

8. **One-hot & mapping** — one-hot encode `specialty` and `region`, and add static/int columns if needed. Ensure IDs are excluded from model features.

9. **Forecasting with Prophet** — aggregate monthly fills (`ds`, `y`), build Prophet with multiplicative seasonality, forecast 12 months, and compute MAPE post-Jan-2023.

10. **Train/test + XGBoost** — create target `next_quarter_refill` (1 if patient has a fill next quarter), split (`train_test_split`), train `XGBClassifier`, evaluate accuracy, ROC/AUC, and extract feature importances.

11. **Plots & diagnostics** — histograms for therapy duration, bar charts for unique drugs, boxplots for metrics, feature importance bar chart, pie chart for refill distribution, forecast vs actual with confidence band.

12. **Outputs** — save final tables/plots as CSV/PNG and (optionally) model artifacts.

---

# Pandas DataFrame cheat-sheet (SQL/Snowflake style, compact & readable)

> Note: left column = SQL-style intent; right column = pandas code (Snowflake-esque mental mapping).

**SELECT columns**
SQL: `SELECT col1, col2 FROM claims`
pandas:

```
df[['patient_id','start_date','drug_name']]
```

1.

**FILTER / WHERE**
SQL: `WHERE region='South' AND year=2022`
pandas:

```
df[(df.region=='South') & (df.year==2022)]
```

2.

**LEFT / INNER / RIGHT JOIN**
SQL: `FROM claims c LEFT JOIN prescribers p ON c.provider_id=p.hcp_id`
pandas:

```
merged = claims.merge(prescribers, left_on='provider_id',
right_on='hcp_id', how='left')
# inner: how='inner', right: how='right'
```

3.

**GROUP BY + AGG (aggregations)**
SQL: `GROUP BY patient_id, fiscal_quarter`
pandas:

```
agg = df.groupby(['patient_id','year','fiscal_quarter']).agg(
    total_fills=('patient_id','count'),
    avg_gap_days=('gap_days','mean'),
    days_supply_sum=('days_supply','sum')
).reset_index()
```

4.

**HAVING (post-agg filter)**
SQL: `HAVING count(*) > 1`
pandas:

```
g = df.groupby('patient_id').size().reset_index(name='n')
g[g.n > 1]
```

5.

**ORDER BY / SORT**
SQL: `ORDER BY start_date DESC`
pandas:

```
df.sort_values(['patient_id','start_date'], ascending=[True, False])
```

6.

**ROW_NUMBER / WINDOW (lag/lead/rolling)**
SQL: `LAG(end_date) OVER (PARTITION BY patient_id ORDER BY start_date)`
pandas:

```
df = df.sort_values(['patient_id','start_date'])
df['previous_end'] = df.groupby('patient_id')['end_date'].shift(1)
df['gap_days'] = (df['start_date'] - df['previous_end']).dt.days
```

7.

**CASE WHEN / conditional column**
SQL: `CASE WHEN gap > 180 THEN 1 ELSE 0 END as new_start`

pandas:

```
df['new_start'] = (df['gap_days'] > 180).astype(int)
```

8.

**DATE PARTS (year, quarter, month)**
SQL: `EXTRACT(QUARTER FROM date)`
pandas:

```
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['fiscal_quarter'] = df['date'].dt.quarter
df['month'] = df['date'].dt.month
```

9.

**PIVOT / CROSSTAB**
SQL: `PIVOT` (counts per month)
pandas:

```
pivot = df.pivot_table(index='patient_id', columns='year',
values='fill_count', aggfunc='sum').fillna(0)
# or pd.crosstab(df.patient_id, df.year, values=df.fill_count,
aggfunc='sum')
```

10.

**MELT (UNPIVOT)**
pandas:

```
melted = pd.melt(df, id_vars=['patient_id'],
value_vars=['region_North','region_South'], var_name='region',
value_name='flag')
```

11.

**ONE-HOT / DUMMY Encode**
pandas:

```
df = pd.get_dummies(df, columns=['specialty','region'],
prefix_sep='_')
```

12.

**MERGE + dedupe (snowflake-like left then group)**
pandas:

```
merged = df1.merge(df2, left_on='ndc', right_on='ndc', how='left')
merged =
merged.drop_duplicates(subset=['patient_id','start_date','ndc'])
```

13.

**CREATE TARGET next-quarter refill**
(SQL logic: check if any fill in next quarter per patient)
pandas approach (one-liner logic outline):

```
# assume quarterly_df grouped by patient+year+quarter with 'fills'
quarterly_df['next_q_has_fill'] =
quarterly_df.groupby('patient_id')['fills'].shift(-1).fillna(0).astype
(int) > 0
quarterly_df['target'] = quarterly_df['next_q_has_fill'].astype(int)
```

14.

**SPLIT train/test and standard ML flow**
pandas → sklearn:

```
from sklearn.model_selection import train_test_split
X = df.drop(columns=['patient_id','target'])
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

15.

**XGBoost fit (classifier)**

```
import xgboost as xgb
model = xgb.XGBClassifier(n_estimators=100, max_depth=6,
learning_rate=0.1, use_label_encoder=False, eval_metric='logloss')
model.fit(X_train, y_train)
preds = model.predict(X_test)
```

16.

**Feature importance (built-in)**

```
importances = pd.Series(model.feature_importances_,
index=X_train.columns).sort_values(ascending=False)
top10 = importances.head(10)
```

17.

**Prophet timeseries (monthly forecast)**

```
monthly =
df_monthly.reset_index().rename(columns={'month_start':'ds','fill_coun
t':'y'})
from prophet import Prophet
m = Prophet(seasonality_mode='multiplicative',
yearly_seasonality=True)
m.fit(monthly)
future = m.make_future_dataframe(periods=12, freq='M')
forecast = m.predict(future)
```

18.

**MAPE calculation**

```
def mape(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

19.

**SAVE outputs**

```
df.to_csv('outputs/quarterly_features.csv', index=False)
plt.savefig('plots/feature_importance.png', dpi=150)
```

20.

---

# Quick tips to maintain Snowflake-like style in pandas

- Think in **tables** and **group by** + **window** operations. Use `.groupby(...).agg(...)` and `.shift()` as your `LAG/LEAD`.
- When you'd do `LEFT JOIN` in SQL, use `merge(..., how='left')`.
- Use `reset_index()` after groupby if you expect a flat table.
- Use `astype()` to coerce types before merging (match keys exactly).
- Use consistent `fillna()` strategies after merges to avoid null explosion.
- Keep a single canonical calendar table and join on `date` to generate fiscal quarters (same as you do in Snowflake).