Novartis Datathon 2025 - Copilot Agent Master Todo

> **Purpose:** This file provides step-by-step instructions for GitHub Copilot Agent to build the entire project from scratch.
> **Execution:** Work through each step sequentially. Mark `[x]` when complete. Do NOT skip steps. **Local Run:** All Python files are provided in complete `.py` format with demo scripts for testing.

Project Summary

Aspect	Details
Goal	Predict 24-month post-generic sales erosion for pharmaceutical drugs
Target	`volume` (monthly sales units)
Scenarios	Scenario 1: Predict months 0-23 (no post-entry data)
	Scenario 2: Predict months 6-23 (months 0-5 provided)
Critical Focus	Bucket 1 (high erosion, mean 0-0.25) weighted 2×
Metric	Custom Prediction Error (PE), lower is better

STEP 1: Project Setup

1.1 Create Directory Structure

PowerShell Commands to create all directories:

```
# Run from Main_project folder
cd D:\Datathon\novartis_datathon_2025\Main_project

# Create all directories
New-Item -ItemType Directory -Force -Path "data\raw"
New-Item -ItemType Directory -Force -Path "data\processed"
New-Item -ItemType Directory -Force -Path "src"
New-Item -ItemType Directory -Force -Path "notebooks"
New-Item -ItemType Directory -Force -Path "models"
New-Item -ItemType Directory -Force -Path "submissions"
New-Item -ItemType Directory -Force -Path "reports\figures"
```

- [ ] **1.1.1** Create folder: `Main_project/data/raw/`
- [ ] **1.1.2** Create folder: `Main_project/data/processed/`
- [ ] **1.1.3** Create folder: `Main_project/src/`

- [ ] **1.1.4** Create folder: `Main_project/notebooks/`
- [ ] **1.1.5** Create folder: `Main_project/models/`
- [ ] **1.1.6** Create folder: `Main_project/submissions/`
- [ ] **1.1.7** Create folder: `Main_project/reports/`
- [ ] **1.1.8** Create folder: `Main_project/reports/figures/`

1.2 Copy Data Files

PowerShell Commands to copy all data files:

```
# Run from Main_project folder
cd D:\Datathon\novartis_datathon_2025\Main_project

# Copy training data
Copy-Item "..\SUBMISSION\Data files\TRAIN\df_volume_train.csv" -Destination
"data\raw\"
Copy-Item "..\SUBMISSION\Data files\TRAIN\df_generics_train.csv" -Destination
"data\raw\"
Copy-Item "..\SUBMISSION\Data files\TRAIN\df_medicine_info_train.csv" -
Destination "data\raw\"

# Copy test data
Copy-Item "..\SUBMISSION\Data files\TEST\df_volume_test.csv" -Destination
"data\raw\"
Copy-Item "..\SUBMISSION\Data files\TEST\df_generics_test.csv" -Destination
"data\raw\"
Copy-Item "..\SUBMISSION\Data files\TEST\df_medicine_info_test.csv" -
Destination "data\raw\"

# Copy metric and submission files
Copy-Item "..\SUBMISSION\Metric files\metric_calculation.py" -Destination
"src\"
Copy-Item "..\SUBMISSION\Submission example\submission_template.csv" -
Destination "submissions\"
Copy-Item "..\SUBMISSION\Submission example\submission_example.csv" -
Destination "submissions\"
```

- [ ] **1.2.1** Copy `SUBMISSION/Data files/TRAIN/df_volume_train.csv` → `data/raw/`
- [ ] **1.2.2** Copy `SUBMISSION/Data files/TRAIN/df_generics_train.csv` → `data/raw/`
- [ ] **1.2.3** Copy `SUBMISSION/Data files/TRAIN/df_medicine_info_train.csv` → `data/raw/`
- [ ] **1.2.4** Copy `SUBMISSION/Data files/TEST/df_volume_test.csv` → `data/raw/`
- [ ] **1.2.5** Copy `SUBMISSION/Data files/TEST/df_generics_test.csv` → `data/raw/`
- [ ] **1.2.6** Copy `SUBMISSION/Data files/TEST/df_medicine_info_test.csv` → `data/raw/`
- [ ] **1.2.7** Copy `SUBMISSION/Metric files/metric_calculation.py` → `src/`
- [ ] **1.2.8** Copy `SUBMISSION/Submission example/submission_template.csv` → `submissions/`
- [ ] **1.2.9** Copy `SUBMISSION/Submission example/submission_example.csv` → `submissions/`

1.3 Verify Dependencies

Create/Update `requirements.txt`:

```
# Core data processing
pandas>=1.5.0
numpy>=1.23.0
scipy>=1.9.0

# Machine Learning
scikit-learn>=1.1.0
lightgbm>=3.3.0
xgboost>=1.7.0
catboost>=1.1.0

# Time Series (optional)
statsmodels>=0.13.0

# Visualization
matplotlib>=3.6.0
seaborn>=0.12.0
plotly>=5.10.0

# Utilities
tqdm>=4.64.0
joblib>=1.2.0

# Jupyter
jupyter>=1.0.0
notebook>=6.5.0
ipykernel>=6.17.0
```

Install dependencies:

```
# Activate virtual environment first
& D:\Datathon\novartis_datathon_2025\saeed_venv\Scripts\Activate.ps1

# Install requirements
pip install -r requirements.txt
```

- ☐ **1.3.1** Check `requirements.txt` has all needed packages
- ☐ **1.3.2** Install: `pip install -r requirements.txt`

🚀 STEP 2: Create Core Python Modules

2.1 Create `src/config.py`

- [ ] **2.1.1** Create configuration file with:

```
# File: src/config.py
# Contains all project paths and constants

from pathlib import Path

# Paths
PROJECT_ROOT = Path(__file__).parent.parent
DATA_RAW = PROJECT_ROOT / "data" / "raw"
DATA_PROCESSED = PROJECT_ROOT / "data" / "processed"
MODELS_DIR = PROJECT_ROOT / "models"
SUBMISSIONS_DIR = PROJECT_ROOT / "submissions"
REPORTS_DIR = PROJECT_ROOT / "reports"

# Data files
VOLUME_TRAIN = DATA_RAW / "df_volume_train.csv"
GENERICS_TRAIN = DATA_RAW / "df_generics_train.csv"
MEDICINE_INFO_TRAIN = DATA_RAW / "df_medicine_info_train.csv"
VOLUME_TEST = DATA_RAW / "df_volume_test.csv"
GENERICS_TEST = DATA_RAW / "df_generics_test.csv"
MEDICINE_INFO_TEST = DATA_RAW / "df_medicine_info_test.csv"

# Constants
PRE_ENTRY_MONTHS = 12  # Months before generic entry for Avg_j calculation
POST_ENTRY_MONTHS = 24  # Months to forecast
BUCKET_1_THRESHOLD = 0.25  # Mean erosion <= 0.25 = Bucket 1
BUCKET_1_WEIGHT = 2  # Bucket 1 weighted 2x in metric
BUCKET_2_WEIGHT = 1

# Metric weights - Scenario 1
S1_MONTHLY_WEIGHT = 0.2
S1_SUM_0_5_WEIGHT = 0.5
S1_SUM_6_11_WEIGHT = 0.2
S1_SUM_12_23_WEIGHT = 0.1

# Metric weights - Scenario 2
S2_MONTHLY_WEIGHT = 0.2
S2_SUM_6_11_WEIGHT = 0.5
S2_SUM_12_23_WEIGHT = 0.3
```

2.2 Create `src/data_loader.py`

- [ ] **2.2.1** Create data loading module with functions:

```
# File: src/data_loader.py
# Functions to load and validate all datasets
```

```python
import pandas as pd
from config import *

def load_volume_data(train=True):
    """Load volume dataset (train or test)"""
    path = VOLUME_TRAIN if train else VOLUME_TEST
    df = pd.read_csv(path)
    # Validate columns: country, brand_name, month, months_postgx, volume
    required_cols = ['country', 'brand_name', 'month', 'months_postgx',
'volume']
    assert all(col in df.columns for col in required_cols), f"Missing columns
in volume data"
    return df

def load_generics_data(train=True):
    """Load generics dataset (train or test)"""
    path = GENERICS_TRAIN if train else GENERICS_TEST
    df = pd.read_csv(path)
    # Validate columns: country, brand_name, months_postgx, n_gxs
    required_cols = ['country', 'brand_name', 'months_postgx', 'n_gxs']
    assert all(col in df.columns for col in required_cols), f"Missing columns
in generics data"
    return df

def load_medicine_info(train=True):
    """Load medicine info dataset (train or test)"""
    path = MEDICINE_INFO_TRAIN if train else MEDICINE_INFO_TEST
    df = pd.read_csv(path)
    # Validate columns: country, brand_name, ther_area, hospital_rate,
main_package, biological, small_molecule
    return df

def load_all_data(train=True):
    """Load all three datasets"""
    volume = load_volume_data(train)
    generics = load_generics_data(train)
    medicine = load_medicine_info(train)
    return volume, generics, medicine

def merge_datasets(volume_df, generics_df, medicine_df):
    """Merge all datasets into unified modeling table"""
    # Merge volume + generics on (country, brand_name, months_postgx)
    merged = volume_df.merge(
        generics_df,
        on=['country', 'brand_name', 'months_postgx'],
        how='left'
    )
    # Merge with medicine_info on (country, brand_name)
    merged = merged.merge(
        medicine_df,
        on=['country', 'brand_name'],
        how='left'
    )
    return merged
```

2.3 Create `src/bucket_calculator.py`

- [ ] **2.3.1** Create bucket calculation module:

```
# File: src/bucket_calculator.py
# Functions to compute Avg_j, normalized volume, and erosion buckets

import pandas as pd
import numpy as np
from config import PRE_ENTRY_MONTHS, BUCKET_1_THRESHOLD

def compute_avg_j(df):
    """
    Compute pre-entry average volume (Avg_j) for each country-brand.
    Avg_j = mean volume over months -12 to -1.
    """
    pre_entry = df[(df['months_postgx'] >= -PRE_ENTRY_MONTHS) &
(df['months_postgx'] <= -1)]
    avg_j = pre_entry.groupby(['country', 'brand_name'])
['volume'].mean().reset_index()
    avg_j.columns = ['country', 'brand_name', 'avg_vol']
    return avg_j

def compute_normalized_volume(df, avg_j_df):
    """
    Compute normalized volume: vol_norm = volume / Avg_j
    """
    merged = df.merge(avg_j_df, on=['country', 'brand_name'], how='left')
    merged['vol_norm'] = merged['volume'] / merged['avg_vol']
    merged['vol_norm'] = merged['vol_norm'].replace([np.inf, -np.inf], np.nan)
    return merged

def compute_mean_erosion(df):
    """
    Compute mean generic erosion for each country-brand.
    Mean erosion = mean(vol_norm) over months 0-23.
    """
    post_entry = df[(df['months_postgx'] >= 0) & (df['months_postgx'] <= 23)]
    mean_erosion = post_entry.groupby(['country', 'brand_name'])
['vol_norm'].mean().reset_index()
    mean_erosion.columns = ['country', 'brand_name', 'mean_erosion']
    return mean_erosion

def assign_buckets(mean_erosion_df):
    """
    Assign erosion bucket based on mean erosion.
    Bucket 1: mean_erosion in [0, 0.25] (high erosion)
    Bucket 2: mean_erosion > 0.25 (lower erosion)
    """
```

```
    mean_erosion_df['bucket'] = np.where(
        (mean_erosion_df['mean_erosion'] >= 0) &
        (mean_erosion_df['mean_erosion'] <= BUCKET_1_THRESHOLD),
        1, 2
    )
    return mean_erosion_df

def create_auxiliary_file(df):
    """
    Create auxiliary file with avg_vol and bucket for metric calculation.
    Returns DataFrame with: country, brand_name, avg_vol, bucket
    """
    avg_j = compute_avg_j(df)
    df_with_norm = compute_normalized_volume(df, avg_j)
    mean_erosion = compute_mean_erosion(df_with_norm)
    buckets = assign_buckets(mean_erosion)

    # Merge avg_vol and bucket
    aux = avg_j.merge(buckets[['country', 'brand_name', 'mean_erosion',
'bucket']],
                      on=['country', 'brand_name'], how='left')
    return aux
```

2.4 Create `src/feature_engineering.py`

- [ ] **2.4.1** Create feature engineering module:

```
# File: src/feature_engineering.py
# Functions to create features for modeling

import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

def create_lag_features(df, lags=[1, 2, 3, 6, 12]):
    """Create lag features for volume"""
    df = df.sort_values(['country', 'brand_name', 'months_postgx'])
    for lag in lags:
        df[f'volume_lag_{lag}'] = df.groupby(['country', 'brand_name'])
['volume'].shift(lag)
    return df

def create_rolling_features(df, windows=[3, 6, 12]):
    """Create rolling mean and std features"""
    df = df.sort_values(['country', 'brand_name', 'months_postgx'])
    for window in windows:
        df[f'volume_rolling_mean_{window}'] = df.groupby(['country',
'brand_name'])['volume'].transform(
            lambda x: x.rolling(window, min_periods=1).mean()
        )
```

```
        df[f'volume_rolling_std_{window}'] = df.groupby(['country',
'brand_name'])['volume'].transform(
            lambda x: x.rolling(window, min_periods=1).std()
        )
    return df

def create_pre_entry_features(df, avg_j_df):
    """Create pre-entry aggregate features per country-brand"""
    pre_entry = df[df['months_postgx'] < 0].copy()

    # Pre-entry slope (linear trend)
    def calc_slope(group):
        if len(group) < 2:
            return 0
        x = group['months_postgx'].values
        y = group['volume'].values
        if np.std(x) == 0:
            return 0
        return np.polyfit(x, y, 1)[0]

    slopes = pre_entry.groupby(['country',
'brand_name']).apply(calc_slope).reset_index()
    slopes.columns = ['country', 'brand_name', 'pre_entry_slope']

    # Pre-entry volatility
    volatility = pre_entry.groupby(['country', 'brand_name'])
['volume'].std().reset_index()
    volatility.columns = ['country', 'brand_name', 'pre_entry_volatility']

    # Merge features
    features = avg_j_df.merge(slopes, on=['country', 'brand_name'],
how='left')
    features = features.merge(volatility, on=['country', 'brand_name'],
how='left')

    return features

def encode_categorical_features(df, columns=['country', 'ther_area',
'main_package']):
    """Label encode categorical columns"""
    encoders = {}
    for col in columns:
        if col in df.columns:
            le = LabelEncoder()
            df[f'{col}_encoded'] = le.fit_transform(df[col].astype(str))
            encoders[col] = le
    return df, encoders

def create_competition_features(df):
    """Create features from generics competition"""
    df = df.sort_values(['country', 'brand_name', 'months_postgx'])

    # Cumulative max generics
    df['n_gxs_cummax'] = df.groupby(['country', 'brand_name'])
```

```
    ['n_gxs'].cummax()

        # Change in number of generics
        df['n_gxs_change'] = df.groupby(['country', 'brand_name'])
    ['n_gxs'].diff().fillna(0)

        # First month with generics
        df['has_generics'] = (df['n_gxs'] > 0).astype(int)

        return df

    def create_time_features(df):
        """Create time-based features"""
        # Months since entry (already have months_postgx)
        df['months_postgx_squared'] = df['months_postgx'] ** 2
        df['months_postgx_log'] = np.log1p(df['months_postgx'].clip(lower=0))

        # Quarter indicator
        df['quarter'] = ((df['months_postgx'] % 12) // 3) + 1

        return df
```

2.5 Create `src/models.py`

- [ ] **2.5.1** Create modeling module:

```
# File: src/models.py
# Model classes and training functions

import pandas as pd
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error
import lightgbm as lgb
import xgboost as xgb
import joblib
from config import MODELS_DIR

class BaselineModels:
    """Simple baseline models for comparison"""

    @staticmethod
    def naive_persistence(avg_j_df, months_to_predict):
        """Predict avg_j for all future months"""
        predictions = []
        for _, row in avg_j_df.iterrows():
            for month in months_to_predict:
                predictions.append({
                    'country': row['country'],
                    'brand_name': row['brand_name'],
```

```
                    'months_postgx': month,
                    'volume': row['avg_vol']
                })
        return pd.DataFrame(predictions)

    @staticmethod
    def linear_decay(avg_j_df, months_to_predict, decay_rate=0.05):
        """Predict linear decay from avg_j"""
        predictions = []
        for _, row in avg_j_df.iterrows():
            for month in months_to_predict:
                decayed_volume = row['avg_vol'] * (1 - decay_rate * month)
                decayed_volume = max(0, decayed_volume)  # No negative volumes
                predictions.append({
                    'country': row['country'],
                    'brand_name': row['brand_name'],
                    'months_postgx': month,
                    'volume': decayed_volume
                })
        return pd.DataFrame(predictions)

    @staticmethod
    def exponential_decay(avg_j_df, months_to_predict, decay_rate=0.1):
        """Predict exponential decay from avg_j"""
        predictions = []
        for _, row in avg_j_df.iterrows():
            for month in months_to_predict:
                decayed_volume = row['avg_vol'] * np.exp(-decay_rate * month)
                predictions.append({
                    'country': row['country'],
                    'brand_name': row['brand_name'],
                    'months_postgx': month,
                    'volume': decayed_volume
                })
        return pd.DataFrame(predictions)


class GradientBoostingModel:
    """LightGBM/XGBoost model for volume prediction"""

    def __init__(self, model_type='lightgbm', params=None):
        self.model_type = model_type
        self.model = None
        self.params = params or self._default_params()

    def _default_params(self):
        if self.model_type == 'lightgbm':
            return {
                'objective': 'regression',
                'metric': 'rmse',
                'boosting_type': 'gbdt',
                'num_leaves': 31,
                'learning_rate': 0.05,
                'feature_fraction': 0.8,
```

```
                'bagging_fraction': 0.8,
                'bagging_freq': 5,
                'verbose': -1,
                'n_estimators': 500,
                'early_stopping_rounds': 50
            }
        else:  # xgboost
            return {
                'objective': 'reg:squarederror',
                'eval_metric': 'rmse',
                'max_depth': 6,
                'learning_rate': 0.05,
                'n_estimators': 500,
                'subsample': 0.8,
                'colsample_bytree': 0.8,
                'early_stopping_rounds': 50
            }

    def fit(self, X_train, y_train, X_val=None, y_val=None):
        if self.model_type == 'lightgbm':
            self.model = lgb.LGBMRegressor(**self.params)
            if X_val is not None:
                self.model.fit(X_train, y_train,
                               eval_set=[(X_val, y_val)],
                               callbacks=[lgb.early_stopping(50)])
            else:
                self.model.fit(X_train, y_train)
        else:
            self.model = xgb.XGBRegressor(**self.params)
            if X_val is not None:
                self.model.fit(X_train, y_train,
                               eval_set=[(X_val, y_val)],
                               verbose=False)
            else:
                self.model.fit(X_train, y_train)
        return self

    def predict(self, X):
        return self.model.predict(X)

    def get_feature_importance(self):
        if self.model is None:
            return None
        return pd.DataFrame({
            'feature': self.model.feature_name_ if self.model_type ==
'lightgbm'
                       else self.model.get_booster().feature_names,
            'importance': self.model.feature_importances_
        }).sort_values('importance', ascending=False)

    def save(self, name):
        path = MODELS_DIR / f"{name}.joblib"
        joblib.dump(self.model, path)
```

```
    def load(self, name):
        path = MODELS_DIR / f"{name}.joblib"
        self.model = joblib.load(path)
        return self
```

2.6 Create `src/evaluation.py`

- [ ] **2.6.1** Create evaluation module:

```
# File: src/evaluation.py
# Functions to evaluate models using official PE metric

import pandas as pd
import numpy as np
from config import *

def compute_pe_scenario1(actual_df, pred_df, aux_df):
    """
    Compute Prediction Error for Scenario 1 (months 0-23).

    PE = 0.2 * (sum|actual-pred| / 24*avg_vol)
       + 0.5 * (|sum(0-5) actual - sum(0-5) pred| / 6*avg_vol)
       + 0.2 * (|sum(6-11) actual - sum(6-11) pred| / 6*avg_vol)
       + 0.1 * (|sum(12-23) actual - sum(12-23) pred| / 12*avg_vol)
    """
    merged = actual_df.merge(
        pred_df, on=['country', 'brand_name', 'months_postgx'],
        suffixes=('_actual', '_pred')
    ).merge(aux_df, on=['country', 'brand_name'])

    results = []
    for (country, brand), group in merged.groupby(['country', 'brand_name']):
        avg_vol = group['avg_vol'].iloc[0]
        bucket = group['bucket'].iloc[0]

        if avg_vol == 0 or np.isnan(avg_vol):
            continue

        # Monthly absolute error (0-23)
        monthly_err =
group['volume_actual'].sub(group['volume_pred']).abs().sum()
        term1 = S1_MONTHLY_WEIGHT * monthly_err / (24 * avg_vol)

        # Accumulated error 0-5
        m0_5 = group[group['months_postgx'].between(0, 5)]
        sum_err_0_5 = abs(m0_5['volume_actual'].sum() -
m0_5['volume_pred'].sum())
        term2 = S1_SUM_0_5_WEIGHT * sum_err_0_5 / (6 * avg_vol)

        # Accumulated error 6-11
```

```
        m6_11 = group[group['months_postgx'].between(6, 11)]
        sum_err_6_11 = abs(m6_11['volume_actual'].sum() -
m6_11['volume_pred'].sum())
        term3 = S1_SUM_6_11_WEIGHT * sum_err_6_11 / (6 * avg_vol)

        # Accumulated error 12-23
        m12_23 = group[group['months_postgx'].between(12, 23)]
        sum_err_12_23 = abs(m12_23['volume_actual'].sum() -
m12_23['volume_pred'].sum())
        term4 = S1_SUM_12_23_WEIGHT * sum_err_12_23 / (12 * avg_vol)

        pe = term1 + term2 + term3 + term4
        results.append({
            'country': country,
            'brand_name': brand,
            'bucket': bucket,
            'PE': pe
        })

    return pd.DataFrame(results)

def compute_pe_scenario2(actual_df, pred_df, aux_df):
    """
    Compute Prediction Error for Scenario 2 (months 6-23).

    PE = 0.2 * (sum|actual-pred| / 18*avg_vol)
       + 0.5 * (|sum(6-11) actual - sum(6-11) pred| / 6*avg_vol)
       + 0.3 * (|sum(12-23) actual - sum(12-23) pred| / 12*avg_vol)
    """
    merged = actual_df.merge(
        pred_df, on=['country', 'brand_name', 'months_postgx'],
        suffixes=('_actual', '_pred')
    ).merge(aux_df, on=['country', 'brand_name'])

    results = []
    for (country, brand), group in merged.groupby(['country', 'brand_name']):
        avg_vol = group['avg_vol'].iloc[0]
        bucket = group['bucket'].iloc[0]

        if avg_vol == 0 or np.isnan(avg_vol):
            continue

        # Filter to months 6-23
        group = group[group['months_postgx'] >= 6]

        # Monthly absolute error (6-23)
        monthly_err =
group['volume_actual'].sub(group['volume_pred']).abs().sum()
        term1 = S2_MONTHLY_WEIGHT * monthly_err / (18 * avg_vol)

        # Accumulated error 6-11
        m6_11 = group[group['months_postgx'].between(6, 11)]
        sum_err_6_11 = abs(m6_11['volume_actual'].sum() -
m6_11['volume_pred'].sum())
```

```
            term2 = S2_SUM_6_11_WEIGHT * sum_err_6_11 / (6 * avg_vol)

            # Accumulated error 12-23
            m12_23 = group[group['months_postgx'].between(12, 23)]
            sum_err_12_23 = abs(m12_23['volume_actual'].sum() -
    m12_23['volume_pred'].sum())
            term3 = S2_SUM_12_23_WEIGHT * sum_err_12_23 / (12 * avg_vol)

            pe = term1 + term2 + term3
            results.append({
                'country': country,
                'brand_name': brand,
                'bucket': bucket,
                'PE': pe
            })

        return pd.DataFrame(results)

    def compute_final_metric(pe_df):
        """
        Compute final weighted metric.
        PE_final = (2/n_B1) * sum(PE_B1) + (1/n_B2) * sum(PE_B2)
        """
        bucket1 = pe_df[pe_df['bucket'] == 1]
        bucket2 = pe_df[pe_df['bucket'] == 2]

        n_b1 = len(bucket1)
        n_b2 = len(bucket2)

        if n_b1 == 0 and n_b2 == 0:
            return np.nan

        score = 0
        if n_b1 > 0:
            score += (BUCKET_1_WEIGHT / n_b1) * bucket1['PE'].sum()
        if n_b2 > 0:
            score += (BUCKET_2_WEIGHT / n_b2) * bucket2['PE'].sum()

        return score

    def evaluate_model(actual_df, pred_df, aux_df, scenario=1):
        """Full evaluation pipeline"""
        if scenario == 1:
            pe_df = compute_pe_scenario1(actual_df, pred_df, aux_df)
        else:
            pe_df = compute_pe_scenario2(actual_df, pred_df, aux_df)

        final_score = compute_final_metric(pe_df)

        # Bucket-level breakdown
        bucket1_score = pe_df[pe_df['bucket'] == 1]['PE'].mean() if
    len(pe_df[pe_df['bucket'] == 1]) > 0 else np.nan
        bucket2_score = pe_df[pe_df['bucket'] == 2]['PE'].mean() if
    len(pe_df[pe_df['bucket'] == 2]) > 0 else np.nan
```

```
    return {
        'final_score': final_score,
        'bucket1_avg_pe': bucket1_score,
        'bucket2_avg_pe': bucket2_score,
        'n_bucket1': len(pe_df[pe_df['bucket'] == 1]),
        'n_bucket2': len(pe_df[pe_df['bucket'] == 2]),
        'pe_details': pe_df
    }
```

2.7 Create `src/submission.py`

- ☐ **2.7.1** Create submission generation module:

```
# File: src/submission.py
# Functions to generate and validate submission files

import pandas as pd
from datetime import datetime
from config import SUBMISSIONS_DIR

def generate_submission(predictions_df, scenario, validate=True):
    """
    Generate submission file from predictions.

    predictions_df must have columns: country, brand_name, months_postgx,
volume
    """
    required_cols = ['country', 'brand_name', 'months_postgx', 'volume']
    assert all(col in predictions_df.columns for col in required_cols), \
        f"Missing columns. Required: {required_cols}"

    submission = predictions_df[required_cols].copy()

    if validate:
        validate_submission(submission, scenario)

    return submission

def validate_submission(submission_df, scenario):
    """Validate submission format and content"""
    # Check no missing values
    assert submission_df['volume'].notna().all(), "Found NaN in volume
predictions"

    # Check no negative volumes
    assert (submission_df['volume'] >= 0).all(), "Found negative volume
predictions"

    # Check months_postgx range
```

```
    if scenario == 1:
        expected_months = set(range(0, 24))  # 0-23
    else:
        expected_months = set(range(6, 24))  # 6-23

    for (country, brand), group in submission_df.groupby(['country',
'brand_name']):
        actual_months = set(group['months_postgx'].values)
        assert actual_months == expected_months, \
            f"Wrong months for {country}/{brand}. Expected {expected_months},
got {actual_months}"

    print(f"☑ Submission validation passed for Scenario {scenario}")
    return True

def save_submission(submission_df, scenario, suffix=""):
    """Save submission to file with timestamp"""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"scenario{scenario}_{timestamp}{suffix}.csv"
    filepath = SUBMISSIONS_DIR / filename
    submission_df.to_csv(filepath, index=False)
    print(f"☑ Saved submission to {filepath}")
    return filepath

def load_submission_template():
    """Load the official submission template"""
    template_path = SUBMISSIONS_DIR / "submission_template.csv"
    return pd.read_csv(template_path)
```

🚀 STEP 3: Data Quality, Validation & Preprocessing

3.1 Create `notebooks/01_data_exploration.ipynb`

- ☐ **3.1.1** Create notebook with cells for initial data loading and validation:

Cell Group 1: Load & Inspect All Datasets

```
# Load all datasets
# Display shape, dtypes, and info for each:
# - df_volume_train/test: columns (country, brand_name, month, months_postgx,
volume)
# - df_generics_train/test: columns (country, brand_name, months_postgx,
n_gxs)
# - df_medicine_info_train/test: columns (country, brand_name, ther_area,
hospital_rate, main_package, biological, small_molecule)
```

Cell Group 2: Missing Data Analysis

- [ ] **3.1.2** Check missing values per column per dataset
- [ ] **3.1.3** Document missing patterns:
  - Are missing values systematic (e.g., certain months or brands)?
  - Which columns have missing values that need imputation?

Cell Group 3: Duplicate Detection

- [ ] **3.1.4** Check for duplicate rows on key combinations:
  - `df_volume` : duplicates on `(country, brand_name, months_postgx)`
  - `df_generics` : duplicates on `(country, brand_name, months_postgx)`
  - `df_medicine_info` : duplicates on `(country, brand_name)`
- [ ] **3.1.5** If duplicates found, investigate and decide: keep first, average, or remove?

Cell Group 4: Range & Distribution Validation

- [ ] **3.1.6** Validate `hospital_rate` is between 0 and 100
- [ ] **3.1.7** Validate `months_postgx` range: should be -24 to +23 for train
- [ ] **3.1.8** Validate `n_gxs` increases on/after entry (months_postgx >= 0)
- [ ] **3.1.9** Check for impossible/negative `volume` values
- [ ] **3.1.10** Summarize min/max values by brand and overall for `volume`

Cell Group 5: Unique Combinations

- [ ] **3.1.11** Count unique country-brand combinations:
  - Training: Expected ~1,953
  - Test: Expected ~340
- [ ] **3.1.12** Verify no overlap between train and test country-brand pairs

Cell Group 6: Data Quality Report

- [ ] **3.1.13** Generate `reports/data_quality_report.md` with:
  - Missing value summary table
  - Duplicate findings
  - Range violations flagged
  - Recommendations for cleaning

3.2 Create `notebooks/02_bucket_analysis.ipynb`

- [ ] **3.2.1** Create notebook for bucket computation and validation:

Cell Group 1: Compute Avg_j (Pre-Entry Average)

```
# For each (country, brand_name):
#   Filter rows where months_postgx in [-12, -1]
#   Compute: Avg_j = mean(volume) over these 12 months
# Handle edge cases: brands with fewer than 12 pre-entry months
```

Cell Group 2: Compute Normalized Volume

```
# For each post-entry row (months_postgx >= 0):
#   vol_norm = volume / Avg_j
# Handle division by zero (Avg_j = 0 or NaN)
```

Cell Group 3: Compute Mean Generic Erosion

```
# For each (country, brand_name):
#   Filter rows where months_postgx in [0, 23]
#   mean_erosion = mean(vol_norm)
```

Cell Group 4: Assign Erosion Buckets

```
# Bucket 1: mean_erosion in [0, 0.25] → High erosion (double weighted!)
# Bucket 2: mean_erosion in (0.25, 1] → Lower erosion
```

Cell Group 5: Bucket Distribution Analysis

- ☐ **3.2.2** Count and percentage of Bucket 1 vs Bucket 2
- ☐ **3.2.3** Cross-tabulate buckets by:
  - `ther_area`
  - `biological` vs `small_molecule`
  - `hospital_rate` (binned: low/medium/high)
- ☐ **3.2.4** Compare avg_vol distribution between buckets

Cell Group 6: Save Auxiliary File

- ☐ **3.2.5** Save `data/processed/aux_bucket_avgvol.csv` with columns:
  - `country`, `brand_name`, `avg_vol`, `mean_erosion`, `bucket`
- ☐ **3.2.6 IMPORTANT:** Do NOT use `auxiliar_metric_computation_example.csv` as input features

3.3 Create `notebooks/03_eda.ipynb` - Comprehensive EDA

- ☐ **3.3.1** Create comprehensive EDA notebook with the following analysis sections:

EDA Section A: Time Series Visualization

A1. Sample Brand Trajectories

- [ ] **3.3.2** Plot volumes over time (using `month` column) for 5-10 sample brands
- [ ] **3.3.3** Mark Month 0 (generic entry) with vertical line on each plot
- [ ] **3.3.4** Show full lifecycle: pre-entry stability → generic entry → post-entry erosion

A2. Life-Cycle Trajectory Analysis

- [ ] **3.3.5** For each brand, identify phases:
  - **Pre-entry (months_postgx < 0):** Growth or stabilization phase
  - **At entry (month 0):** Inflection point
  - **Post-entry (months_postgx ≥ 0):** Erosion phase
- [ ] **3.3.6** Plot typical shapes: growth → plateau → drop
- [ ] **3.3.7** Identify variations by therapeutic area or region

A3. Calendar/Macro Effects

- [ ] **3.3.8** Look for global patterns separate from generic entry effects:
  - E.g., drop in 2020 for many countries (COVID impact?)
  - Seasonal prescribing patterns
- [ ] **3.3.9** Document: "Part of variation might be driven by macro events not explicitly modeled"

EDA Section B: Bucket 1 vs Bucket 2 Comparison

B1. Erosion Curve Comparison

- [ ] **3.3.10** Compare average erosion curves (normalized volume vs months_postgx):
  - **Bucket 1:** Mean erosion 0–0.25 (sharp drops)
  - **Bucket 2:** Mean erosion >0.25–1 (more stable)
- [ ] **3.3.11** Analyze:
  - How quickly volumes fall in each bucket
  - How early the main drop occurs (first 6–12 months vs later)

B2. Drop Speed Analysis

- [ ] **3.3.12** Calculate and compare:
  - Time to 50% erosion (months to reach 0.5 normalized volume)
  - Erosion velocity in first 6 months
  - Final equilibrium level (months 18-23 average)

B3. Bucket Characteristics

- [ ] **3.3.13** Compare distributions by bucket:
  - Pre-entry average volume (`avg_vol`)
  - Therapeutic areas represented

- Biological vs small_molecule proportion
- Hospital rate distribution

EDA Section C: Generic Competition Analysis (n_gxs)

C1. n_gxs Trajectory Patterns

- [ ] **3.3.14** Plot n_gxs trajectories for sample brands
- [ ] **3.3.15** Confirm n_gxs = 0 before entry, increases on/after entry

C2. Competition Impact Analysis

- [ ] **3.3.16** Explore how erosion is affected by:
  - **Timing of first generic:** Does earlier entry cause faster erosion?
  - **Speed of n_gxs rise:** 0→1→2→... How fast do competitors accumulate?
  - **Level of competition:** Compare erosion with 1 vs 3 vs 5+ generics

C3. Non-Linear Competition Effects

- [ ] **3.3.17** Investigate whether:
  - First few entrants cause disproportionate drop
  - Additional generics have diminishing incremental impact
- [ ] **3.3.18** Plot: erosion rate vs n_gxs level (expect diminishing returns)

EDA Section D: Drug Characteristics Analysis (df_medicine_info)

D1. Therapeutic Area Analysis

- [ ] **3.3.19** Analyze by `ther_area`:
  - Which therapeutic areas have highest erosion?
  - Do oncology drugs erode differently from cardiovascular or anti-infectives?
  - Plot erosion curves by therapeutic area

D2. Biological vs Small Molecule

- [ ] **3.3.20** Compare `biological` vs `small_molecule`:
  - **Hypothesis:** Biologicals erode more slowly due to:
    - Biosimilar dynamics and regulation
    - Complexity of substitution
    - Clinical guidelines
- [ ] **3.3.21** Plot average erosion curves for each type

D3. Hospital Rate Analysis

- [ ] **3.3.22** Analyze by `hospital_rate`:
  - Bin into: Low (0-25%), Medium (25-75%), High (75-100%)

    - **Hypothesis:** High hospital_rate may show different erosion shapes:
        - Tender-driven, stepwise changes
        - Different prescribing dynamics vs retail
- [ ] **3.3.23** Plot erosion curves by hospital_rate bin

D4. Main Package Analysis

- [ ] **3.3.24** Analyze by `main_package` (PILL, INJECTION, EYE DROP, etc.):
    - Does packaging format correlate with substitution ease?
    - Different usage patterns?

EDA Section E: Train vs Validation Distribution Alignment

E1. Volume Distribution Comparison

- [ ] **3.3.25** Create pseudo train/validation split (time-based)
- [ ] **3.3.26** Compare basic stats between train and validation:
    - Mean, median, standard deviation of volume
    - Key quantiles (10th, 50th, 90th)

E2. Bucket Distribution Alignment

- [ ] **3.3.27** Compare:
    - Bucket 1/Bucket 2 proportions in train vs validation
    - Mean normalized erosion across buckets

E3. Segment-Level Alignment

- [ ] **3.3.28** Compare distributions by:
    - `ther_area`
    - `biological` vs `small_molecule`
    - High vs low `hospital_rate`
- [ ] **3.3.29** Ensure validation ranges and erosion patterns look similar to training
- [ ] **3.3.30** If systematic differences exist, document and interpret metrics accordingly

---

EDA Section F: Interaction Effects & Segmentation Insights

F1. Expected Non-Linear Effects

- [ ] **3.3.31** Document anticipated non-linear/interaction effects:
    - **Biological vs small_molecule:** Different erosion speeds
    - **Hospital_rate × erosion:** Tender-driven vs retail dynamics
    - **Therapeutic_area × erosion:** Disease-specific patterns
    - **n_gxs trajectory:** Diminishing returns from additional generics

F2. Segmentation Recommendations

- [ ] **3.3.32** Based on EDA findings, evaluate whether to:
  - Train separate models for Bucket 1 vs Bucket 2 (for training/analysis only, NOT using bucket as input feature)
  - Train separate models for biological vs small_molecule
  - Add interaction features (e.g., `ther_area × n_gxs`, `hospital_rate × months_postgx`)

F3. Model Type Recommendation

- [ ] **3.3.33** Document: Tree-based models (LightGBM, XGBoost, CatBoost) are strong candidates because they can:
  - Handle interactions automatically
  - Capture non-linear relationships
  - Handle categorical variables natively

EDA Section G: Data Cleaning Decisions

G1. Impossible Values

- [ ] **3.3.34** Identify clearly impossible values:
  - Negative volumes
  - hospital_rate outside 0-100
  - n_gxs > 0 before months_postgx = 0
- [ ] **3.3.35** Decide treatment: remove, clip, or impute

G2. Outlier Analysis

- [ ] **3.3.36** Identify extreme volume outliers
- [ ] **3.3.37** Decide: keep (may be valid) or winsorize

EDA Section H: Save Outputs

- [ ] **3.3.38** Save key figures to `reports/figures/`:
  - `erosion_curves_by_bucket.png`
  - `volume_trajectories_sample.png`
  - `n_gxs_impact_analysis.png`
  - `ther_area_erosion_comparison.png`
  - `biological_vs_smallmolecule.png`
  - `hospital_rate_erosion.png`
  - `train_val_distribution_comparison.png`
- [ ] **3.3.39** Generate `reports/eda_report.md` with:
  - Executive summary of key findings
  - Data quality issues found
  - Bucket distribution analysis
  - Key patterns discovered

- Feature engineering recommendations
- Model type recommendations
- Segmentation strategy recommendations

🚀 STEP 4: Feature Engineering

4.1 Create `notebooks/04_feature_engineering.ipynb`

- [ ] **4.1.1** Create comprehensive feature engineering notebook:

Feature Category 1: Pre-Entry Features (CRITICAL for Scenario 1)

> ⚠ **Constraint:** For Scenario 1, only use pre-entry information (months_postgx < 0)

1A. Pre-Entry Average Volume (avg_vol)

- [ ] **4.1.2** Compute `avg_vol` : Mean volume over last 12 months before entry
- [ ] **4.1.3** This is the key scale parameter and metric normalization factor

1B. Pre-Entry Trend / Slope

- [ ] **4.1.4** Compute `pre_entry_slope` : Linear trend of volume over pre-entry months
  - Captures whether brand was growing or declining before generics
- [ ] **4.1.5** Compute `pre_entry_volatility` : Std dev of pre-entry volumes

1C. Pre-Entry Trajectory Shape

- [ ] **4.1.6** Compute `pre_entry_growth_rate` : (vol_m-1 - vol_m-12) / vol_m-12
- [ ] **4.1.7** Identify phase: growth, plateau, or decline before entry

---

Feature Category 2: Lag Features on Volume

2A. Recent Volume Lags

- [ ] **4.1.8** Create lag features (respecting time causality):
  - `volume_lag_1` , `volume_lag_2` , `volume_lag_3` (previous months)
  - `volume_lag_6` , `volume_lag_12` (if available)
- [ ] **4.1.9 For Scenario 1:** Only use pre-entry lags at prediction time
- [ ] **4.1.10 For Scenario 2:** Can use months 0-5 actuals for lags

2B. Rolling Statistics

- [ ] **4.1.11** Create rolling features:
  - `volume_rolling_mean_3` , `volume_rolling_mean_6` , `volume_rolling_mean_12`
  - `volume_rolling_std_3` , `volume_rolling_std_6`

Feature Category 3: Time Since Generic Entry

3A. Basic Time Features

- [ ] **4.1.12** Use `months_postgx` directly (already aligned)
- [ ] **4.1.13** Create `months_postgx_squared` (erosion shape is non-linear)
- [ ] **4.1.14** Create piecewise indicators:
  - `is_first_6_months` : months_postgx in [0, 5]
  - `is_months_6_11` : months_postgx in [6, 11]
  - `is_months_12_plus` : months_postgx >= 12

Feature Category 4: Generics Competition Features

4A. Basic Competition Features

- [ ] **4.1.15** Use `n_gxs` at each month
- [ ] **4.1.16** Create `n_gxs_cummax` : Cumulative max generics seen up to month t

4B. Competition Dynamics

- [ ] **4.1.17** Create `n_gxs_change` : Change in number of generics from previous month
- [ ] **4.1.18** Create `first_generic_entered` : Indicator (n_gxs changes from 0 to >0)
- [ ] **4.1.19** Create `months_since_first_generic` : Time since n_gxs first became >0

4C. Competition Bins

- [ ] **4.1.20** Create `n_gxs_bin` : low (1-2), medium (3-5), high (6+)

Feature Category 5: Drug Characteristics (Static Features)

5A. Therapeutic Area

- [ ] **4.1.21** Encode `ther_area` :
  - Option A: One-hot encoding (if few categories)
  - Option B: Target encoding (if many categories)
  - Option C: Leave as categorical for LightGBM/CatBoost

5B. Drug Type

- [ ] **4.1.22** Use `biological` (binary: 0/1)
- [ ] **4.1.23** Use `small_molecule` (binary: 0/1)

5C. Hospital Rate

- [ ] **4.1.24** Use `hospital_rate` (numeric 0-100)
- [ ] **4.1.25** Optionally bin: `hospital_rate_bin` (low/medium/high)

5D. Package Type

- [ ] **4.1.26** Encode `main_package`:
  - Aggregate rare categories to "OTHER" if needed
  - One-hot or label encode

Feature Category 6: Seasonality / Calendar Features

6A. Calendar Features from `month`

- [ ] **4.1.27** Extract `month_of_year` from `month` column (Jan=1, Dec=12)
- [ ] **4.1.28** Create `quarter` indicator (Q1, Q2, Q3, Q4)
- [ ] **4.1.29** Optionally: `year` if calendar year effects observed in EDA

Feature Category 7: Interaction Features (Optional)

7A. Key Interactions

- [ ] **4.1.30** Consider creating interactions identified in EDA:
  - `biological × months_postgx`
  - `hospital_rate × months_postgx`
  - `n_gxs × months_postgx`
  - `ther_area × n_gxs` (if using embeddings or target encoding)

Feature Category 8: Identifier Encoding

8A. Country Encoding

- [ ] **4.1.31** Encode `country`:
  - Label encoding for tree models
  - Or derive higher-level groupings (region)
- [ ] **4.1.32** Do NOT use raw country string as-is

8B. Brand Name

- [ ] **4.1.33** `brand_name` should NOT be used directly as a feature
  - Can be used for grouping/aggregation only
  - Or target encoding if you want brand-level signals

Feature Priority Summary

Priority	Feature	Notes
High	`avg_vol` (pre-entry average)	Key scale parameter, used in metric

Priority	Feature	Notes
🟢 High	`pre_entry_slope`	Captures growth/decline trajectory
🟢 High	`months_postgx`	Time since entry (+ squared)
🟢 High	`n_gxs`	Number of generics at month t
🟢 High	`biological`, `small_molecule`	Drug type affects erosion speed
🟡 Medium	`hospital_rate`	Channel dynamics
🟡 Medium	`ther_area`	Therapeutic patterns
🟡 Medium	`volume_lag_*`	Recent volume history
🟡 Medium	`volume_rolling_mean_*`	Smoothed trends
🔴 Low	Fine-grained `main_package`	May be noisy
🔴 Low	Raw identifiers	Need proper encoding

Constraint Reminders

> ⚠ **CRITICAL CONSTRAINTS:**
>
> 1. **Scenario 1:** Use only pre-entry information when simulating (no post-entry actuals at prediction time)
> 2. **Scenario 2:** Can use months 0-5 actuals as features
> 3. **Respect time causality:** Never use future information as features
> 4. **Do NOT use `auxiliar_metric_computation_example.csv` as input features**
> 5. **Do NOT use bucket as an input feature** (it's derived from target)

Save Feature Tables

- [ ] **4.1.34** Save `data/processed/features_train.csv`
- [ ] **4.1.35** Save `data/processed/features_test.csv`
- [ ] **4.1.36** Generate `reports/feature_dictionary.md` documenting all features

🚀 STEP 5: Create Model Training Notebooks

5.1 Create `notebooks/05_baseline_models.ipynb`

- [ ] **5.1.1** Create baseline models notebook:

Baseline Model 1: Naive Persistence

- [ ] **5.1.2** Implement: Predict `avg_vol` (pre-entry average) for all future months
- [ ] **5.1.3** This is a simple "no erosion" baseline

Baseline Model 2: Linear Decay

- [ ] **5.1.4** Implement: `predicted_volume = avg_vol × (1 - decay_rate × months_postgx)`
- [ ] **5.1.5** Tune decay_rate on validation set

Baseline Model 3: Exponential Decay

- [ ] **5.1.6** Implement: `predicted_volume = avg_vol × exp(-decay_rate × months_postgx)`
- [ ] **5.1.7** Tune decay_rate on validation set

Baseline Evaluation

- [ ] **5.1.8** Evaluate each baseline on validation set using PE metric
- [ ] **5.1.9** Compare PE scores by bucket (Bucket 1 vs Bucket 2)
- [ ] **5.1.10** Save results to `reports/baseline_results.md`
- [ ] **5.1.11** Note: Linear baselines useful for sanity checks but likely NOT sufficient to capture all interactions

---

5.2 Create `notebooks/06_gradient_boosting.ipynb`

- [ ] **5.2.1** Create gradient boosting notebook:

Model Selection Rationale

> **Why Tree-Based Models?** Based on EDA findings, tree-based models (LightGBM, XGBoost, CatBoost) are strong candidates because they:
>
> - Handle interactions automatically
> - Capture non-linear relationships (e.g., diminishing returns of n_gxs)
> - Handle categorical variables natively
> - Are robust to outliers and missing values

Data Preparation

- [ ] **5.2.2** Prepare features for Scenario 1 (pre-entry features only at prediction time)
- [ ] **5.2.3** Create time-series train/validation split (respect temporal ordering)
- [ ] **5.2.4** Verify no data leakage from post-entry to pre-entry

Single Global Model Training

- [ ] **5.2.5** Train LightGBM model on all data
- [ ] **5.2.6** Train XGBoost model on all data
- [ ] **5.2.7** Train CatBoost model on all data (optional)
- [ ] **5.2.8** Hyperparameter tuning using cross-validation or Optuna

Feature Importance Analysis

- [ ] **5.2.9** Extract and plot feature importances
- [ ] **5.2.10** Identify top 10 features driving predictions
- [ ] **5.2.11** Verify expected features (avg_vol, months_postgx, n_gxs) are important

Interaction Effect Validation

- [ ] **5.2.12** Check if model captures expected interactions:
  - biological vs small_molecule: Different erosion speeds
  - hospital_rate × months_postgx: Different dynamics
  - n_gxs: Diminishing incremental impact

Model Evaluation

- [ ] **5.2.13** Evaluate on validation set using official PE metric
- [ ] **5.2.14** Break down performance by Bucket 1 vs Bucket 2
- [ ] **5.2.15** Identify worst-predicted brands for error analysis
- [ ] **5.2.16** Save best model to `models/`

5.3 Create `notebooks/06b_segmented_models.ipynb` (Optional but Recommended)

- [ ] **5.3.1** If EDA suggests very different behaviors, create segmented models:

Segmentation Strategy 1: Bucket-Based Models

> ⚠ **IMPORTANT:** Train separate models for Bucket 1 vs Bucket 2, but do NOT use bucket as input feature (it's derived from target)

- [ ] **5.3.2** Train model on Bucket 1 brands only (for training/analysis)
- [ ] **5.3.3** Train model on Bucket 2 brands only (for training/analysis)
- [ ] **5.3.4** Compare feature importances between bucket-specific models
- [ ] **5.3.5** At prediction time: Use a classifier to predict bucket, then apply appropriate model

Segmentation Strategy 2: Drug Type Models

- [ ] **5.3.6** Train model on `biological = True` brands
- [ ] **5.3.7** Train model on `small_molecule = True` brands
- [ ] **5.3.8** Compare erosion patterns learned by each model

Segmentation Strategy 3: Hospital Rate Models

- [ ] **5.3.9** Train model on high hospital_rate (>75%) brands
- [ ] **5.3.10** Train model on low hospital_rate (<25%) brands
- [ ] **5.3.11** Compare tender-driven vs retail erosion patterns

Segmentation Evaluation

- [ ] **5.3.12** Compare segmented models vs single global model
- [ ] **5.3.13** Evaluate using PE metric (overall and by bucket)
- [ ] **5.3.14** Document: When does segmentation help vs hurt?

5.4 Create `notebooks/07_scenario2_model.ipynb`

- [ ] **5.4.1** Create Scenario 2 specific notebook:

Scenario 2 Feature Engineering

> Scenario 2 provides months 0-5 actuals, which is VERY valuable information

- [ ] **5.4.2** Add features from observed months 0-5:
  - `actual_erosion_0_5` : Mean normalized volume in months 0-5
  - `actual_volume_month_5` : Most recent observation
  - `erosion_velocity_0_5` : Rate of decline in months 0-5
  - `lag_features` from months 0-5

Scenario 2 Model Training

- [ ] **5.4.3** Train model using pre-entry features + months 0-5 actuals
- [ ] **5.4.4** The model should learn: "Given how erosion started, how will it continue?"
- [ ] **5.4.5** Expected: Scenario 2 should have LOWER PE than Scenario 1 (more information)

Scenario 2 Evaluation

- [ ] **5.4.6** Evaluate on validation set using Scenario 2 PE formula
- [ ] **5.4.7** Compare with Scenario 1 model performance
- [ ] **5.4.8** Save model to `models/`

5.5 Model Selection Summary

- [ ] **5.5.1** Create comparison table:

Model	Scenario 1 PE	Scenario 2 PE	Bucket 1 PE	Bucket 2 PE
Naive Persistence				
Linear Decay				
Exponential Decay				
LightGBM Global				
XGBoost Global				
Segmented Models				

- [ ] **5.5.2** Select best model for Scenario 1 submission
- [ ] **5.5.3** Select best model for Scenario 2 submission
- [ ] **5.5.4** Document selection rationale in `reports/model_comparison.md`

🚀 STEP 6: Create Prediction and Submission Notebooks

6.1 Create `notebooks/08_generate_predictions.ipynb`

- [ ] **6.1.1** Create prediction generation notebook:
  1. Load trained models
  2. Load test data
  3. Generate Scenario 1 predictions (228 × 24 = 5,472 rows)
  4. Generate Scenario 2 predictions (112 × 18 = 2,016 rows)
  5. Validate predictions
  6. Save predictions to `submissions/`

6.2 Create `notebooks/09_submission_validation.ipynb`

- [ ] **6.2.1** Create validation notebook:
  1. Load submission files
  2. Check format against template
  3. Validate all country-brand combinations present
  4. Validate months_postgx coverage
  5. Check for negative/missing values
  6. Final quality check

🚀 STEP 7: Create Report Generation

7.1 Create EDA Report

- [ ] **7.1.1** Generate `reports/eda_report.md` with:
  - Data summary statistics
  - Missing value analysis
  - Bucket distribution (counts and percentages)
  - Key visualizations (referenced)
  - Initial modeling hypotheses

7.2 Create Model Comparison Report

- [ ] **7.2.1** Generate `reports/model_comparison.md` with:
  - Baseline model results
  - Gradient boosting results
  - PE scores by scenario and bucket
  - Best model selection rationale

7.3 Create Feature Importance Report

- [ ] **7.3.1** Generate `reports/feature_importance.md` with:
  - Top 10 features
  - Feature importance plots
  - Interpretation

7.4 Create Bucket 1 Focus Report

- [ ] **7.4.1** Generate `reports/bucket1_focus_report.md` with:
  - Bucket 1 characteristics

- Model performance on Bucket 1
- Strategies for improvement
- Hardest-to-predict cases analysis

🚀 STEP 8: Create End-to-End Pipeline

8.1 Create `src/pipeline.py`

- ☐ **8.1.1** Create full pipeline script:

```
# File: src/pipeline.py
# End-to-end pipeline from raw data to submission

import argparse
from data_loader import load_all_data, merge_datasets
from bucket_calculator import create_auxiliary_file
from feature_engineering import *
from models import GradientBoostingModel
from evaluation import evaluate_model
from submission import generate_submission, save_submission
from config import *

def run_pipeline(scenario=1, model_type='lightgbm'):
    """Run complete pipeline"""
    print("=" * 50)
    print(f"Running Pipeline - Scenario {scenario}")
    print("=" * 50)

    # Step 1: Load data
    print("\n📂 Loading data...")
    volume_train, generics_train, medicine_train = load_all_data(train=True)
    volume_test, generics_test, medicine_test = load_all_data(train=False)

    # Step 2: Merge datasets
    print("\n🔗 Merging datasets...")
    train_merged = merge_datasets(volume_train, generics_train,
medicine_train)
    test_merged = merge_datasets(volume_test, generics_test, medicine_test)

    # Step 3: Create auxiliary file
    print("\n📊 Computing buckets and avg_vol...")
    aux_df = create_auxiliary_file(train_merged)
    aux_df.to_csv(DATA_PROCESSED / "aux_bucket_avgvol.csv", index=False)

    # Step 4: Feature engineering
    print("\n🔧 Engineering features...")
    train_features = create_lag_features(train_merged)
    train_features = create_rolling_features(train_features)
    train_features = create_competition_features(train_features)
    train_features = create_time_features(train_features)
```

```
    train_features, encoders = encode_categorical_features(train_features)

    # Step 5: Prepare training data
    print("\n📊 Preparing training data...")
    # Filter to post-entry months for training
    train_data = train_features[train_features['months_postgx'] >= 0].copy()

    # Define feature columns
    feature_cols = [col for col in train_data.columns if col not in
                    ['country', 'brand_name', 'month', 'volume', 'vol_norm']]

    X_train = train_data[feature_cols].fillna(0)
    y_train = train_data['volume']

    # Step 6: Train model
    print(f"\n🤖 Training {model_type} model...")
    model = GradientBoostingModel(model_type=model_type)
    model.fit(X_train, y_train)
    model.save(f"scenario{scenario}_{model_type}")

    # Step 7: Generate predictions
    print("\n🔮 Generating predictions...")
    # Apply same feature engineering to test
    test_features = create_lag_features(test_merged)
    test_features = create_rolling_features(test_features)
    test_features = create_competition_features(test_features)
    test_features = create_time_features(test_features)
    test_features, _ = encode_categorical_features(test_features)

    # Filter to correct months for scenario
    if scenario == 1:
        test_predict = test_features[test_features['months_postgx'].between(0,
23)]
    else:
        test_predict = test_features[test_features['months_postgx'].between(6,
23)]

    X_test = test_predict[feature_cols].fillna(0)
    predictions = model.predict(X_test)

    # Step 8: Create submission
    print("\n📝 Creating submission...")
    submission_df = test_predict[['country', 'brand_name',
'months_postgx']].copy()
    submission_df['volume'] = predictions
    submission_df['volume'] = submission_df['volume'].clip(lower=0)  # No
negative volumes

    submission = generate_submission(submission_df, scenario)
    filepath = save_submission(submission, scenario)

    print("\n✅ Pipeline complete!")
    print(f"   Submission saved to: {filepath}")
```

```
    return submission

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('--scenario', type=int, default=1, choices=[1, 2])
    parser.add_argument('--model', type=str, default='lightgbm', choices=
['lightgbm', 'xgboost'])
    args = parser.parse_args()

    run_pipeline(scenario=args.scenario, model_type=args.model)
```

📊 STEP 9: Validation and Testing

9.1 Validate Metric Implementation

- ☐ **9.1.1** Compare `src/evaluation.py` with official `metric_calculation.py`
- ☐ **9.1.2** Test with known inputs from `auxiliar_metric_computation_example.csv`
- ☐ **9.1.3** Verify bucket weighting is correct

9.2 Cross-Validation

- ☐ **9.2.1** Implement time-series cross-validation
- ☐ **9.2.2** Compute CV scores for all models
- ☐ **9.2.3** Check for overfitting

9.3 Final Validation

- ☐ **9.3.1** Validate Scenario 1 submission format
- ☐ **9.3.2** Validate Scenario 2 submission format
- ☐ **9.3.3** Check row counts: Scenario 1 = 5,472, Scenario 2 = 2,016

📊 Progress Tracking

Step	Description	Status
1	Project Setup	☐ Not Started
2	Core Python Modules	☐ Not Started
3	Data Processing Notebook	☐ Not Started
4	Feature Engineering	☐ Not Started
5	Model Training	☐ Not Started
6	Predictions & Submission	☐ Not Started
7	Reports	☐ Not Started
8	Pipeline	☐ Not Started

Step	Description	Status
9	Validation	☐ Not Started

Critical Reminders for Copilot Agent

Priority 1: Bucket 1 (HIGH EROSION)

- Bucket 1 is weighted **2×** in the final metric
- Always check Bucket 1 performance separately
- Consider training separate models for Bucket 1

Priority 2: Early Months

- **Scenario 1:** Months 0-5 carry **50%** weight
- **Scenario 2:** Months 6-11 carry **50%** weight
- Optimize for early month accuracy

Priority 3: Normalization

- All errors normalized by `Avg_j` (12-month pre-entry average)
- Handle cases where `Avg_j = 0` or `NaN`

Priority 4: No Negative Predictions

- Volume cannot be negative
- Always clip predictions to `>= 0`

Priority 5: Exact Submission Format

- Columns: `country`, `brand_name`, `months_postgx`, `volume`
- Scenario 1: months 0-23 for each country-brand
- Scenario 2: months 6-23 for each country-brand

Final Project Structure

```
Main_project/
├── data/
│   ├── raw/
│   │   ├── df_volume_train.csv
│   │   ├── df_volume_test.csv
│   │   ├── df_generics_train.csv
│   │   ├── df_generics_test.csv
│   │   ├── df_medicine_info_train.csv
│   │   └── df_medicine_info_test.csv
│   └── processed/
│       ├── merged_train.csv
│       ├── merged_test.csv
```

```
    │       ├── features_train.csv
    │       ├── features_test.csv
    │       └── aux_bucket_avgvol.csv
    ├── src/
    │   ├── config.py
    │   ├── data_loader.py
    │   ├── bucket_calculator.py
    │   ├── feature_engineering.py
    │   ├── models.py
    │   ├── evaluation.py
    │   ├── submission.py
    │   ├── pipeline.py
    │   └── metric_calculation.py (copied from SUBMISSION)
    ├── notebooks/
    │   ├── 01_data_exploration.ipynb
    │   ├── 02_bucket_analysis.ipynb
    │   ├── 03_eda.ipynb
    │   ├── 04_feature_engineering.ipynb
    │   ├── 05_baseline_models.ipynb
    │   ├── 06_gradient_boosting.ipynb
    │   ├── 07_scenario2_model.ipynb
    │   ├── 08_generate_predictions.ipynb
    │   └── 09_submission_validation.ipynb
    ├── models/
    │   ├── scenario1_lightgbm.joblib
    │   └── scenario2_lightgbm.joblib
    ├── submissions/
    │   ├── submission_template.csv
    │   ├── scenario1_YYYYMMDD_HHMMSS.csv
    │   └── scenario2_YYYYMMDD_HHMMSS.csv
    ├── reports/
    │   ├── figures/
    │   ├── eda_report.md
    │   ├── model_comparison.md
    │   ├── feature_importance.md
    │   └── bucket1_focus_report.md
    ├── Docs/
    ├── SUBMISSION/
    ├── requirements.txt
    ├── Todo.md
    └── main_todo.md (this file)
```

🎯 SUCCESS CRITERIA

Metric	Target
Scenario 1 PE	< 1.0 (good), < 0.5 (excellent)
Scenario 2 PE	< 0.8 (good), < 0.4 (excellent)
Bucket 1 Accuracy	Better than Bucket 2

Metric	Target
Submission Valid	☑ All checks pass

☑ **when complete.**