# Subway Business Intelligence System

## Complete Project Roadmap & Development Guide

---

## 🎯 Project Overview

**Goal**: Build a comprehensive business intelligence system for Subway restaurants using web scraping, PostgreSQL, and Python to provide actionable insights for business decisions.

**Final Deliverables**:

- Automated data collection pipeline
- PostgreSQL database with optimized schema
- Analytics dashboard with key business metrics
- Recommendation engine for store performance
- Competitive analysis reports

---

## 📚 Phase 1: Python Foundation (Weeks 1-2)

*Estimated Time: 20-25 hours*

### Week 1: Core Python Basics

**Learning Objectives**:

- [ ] Variables, data types, and basic operations
- [ ] Control structures (if/else, loops)
- [ ] Functions and scope
- [ ] Lists, dictionaries, tuples
- [ ] String manipulation and formatting

**Resources**:

- Python.org tutorial (sections 1-9)
- Automate the Boring Stuff (chapters 1-6)
- Practice: HackerRank Python challenges

**Mini Project**: Build a simple restaurant data organizer that reads CSV files and performs basic analysis

### Week 2: Intermediate Concepts

**Learning Objectives**:

- [ ] Exception handling (try/except/finally)
- [ ] File I/O operations
- [ ] Basic OOP (classes, methods, attributes)
- [ ] Modules and packages
- [ ] Virtual environments

**Resources**:

- Real Python OOP tutorials
- Python Module of the Week (PyMOTW)

**Mini Project**: Create a Restaurant class that can store and manipulate restaurant data

---

## 📊 Phase 2: Data Handling Mastery (Weeks 3-4)

*Estimated Time: 25-30 hours*

### Week 3: Pandas & Data Manipulation

**Learning Objectives**:

- [ ] DataFrame creation and manipulation
- [ ] Data cleaning techniques
- [ ] Merging, joining, and concatenating data
- [ ] Grouping and aggregation
- [ ] Handling missing data
- [ ] Date/time operations

**Key Libraries**:

```python
import pandas as pd
import numpy as np
import datetime as dt
from dateutil import parser
```

**Resources**:

- Pandas documentation and tutorials
- "Python for Data Analysis" by Wes McKinney
- Kaggle Learn: Pandas course

**Practice Project**: Clean and analyze sample restaurant data (create mock Subway data)

### Week 4: APIs and JSON Handling

**Learning Objectives**:

- [ ] Making HTTP requests with `requests`
- [ ] Parsing JSON responses
- [ ] API authentication and rate limiting
- [ ] Error handling for network requests
- [ ] Working with nested JSON structures

**Key Libraries**:

```python
import requests
import json
import time
import random
```

**Practice Project**: Build a simple API client that fetches restaurant data from public APIs (like Yelp API)

---

## 🕷️ Phase 3: Web Scraping Skills (Weeks 5-6)

*Estimated Time: 25-30 hours*

### Week 5: Beautiful Soup Fundamentals

**Learning Objectives**:

- [ ] HTML structure and parsing
- [ ] CSS selectors and XPath
- [ ] Extracting text, attributes, and links
- [ ] Handling different HTML structures
- [ ] Managing sessions and cookies

**Key Libraries**:

```python
from bs4 import BeautifulSoup
import requests
import lxml
import html5lib
```

**Resources**:

- Beautiful Soup documentation
- "Web Scraping with Python" by Ryan Mitchell
- ScrapingBee blog tutorials

**Practice Project**: Scrape basic restaurant information from a simple restaurant directory

### Week 6: Advanced Scraping Techniques

**Learning Objectives**:

- [ ] Handling JavaScript-rendered content (Selenium basics)
- [ ] Managing rate limits and being respectful
- [ ] Rotating user agents and proxies
- [ ] Handling forms and dynamic content
- [ ] Data validation and error recovery

**Key Libraries**:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
import undetected_chromedriver as uc
```

**Practice Project**: Scrape Subway store locator and menu information

---

## 🗄️ Phase 4: Database Integration (Week 7)

*Estimated Time: 15-20 hours*

### PostgreSQL & Python Connection

**Learning Objectives**:

- [ ] PostgreSQL installation and setup
- [ ] Database connection with psycopg2/SQLAlchemy
- [ ] Creating tables and indexes
- [ ] Batch inserts and performance optimization
- [ ] Transaction management
- [ ] Environment variables and security

**Key Libraries**:

```python
import psycopg2
from sqlalchemy import create_engine
import python-dotenv
import configparser
```

**Database Schema Design**: [See Phase 5 for detailed schema]

---

## 🧬 Phase 5: Database Architecture Design

*Estimated Time: 10-15 hours*

### Core Tables Structure

```sql
```

```sql
-- Stores table
CREATE TABLE stores (
    store_id SERIAL PRIMARY KEY,
    store_number VARCHAR(10) UNIQUE,
    name VARCHAR(255),
    address TEXT,
    city VARCHAR(100),
    state VARCHAR(50),
    zip_code VARCHAR(10),
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    phone VARCHAR(20),
    opening_hours JSONB,
    services TEXT[],
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Menu items table
CREATE TABLE menu_items (
    item_id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    category VARCHAR(100),
    description TEXT,
    base_price DECIMAL(8, 2),
    calories INTEGER,
    nutritional_info JSONB,
    availability_regions TEXT[],
    is_seasonal BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Store pricing (regional variations)
CREATE TABLE store_pricing (
    pricing_id SERIAL PRIMARY KEY,
    store_id INTEGER REFERENCES stores(store_id),
    item_id INTEGER REFERENCES menu_items(item_id),
    price DECIMAL(8, 2),
    effective_date DATE,
    end_date DATE,
    UNIQUE(store_id, item_id, effective_date)
);
```

```sql
-- Reviews table
CREATE TABLE reviews (
    review_id SERIAL PRIMARY KEY,
    store_id INTEGER REFERENCES stores(store_id),
    source VARCHAR(50), -- 'google', 'yelp', 'facebook'
    source_review_id VARCHAR(255),
    rating INTEGER CHECK (rating >= 1 AND rating <= 5),
    review_text TEXT,
    reviewer_name VARCHAR(255),
    review_date DATE,
    scraped_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(source, source_review_id)
);

-- Competitors table
CREATE TABLE competitors (
    competitor_id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    category VARCHAR(100), -- 'sandwich_shop', 'fast_food', etc.
    store_id INTEGER REFERENCES stores(store_id), -- nearby subway store
    distance_meters INTEGER,
    address TEXT,
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    avg_rating DECIMAL(3, 2),
    price_range VARCHAR(10) -- '$', '$$', '$$$'
);

-- Demographics data
CREATE TABLE demographics (
    demo_id SERIAL PRIMARY KEY,
    store_id INTEGER REFERENCES stores(store_id),
    zip_code VARCHAR(10),
    median_income INTEGER,
    population INTEGER,
    avg_age DECIMAL(4, 2),
    education_level_college_pct DECIMAL(5, 2),
    data_year INTEGER
);

-- Performance metrics (calculated daily)
CREATE TABLE daily_metrics (
    metric_id SERIAL PRIMARY KEY,
    store_id INTEGER REFERENCES stores(store_id),
```

```
    date DATE,
    avg_rating DECIMAL(3, 2),
    total_reviews INTEGER,
    new_reviews INTEGER,
    competitor_avg_rating DECIMAL(3, 2),
    market_position_rank INTEGER,
    calculated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(store_id, date)
);
```

### Key Indexes for Performance

sql

```
-- Geographic queries
CREATE INDEX idx_stores_location ON stores USING GIST(ST_Point(longitude, latitude));
CREATE INDEX idx_competitors_location ON competitors USING GIST(ST_Point(longitude, latitude));

-- Time-series queries
CREATE INDEX idx_reviews_date ON reviews(review_date);
CREATE INDEX idx_daily_metrics_date ON daily_metrics(date);

-- Lookup optimization
CREATE INDEX idx_reviews_store_source ON reviews(store_id, source);
CREATE INDEX idx_pricing_store_item ON store_pricing(store_id, item_id);
```

---

## 🔄 Phase 6: Data Pipeline Development (Weeks 8-9)

*Estimated Time: 30-35 hours*

### Week 8: Basic Scrapers

**Components to Build**:

#### 1. Store Locator Scraper

python

```python
class SubwayStoreScraper:
    def __init__(self):
        self.session = requests.Session()
        self.base_url = "https://www.subway.com/locations"

    def get_stores_by_region(self, state, city=None):
        # Implementation for scraping store data
        pass

    def get_store_details(self, store_url):
        # Get detailed store information
        pass
```

### 2. Menu Scraper

```python
class SubwayMenuScraper:
    def scrape_current_menu(self):
        # Get current menu items and pricing
        pass

    def track_menu_changes(self):
        # Detect when menu items change
        pass
```

### 3. Reviews Scraper

```python
class ReviewsScraper:
    def scrape_google_reviews(self, store_id):
        # Google Reviews scraping
        pass

    def scrape_yelp_reviews(self, store_id):
        # Yelp reviews scraping
        pass
```

## Week 9: Pipeline Automation

**Learning Objectives**:

- [ ] Task scheduling with `schedule` or `APScheduler`
- [ ] Logging and monitoring
- [ ] Error recovery and retry logic
- [ ] Data validation and quality checks
- [ ] Pipeline orchestration

**Key Libraries**:

```python
import schedule
import logging
from apscheduler.schedulers.blocking import BlockingScheduler
```

**Pipeline Structure**:

```python
class SubwayDataPipeline:
    def __init__(self):
        self.db = DatabaseManager()
        self.store_scraper = SubwayStoreScraper()
        self.menu_scraper = SubwayMenuScraper()
        self.reviews_scraper = ReviewsScraper()

    def daily_update(self):
        # Run daily data updates
        pass

    def weekly_deep_scrape(self):
        # Comprehensive data refresh
        pass

    def validate_data(self):
        # Data quality checks
        pass
```

---

## 📈 Phase 7: Analytics & SQL Mastery (Weeks 10-11)

*Estimated Time: 25-30 hours*

### Advanced SQL Queries for Business Intelligence

### 1. Store Performance Analysis

sql

```sql
-- Top performing stores by region
WITH store_performance AS (
    SELECT
        s.store_id,
        s.city,
        s.state,
        AVG(r.rating) as avg_rating,
        COUNT(r.review_id) as total_reviews,
        PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY r.rating) as median_rating
    FROM stores s
    LEFT JOIN reviews r ON s.store_id = r.store_id
    WHERE r.review_date >= CURRENT_DATE - INTERVAL '90 days'
    GROUP BY s.store_id, s.city, s.state
)
SELECT
    city,
    state,
    COUNT(*) as store_count,
    AVG(avg_rating) as region_avg_rating,
    SUM(total_reviews) as region_total_reviews,
    RANK() OVER (ORDER BY AVG(avg_rating) DESC) as region_rank
FROM store_performance
GROUP BY city, state
ORDER BY region_avg_rating DESC;
```

### 2. Competitive Analysis

sql

```sql
-- Market position analysis
WITH market_analysis AS (
    SELECT
        s.store_id,
        s.city,
        s.state,
        AVG(s_ratings.avg_rating) as subway_rating,
        AVG(c.avg_rating) as competitor_avg_rating,
        COUNT(c.competitor_id) as competitor_count
    FROM stores s
    LEFT JOIN daily_metrics s_ratings ON s.store_id = s_ratings.store_id
    LEFT JOIN competitors c ON s.store_id = c.store_id
    WHERE s_ratings.date >= CURRENT_DATE - INTERVAL '30 days'
    GROUP BY s.store_id, s.city, s.state
)
SELECT
    *,
    subway_rating - competitor_avg_rating as competitive_advantage,
    CASE
        WHEN subway_rating > competitor_avg_rating THEN 'Leading'
        WHEN subway_rating = competitor_avg_rating THEN 'Competitive'
        ELSE 'Lagging'
    END as market_position
FROM market_analysis
ORDER BY competitive_advantage DESC;
```

### 3. Revenue Opportunity Analysis

```sql
```

```sql
-- Identify underperforming stores in high-potential areas
WITH demographic_opportunity AS (
    SELECT
        s.store_id,
        s.city,
        d.median_income,
        d.population,
        AVG(r.rating) as current_rating,
        COUNT(r.review_id) as review_volume,
        d.median_income * d.population / 1000000 as market_potential_score
    FROM stores s
    JOIN demographics d ON s.store_id = d.store_id
    LEFT JOIN reviews r ON s.store_id = r.store_id
    WHERE r.review_date >= CURRENT_DATE - INTERVAL '180 days'
    GROUP BY s.store_id, s.city, d.median_income, d.population
)
SELECT
    *,
    CASE
        WHEN current_rating < 4.0 AND market_potential_score > 50 THEN 'High Priority'
        WHEN current_rating < 4.0 AND market_potential_score > 25 THEN 'Medium Priority'
        ELSE 'Low Priority'
    END as improvement_priority
FROM demographic_opportunity
WHERE current_rating < 4.5  -- Focus on stores with room for improvement
ORDER BY market_potential_score DESC, current_rating ASC;
```

### 4. Menu Performance Tracking

```sql
```

```sql
-- Track regional menu preferences and pricing optimization
SELECT
    mi.name as menu_item,
    mi.category,
    s.state,
    AVG(sp.price) as avg_price,
    COUNT(DISTINCT s.store_id) as stores_offering,
    -- Calculate price elasticity indicators
    CORR(sp.price, dm.avg_rating) as price_rating_correlation
FROM menu_items mi
JOIN store_pricing sp ON mi.item_id = sp.item_id
JOIN stores s ON sp.store_id = s.store_id
JOIN daily_metrics dm ON s.store_id = dm.store_id
WHERE sp.effective_date <= CURRENT_DATE
    AND (sp.end_date IS NULL OR sp.end_date > CURRENT_DATE)
    AND dm.date >= CURRENT_DATE - INTERVAL '90 days'
GROUP BY mi.name, mi.category, s.state
HAVING COUNT(DISTINCT s.store_id) >= 5  -- Only include items in 5+ stores
ORDER BY price_rating_correlation DESC;
```

## Window Functions & Advanced Analytics

sql

```sql
-- Trend analysis with moving averages
SELECT
    store_id,
    date,
    avg_rating,
    AVG(avg_rating) OVER (
        PARTITION BY store_id
        ORDER BY date
        ROWS BETWEEN 29 PRECEDING AND CURRENT ROW
    ) as rating_30day_avg,
    LAG(avg_rating, 30) OVER (
        PARTITION BY store_id
        ORDER BY date
    ) as rating_30days_ago,
    CASE
        WHEN avg_rating > LAG(avg_rating, 30) OVER (PARTITION BY store_id ORDER BY date)
        THEN 'Improving'
        ELSE 'Declining'
    END as trend_direction
FROM daily_metrics
WHERE date >= CURRENT_DATE - INTERVAL '1 year'
ORDER BY store_id, date;
```

---

## 📊 Phase 8: Dashboard & Visualization (Week 12)

*Estimated Time: 20-25 hours*

### Dashboard Components

**Key Libraries**:

```python
import streamlit as st
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

#### 1. Executive Dashboard

- Overall performance KPIs
- Regional performance heatmaps

- Trend analysis charts
- Competitive positioning

### 2. Store-Level Analytics

- Individual store performance
- Customer satisfaction trends
- Local competitive analysis
- Demographic insights

### 3. Menu Analytics

- Item performance by region
- Pricing optimization recommendations
- Seasonal trend analysis

---

## 🚀 Phase 9: Advanced Features (Weeks 13-14)

*Estimated Time: 25-30 hours*

### Machine Learning Integration

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import joblib

class PerformancePredictor:
    def predict_store_rating(self, demographic_features, competitor_data):
        # Predict potential store performance
        pass

    def identify_expansion_opportunities(self):
        # Find optimal locations for new stores
        pass
```

### Recommendation Engine

- Underperforming store alerts

- Menu optimization suggestions
- Pricing strategy recommendations
- Market expansion opportunities

---

## 🗓️ Project Timeline Summary

| Phase | Duration | Key Deliverables |
|---|---|---|
| Phase 1-2 | Weeks 1-4 | Python foundation + Data handling |
| Phase 3 | Weeks 5-6 | Web scraping capabilities |
| Phase 4-5 | Week 7 | Database setup + Schema design |
| Phase 6 | Weeks 8-9 | Automated data pipeline |
| Phase 7 | Weeks 10-11 | Advanced SQL analytics |
| Phase 8 | Week 12 | Interactive dashboard |
| Phase 9 | Weeks 13-14 | ML features + Recommendations |

**Total Timeline: 14 weeks (3.5 months)**

---

## 🛠️ Technical Stack

**Languages & Core Libraries**:

- Python 3.9+
- PostgreSQL 13+
- HTML/CSS/JavaScript (for dashboard)

**Python Libraries**:

```bash
```

```
# Data handling
pandas>=1.3.0
numpy>=1.21.0

# Web scraping
requests>=2.26.0
beautifulsoup4>=4.10.0
selenium>=4.0.0
lxml>=4.6.0

# Database
psycopg2-binary>=2.9.0
SQLAlchemy>=1.4.0

# Automation & Scheduling
schedule>=1.1.0
APScheduler>=3.8.0

# Visualization
streamlit>=1.2.0
plotly>=5.3.0
folium>=0.12.0

# Machine Learning
scikit-learn>=1.0.0
joblib>=1.1.0

# Utilities
python-dotenv>=0.19.0
configparser>=5.0.0
logging>=0.4.9.6
```

**Development Tools**:

- Git for version control
- Docker for containerization
- pytest for testing
- Black for code formatting

---

## 📁 Project Structure

```
subway-bi-system/
├── config/
│   ├── config.ini
│   └── database_config.py
├── data/
│   ├── raw/
│   ├── processed/
│   └── exports/
├── src/
│   ├── scrapers/
│   │   ├── store_scraper.py
│   │   ├── menu_scraper.py
│   │   └── reviews_scraper.py
│   ├── database/
│   │   ├── models.py
│   │   ├── connection.py
│   │   └── queries.py
│   ├── pipeline/
│   │   ├── data_pipeline.py
│   │   ├── scheduler.py
│   │   └── validators.py
│   ├── analytics/
│   │   ├── performance_analyzer.py
│   │   ├── competitor_analyzer.py
│   │   └── ml_models.py
│   └── dashboard/
│       ├── app.py
│       ├── components/
│       └── utils/
├── sql/
│   ├── schema.sql
│   ├── indexes.sql
│   └── views.sql
├── tests/
├── logs/
├── requirements.txt
├── README.md
└── main.py
```

---

## 🎯 Success Metrics

**Technical Achievements**:

- [ ] Successfully scrape data from 1000+ Subway locations
- [ ] Maintain 99%+ data pipeline uptime
- [ ] Query response times under 2 seconds
- [ ] Dashboard loads under 5 seconds

**Business Value**:

- [ ] Identify top 10% and bottom 10% performing stores
- [ ] Generate actionable recommendations for underperformers
- [ ] Create competitive analysis for major markets
- [ ] Build predictive models with 80%+ accuracy

**Learning Objectives**:

- [ ] Master advanced SQL queries and optimization
- [ ] Build production-ready Python applications
- [ ] Understand end-to-end data pipeline development
- [ ] Create compelling data visualizations

---

## 📚 Additional Resources

**Books**:

- "Automate the Boring Stuff with Python" by Al Sweigart
- "Python for Data Analysis" by Wes McKinney
- "Web Scraping with Python" by Ryan Mitchell
- "Learning SQL" by Alan Beaulieu

**Online Courses**:

- freeCodeCamp Python courses
- Kaggle Learn (Python, Pandas, SQL)
- Real Python tutorials
- PostgreSQL Tutorial

**Documentation**:

- Python Documentation
- Pandas Documentation
- PostgreSQL Documentation
- Beautiful Soup Documentation

---

## ⚠️ Important Considerations

**Legal & Ethical**:

- Always respect robots.txt files
- Implement proper rate limiting
- Don't overload target servers
- Consider terms of service for each site
- Only scrape publicly available data

**Data Quality**:

- Implement comprehensive data validation
- Handle edge cases and missing data
- Regular data quality audits
- Backup strategies

**Performance**:

- Database indexing strategy
- Query optimization
- Caching frequently accessed data
- Horizontal scaling considerations

---

*This roadmap is your comprehensive guide to building a professional-grade business intelligence system. Focus on one phase at a time, and don't hesitate to spend extra time mastering concepts before moving forward. The key to success is consistent daily practice and building increasingly complex projects.*