Data Engineering and MLOps in Business

Code Refactoring and model monitoring

Primoz Konda

AAUBS

March 25, 2025

pk@business.aau.dk

Outline

Where did we end yesterday?

- ?
- Questions?

What is Code Refactoring?

Definition

Code refactoring is the process of restructuring existing code to improve its internal structure, without changing its external behavior.

Goals

To make the code easier to read, understand, and maintain, as well as to improve its performance, scalability, and reliability

Code Smell Example: Long Function

```python
def calculate_salary(employee_data):
    total_salary = 0
    for employee in employee_data:
        salary = employee['salary']
        if salary < 20000:
            bonus = 0.05 * salary
        elif salary < 50000:
            bonus = 0.1 * salary
        else:
            bonus = 0.15 * salary
        total_salary += salary + bonus
    tax = 0.2 * total_salary
    net_salary = total_salary - tax
    if net_salary < 15000:
        print("Warning: Net salary is too low!")
    return net_salary
```

Problems

- The function is too long and complex
- It performs multiple tasks at once (calculating salary, tax, and net salary)
- It mixes calculation and printing

Code Smell Example: Long Function

```python
def calculate_bonus(salary):
    if salary < 20000:
        return 0.05 * salary
    elif salary < 50000:
        return 0.1 * salary
    else:
        return 0.15 * salary

def calculate_total_salary(employee_data):
    total_salary = 0
    for employee in employee_data:
        salary = employee['salary']
        total_salary += salary + calculate_bonus(salary)
    return total_salary

def calculate_net_salary(total_salary):
    tax = 0.2 * total_salary
    net_salary = total_salary - tax
    if net_salary < 15000:
        print("Warning: Net salary is too low!")
    return net_salary
```

Code Smell Example: Long Function

```python
def calculate_bonus(salary):
    if salary < BONUS_THRESHOLD_1:
        return BONUS_RATE_1 * salary
    elif salary < BONUS_THRESHOLD_2:
        return BONUS_RATE_2 * salary
    else:
        return BONUS_RATE_3 * salary

def calculate_total_salary(employee_data):
    total_salary = 0
    for employee in employee_data:
        salary = employee['salary']
        total_salary += salary + calculate_bonus(salary)
    return total_salary

def calculate_net_salary(total_salary):
    tax = TAX_RATE * total_salary
    net_salary = total_salary - tax
    if net_salary < SALARY_WARNING:
        print("Warning: Net salary is too low!")
    return net_salary
```

Outside function:

$BONUS\_THRESHOLD\_1 = 20000$
$BONUS\_THRESHOLD\_2 = 50000$
$BONUS\_RATE\_1 = 0.05$
$BONUS\_RATE\_2 = 0.1$
$BONUS\_RATE\_3 = 0.15$
$TAX\_RATE = 0.2$
$SALARY\_WARNING = 15000$

We need to talk...

Do you think your code is well-written?

Code Smell types

Common types of code smell:

- Long functions
- Duplicate code
- Dead code
- Data Clumps
- Improper names

Code Smell types: Duplicate Code

```
x1 = 1
y1 = x1 * 2
z1 = y1 + 3

x2 = 2
y2 = x2 * 2
z2 = y2 + 3

x3 = 3
y3 = x3 * 2
z3 = y3 + 3
```

```
results = []
for i in range(1, 3):
    x = i
    y = x * 2
    z = y + 3
    results.append((x, y, z))
```

Code Smell types: Dead Code

It can be a function that is never called, a variable that is never used, or a conditional branch that is never taken.

```python
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b

result = add(2, 3)
```

```python
Age = int(input("Enter the age: "))
if Age >= 0 and Age <= 2:
    print("Person is an infant")
elif Age >= 3 and Age <= 18:
    print("Person is a child")
elif Age > 18:
    print("Person is an adult")
else:
    print("Person is less than 0 years old")
```

Code Smell types: Data Clumps

Data clumps occur when several data items are always found together.

```
def calculate_distance(x1, y1, x2, y2):
    return ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

def calculate_slope(x1, y1, x2, y2):
    return (y2 - y1) / (x2 - x1)

point1_x = 2
point1_y = 3
point2_x = 5
point2_y = 7

distance = calculate_distance(point1_x, point1_y, point2_x, point2_y)
slope = calculate_slope(point1_x, point1_y, point2_x, point2_y)
```

Code Smell types: Data Clumps

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])



def calculate_distance(point1, point2):
    return ((point2.x - point1.x) ** 2 + (point2.y - point1.y) ** 2) ** 0.5

def calculate_slope(point1, point2):
    return (point2.y - point1.y) / (point2.x - point1.x)

point1 = Point(2, 3)
point2 = Point(5, 7)

distance = calculate_distance(point1, point2)
slope = calculate_slope(point1, point2)
```

Code Smell types: Improper names

Improper naming of variables, classes, and functions can make the code harder to understand and maintain.

```
def f(x):
    return x * 2

y = 5
z = f(y)
```

```
def double(x):
    return x * 2

number = 5
result = double(number)
```

Reasons for Model Performance Degradation Over Time

- **Concept Drift:** Changes in the underlying data distribution that the model was not trained on, leading to reduced accuracy.
- **Data Drift:** Variations in input data characteristics over time, causing the model to misinterpret new data.
- **Model Drift:** The model's internal parameters may change due to factors like feedback loops or retraining on biased data.
- **Data Quality Issues:** Inconsistent, missing, or erroneous data can adversely affect model predictions.
- **Model Collapse:** Degradation resulting from training on synthetic data generated by previous models, leading to loss of diversity in predictions.

Monitoring Existing Models with New Data

- **Performance Metrics Tracking:** Continuously evaluate metrics such as accuracy, precision, recall, and F1-score to detect performance drops.
- **Data Drift Detection:** Implement statistical tests to identify shifts in input data distributions.
- **Prediction Drift Monitoring:** Compare model predictions over time to identify any significant changes.

Monitoring Existing Models with New Data (2)

- **Concept Drift Detection:** Utilize algorithms that can adapt to changes in data patterns, ensuring the model remains relevant.
- **Logging and Alerting Systems:** Set up robust logging to monitor API usage and establish alerts for anomalies or performance degradation.

Developing a Strategy for Model Monitoring

- **Define Clear Objectives:** Establish what aspects of model performance are critical to monitor based on business goals.
- **Select Appropriate Metrics:** Choose metrics that align with the model's intended use and business impact.
- **Implement Continuous Monitoring:** Set up systems to track model performance and data characteristics in real-time.

Developing a Strategy for Model Monitoring (2)

- **Establish Feedback Loops:** Create mechanisms to update the model based on monitoring insights, ensuring it adapts to new data patterns.
- **Automate Retraining Pipelines:** Develop automated processes for retraining models with new data to maintain performance.
- **Ensure Compliance and Transparency:** Maintain documentation and audit trails for model decisions, especially in regulated industries.

Example 1: Predicting Daily Energy Consumption

- **Prediction Task:** Forecasting daily energy consumption based on temporal factors such as time of day and temperature.
- **Data Description:** Historical energy usage data, including timestamps, temperature readings, and energy consumption values.

Example 2: Sports Betting Outcome Prediction

- **Prediction Task:** Forecasting the outcomes of sports matches, including predicting winners and point spreads, to inform betting strategies.
- **Data Description:** Comprehensive datasets comprising historical match results, betting odds, team statistics, player performance metrics, and other relevant features. For example, the "Beat The Bookie: Odds Series Football Dataset" offers 10 years of historical closing odds for over 479,000 football games across 818 leagues worldwide.

Example 3: Predicting Housing Prices

- **Prediction Task:** Estimating house prices based on various features such as the number of rooms, location, and other relevant factors.
- **Data Description:** A dataset with attributes like the number of rooms, location, age of the property, and sale price.

Example 4: Predicting Loan Approval

- **Prediction Task:** Determining the approval status of loan applications based on applicant information.
- **Data Description:** Applicant data including marital status, education level, number of dependents, employment status, and loan approval status.

Example 5: Detecting SMS Spam Messages

- **Prediction Task:** Classifying SMS messages as spam or not spam based on their content.
- **Data Description:** A collection of SMS messages labeled as 'spam' or 'ham' (non-spam), with features extracted from the text content.

Q & A

Questions?