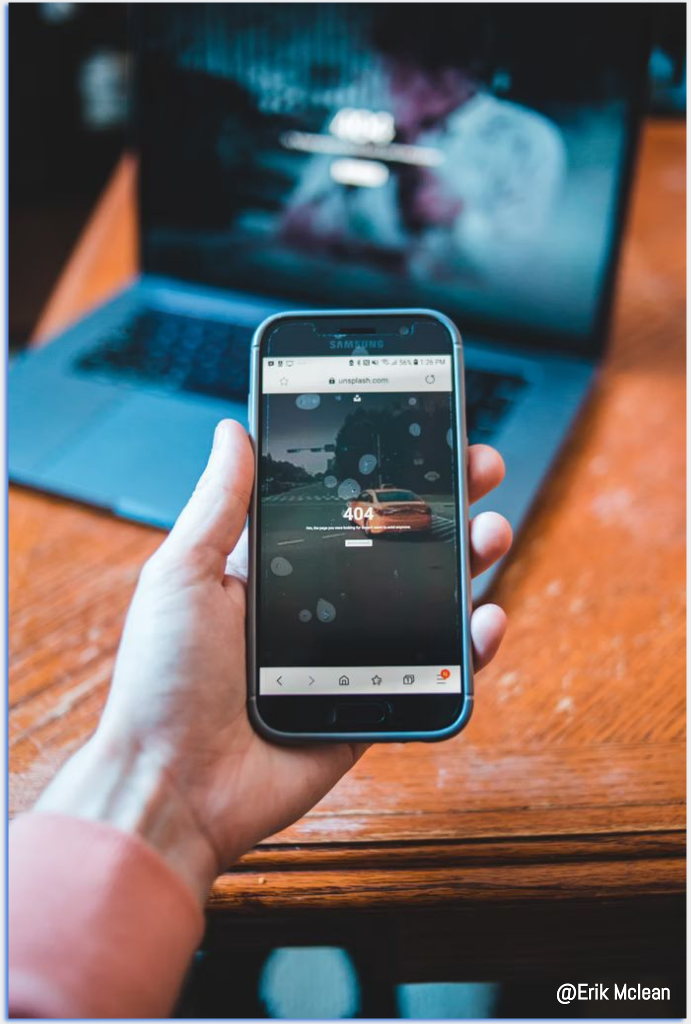


Production Ready Code

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE DCA

Table of Contents

01

Handling Errors

02

Writing Tests and Logs

03

Model drift

04

Automated vs . non-automated retraining

05

Further reading

"Debugging is hard, testing is easy"
"Far better an approximate answer to the *right question*, which is often vague, than an *exact* answer to the wrong question"- J. Tukey

Catching Errors

There are actually a lot of different considerations to errors

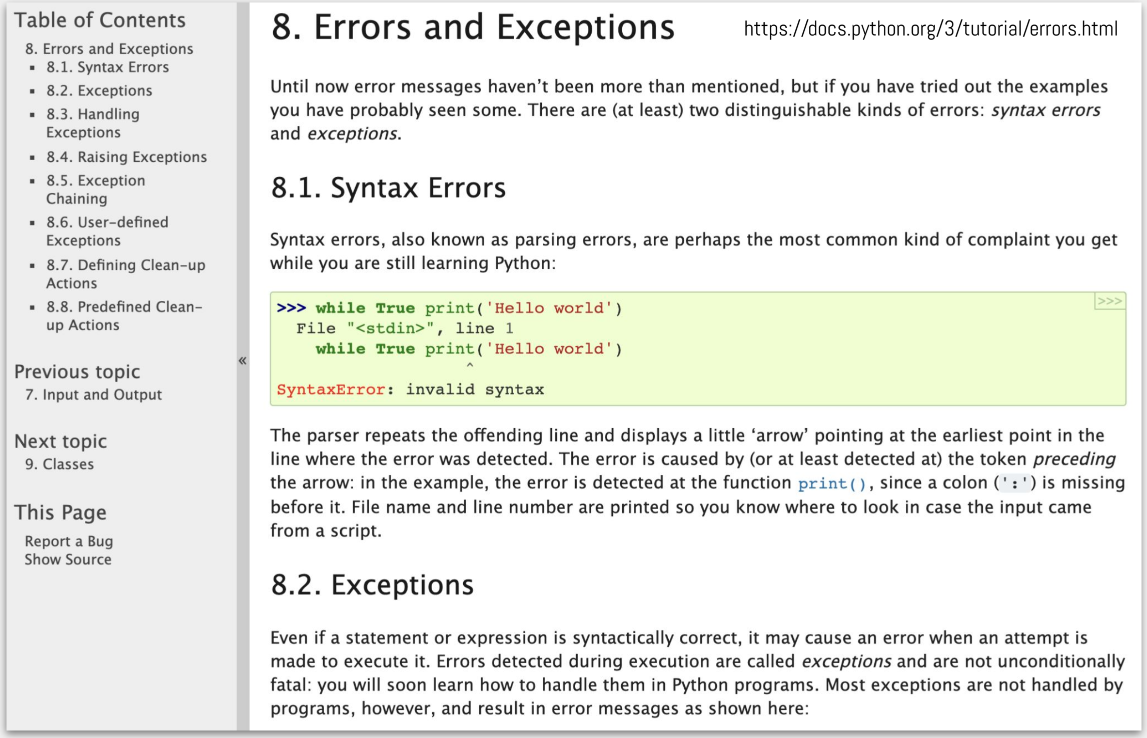


Table of Contents

8. Errors and Exceptions
- 8.1. Syntax Errors
- 8.2. Exceptions
- 8.3. Handling Exceptions
- 8.4. Raising Exceptions
- 8.5. Exception Chaining
- 8.6. User-defined Exceptions
- 8.7. Defining Clean-up Actions
- 8.8. Predefined Clean-up Actions

Previous topic

7. Input and Output

Next topic

9. Classes

This Page

Report a Bug
Show Source

8. Errors and Exceptions

https://docs.python.org/3/tutorial/errors.html

Until now error messages haven't been more than mentioned, but if you have tried out the examples you have probably seen some. There are (at least) two distinguishable kinds of errors: *syntax errors* and *exceptions*.

8.1. Syntax Errors

Syntax errors, also known as parsing errors, are perhaps the most common kind of complaint you get while you are still learning Python:

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

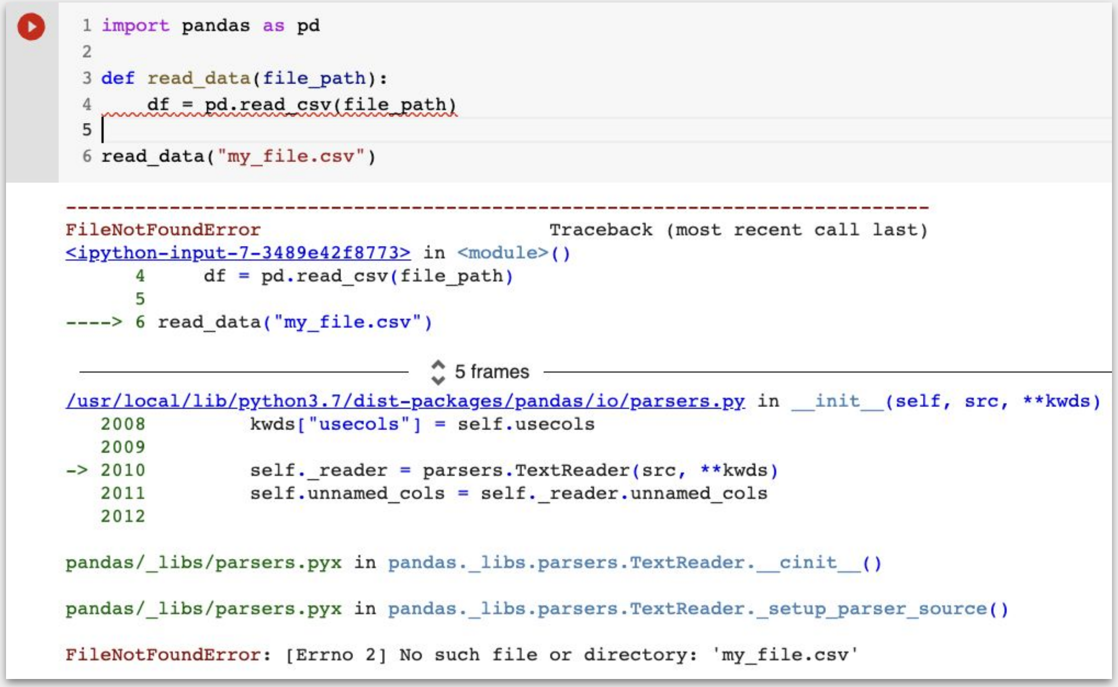
The parser repeats the offending line and displays a little 'arrow' pointing at the earliest point in the line where the error was detected. The error is caused by (or at least detected at) the token *preceding* the arrow: in the example, the error is detected at the function `print()`, since a colon (`':'`) is missing before it. File name and line number are printed so you know where to look in case the input came from a script.

8.2. Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called *exceptions* and are not unconditionally fatal: you will soon learn how to handle them in Python programs. Most exceptions are not handled by programs, however, and result in error messages as shown here:

Catching Errors

Using try and except blocks in order to catch errors



```python
import pandas as pd

def read_data(file_path):
    df = pd.read_csv(file_path)

read_data("my_file.csv")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-7-3489e42f8773> in <module>()
      4     df = pd.read_csv(file_path)
      5
----> 6 read_data("my_file.csv")

5 frames
/usr/local/lib/python3.7/dist-packages/pandas/io/parsers.py in __init__(self, src, **kwds)
   2008         kwds["usecols"] = self.usecols
   2009
-> 2010         self._reader = parsers.TextReader(src, **kwds)
   2011         self.unnamed_cols = self._reader.unnamed_cols
   2012

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source()

FileNotFoundError: [Errno 2] No such file or directory: 'my_file.csv'
```

```python
import pandas as pd

def read_data(file_path):
   try:
       df = pd.read_csv(file_path)
   except:
       print("There is no such {}".format(file_path))
read_data("my_file.csv")
```

```
There is no such my_file.csv
```

Exercise!!!

Testing and Data Science

Testing your code is essential before deployment. It helps you catch errors and faulty conclusions before they make any major impact.

- **Test-driven development (TDD)**: A development process in which you write tests for tasks before you even write the code to implement those tasks.
- **Unit test:** A type of test that covers a "unit" of code—usually a single function—independently from the rest of the program.

Testing and Data Science

Unit Tests - Advantages vs Disadvantages

We want to test our functions in a way that is repeatable and automated. Ideally, we'd run a test program that runs all our unit tests and cleanly lets us know which ones failed and which ones succeeded.

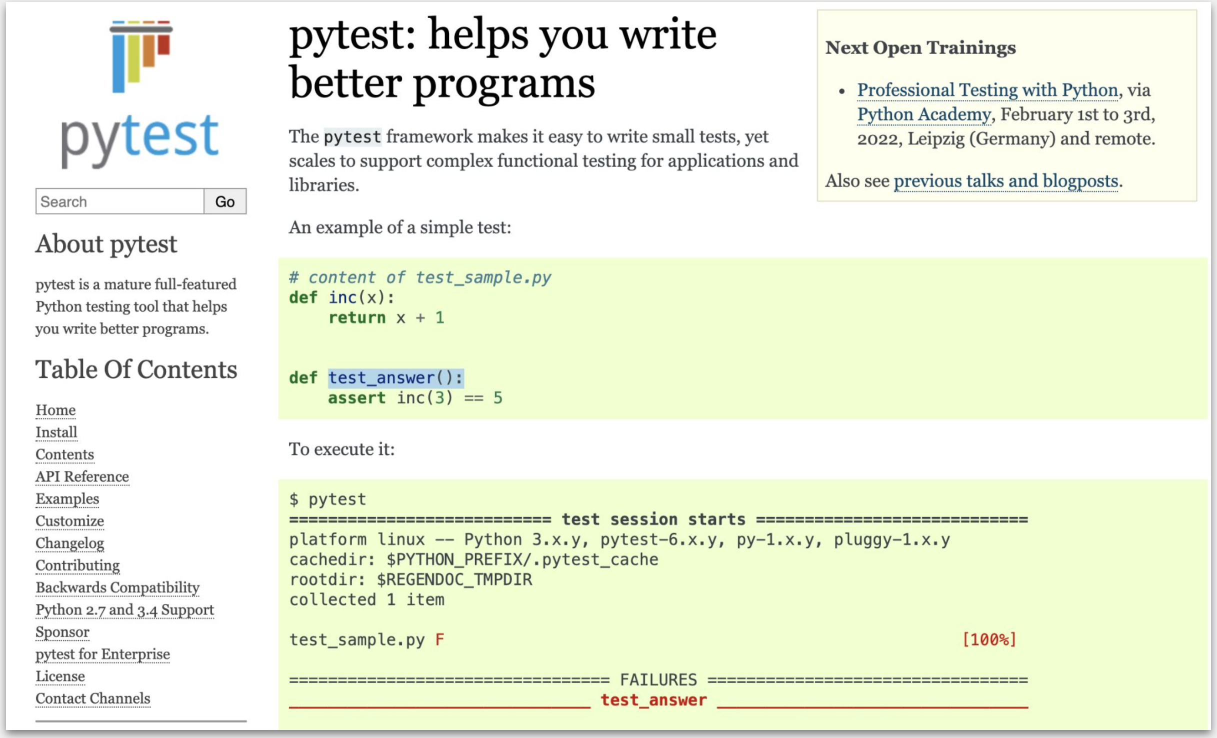
```python
def nearest_square(num):
    """
    Return the nearest perfect square that is
    less than or equal to num
    """
    root = 0
    while (root + 1) ** 2 <= num:
        root += 1
    return root ** 2
```

```python
print("Nearest square <= 5: returned {}, correct answer is 4".format(nearest_square(5)))
print("Nearest square <= -12: returned {}, correct answer is 0".format(nearest_square(-12)))
print("Nearest square <= 9: returned {}, correct answer is 9".format(nearest_square(9)))
print("Nearest square <= 23: returned {}, correct answer is 16".format(nearest_square(23)))
```

```
Nearest square <= 5: returned 4, correct answer is 4
Nearest square <= -12: returned 0, correct answer is 0
Nearest square <= 9: returned 9, correct answer is 9
Nearest square <= 23: returned 16, correct answer is 16
```

Testing and Data Science

Pytest - create effective unit tests



pytest

Search Go

About pytest

pytest is a mature full-featured Python testing tool that helps you write better programs.

Table Of Contents

Home
Install
Contents
API Reference
Examples
Customize
Changelog
Contributing
Backwards Compatibility
Python 2.7 and 3.4 Support
Sponsor
pytest for Enterprise
License
Contact Channels

pytest: helps you write better programs

The `pytest` framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

Next Open Trainings

- Professional Testing with Python, via Python Academy, February 1st to 3rd, 2022, Leipzig (Germany) and remote.

Also see previous talks and blogposts.

An example of a simple test:

```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

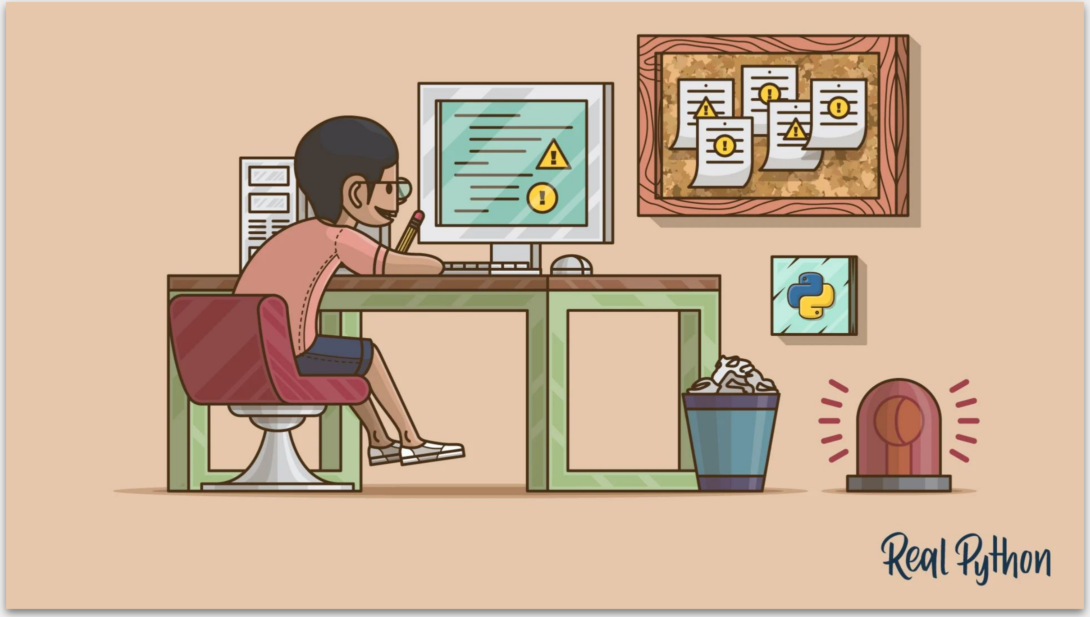
To execute it:

```
$ pytest
=========================== test session starts ============================
platform linux -- Python 3.x.y, pytest-6.x.y, py-1.x.y, pluggy-1.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 1 item

test_sample.py F                                                     [100%]

================================= FAILURES =================================
_______________________________ test_answer ________________________________
```

```
pip -U install pytest pytest-sugar
```

Logging

Understanding the events that occur while running your program



https://realpython.com/python-logging

```python
import logging

logging.basicConfig(
    filename='./results.log',
    level=logging.INFO,
    filemode='w',
    format='%(name)s - %(levelname)s - %(message)s')
```

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

Testing & Logging

assert + try/except + logging + pylint + autopep8

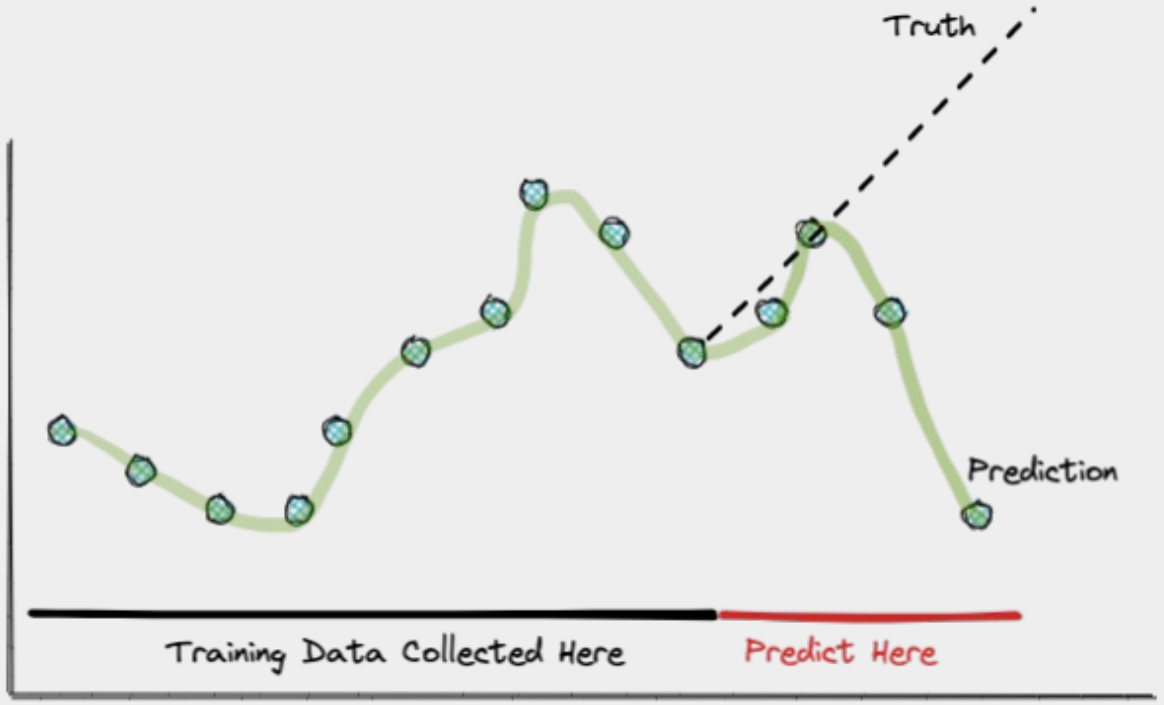
```
def divide_vals(numerator, denominator):
   '''
   Args:
       numerator: (float) numerator of fraction
       denominator: (float) denominator of fraction

   Returns:
       fraction_val: (float) numerator/denominator
   '''
   fraction_val = numerator / denominator
   return fraction_val
```

```
def num_words(text):
   '''
   Args:
       text: (string) string of words

   Returns:
       num_words: (int) number of words in the string
   '''
   total_words = len(text.split())
   return total_words
```

What happens after deployments?

"Change is the only constant in life" Heraclitus.

Model Drift

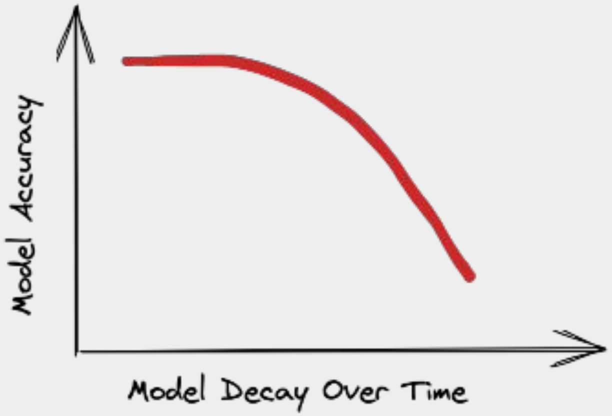


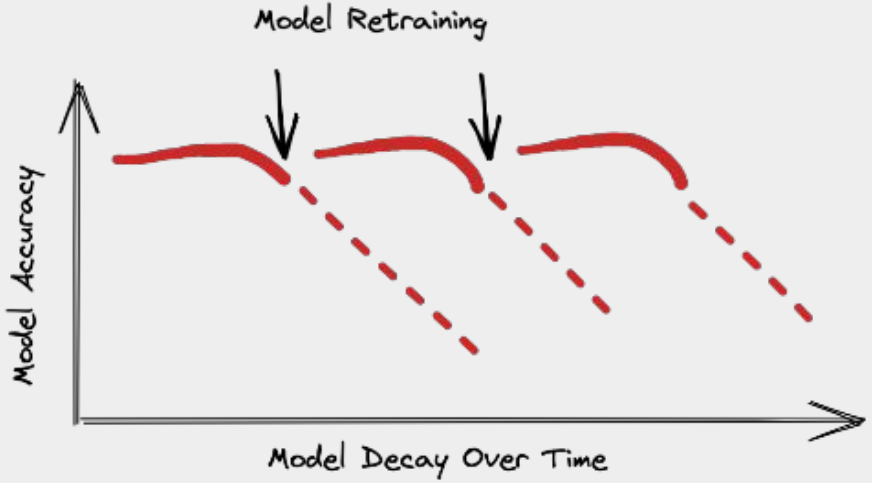
When deploying models into production, it is often the case that the input data changes over time. This shift means that our models may not perform as well over time as they did when the model was originally launched. This process of the model performance degrading over time is known as **model drift**.

Types of change to worry about in ML and DS

Concept Drift	Statistical properties of target variable change (i.e. what you are trying to predict changes) E.g. fraud detection
Data Drift	Statistical properties of input variable change E.g. seasonality, personal preferences, trends change
Upstream Data Changes	Encoding of a feature changes (e.g. switch from Fahrenheit to Celsius) Features are no longer being generated (leads to missing values)

Why you should care about model drift?

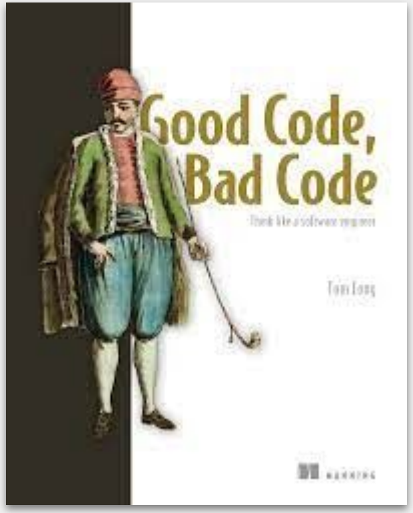




Lesson Conclusion: Production-Ready Code

With this lesson, you are ready to put to practice 5 key areas for launching production-ready machine learning models:

1. Catching errors
2. Writing tests
3. Writing logs
4. Model drift
5. Automated vs. non-automated retraining



https://www.manning.com/books/good-code-bad-code