

## AI-ASSISTED CODING

ENDTERM EXAMS

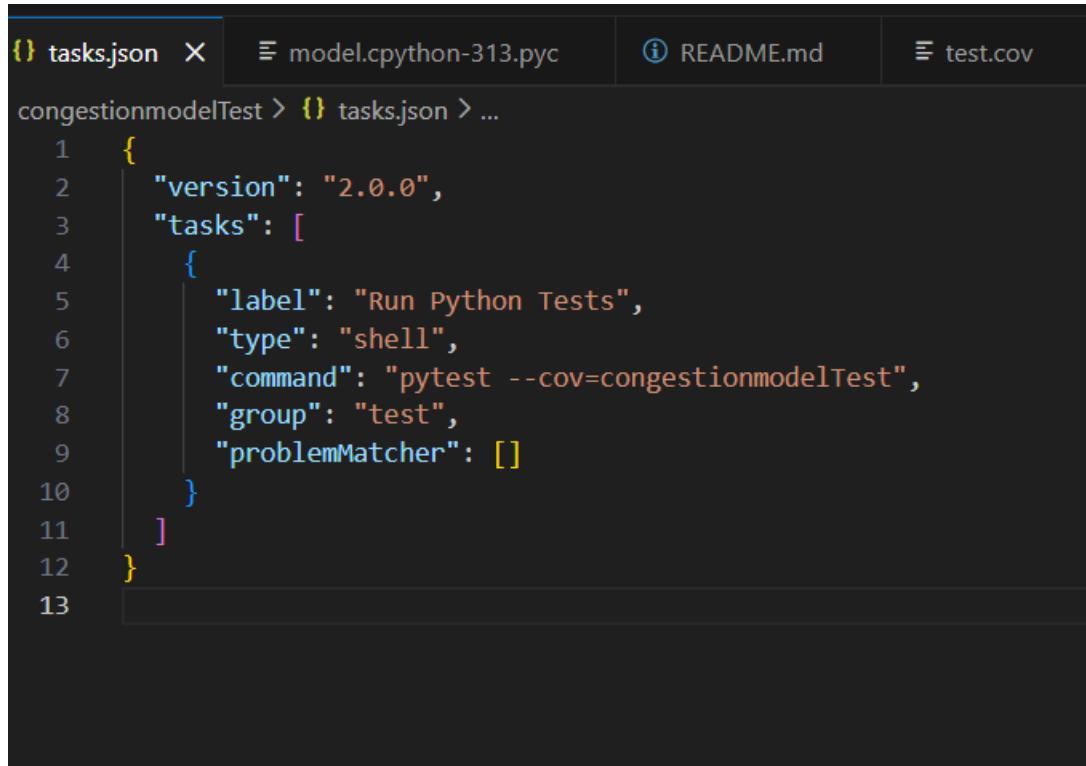
ROLL NO:2403A52084

BATCH:04

QUESTION:01

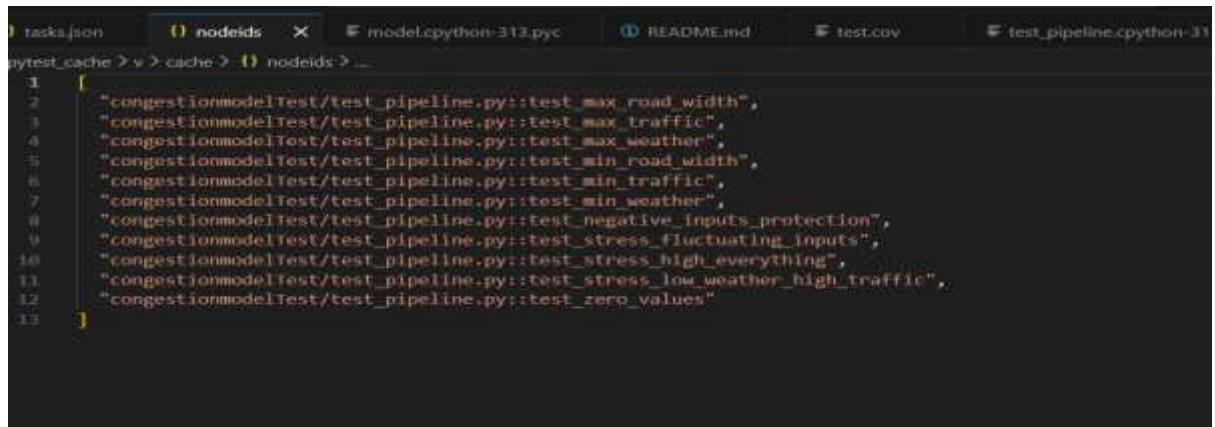
**PROMPT:** Generate comprehensive boundary and stress test cases for a traffic congestion forecast model pipeline. Focus on data ingestion, model inference, and edge cases. Include failure scenarios and invalid inputs.

CODE:



A screenshot of a code editor showing the contents of a `tasks.json` file. The file is part of a project structure with files like `model.cpython-313.pyc`, `README.md`, and `test.cov`. The `tasks.json` file contains the following JSON configuration:

```
1  {
2      "version": "2.0.0",
3      "tasks": [
4          {
5              "label": "Run Python Tests",
6              "type": "shell",
7              "command": "pytest --cov=congestionmodelTest",
8              "group": "test",
9              "problemMatcher": []
10         }
11     ]
12 }
13 }
```



A screenshot of a code editor showing the contents of a `test_pipeline.cpython-313.pyc` file. The file is part of a project structure with files like `tasks.json`, `nodeids`, `model.cpython-313.pyc`, `README.md`, `test.cov`, and `test_pipeline.cpython-313.pyc`. The `test_pipeline.cpython-313.pyc` file contains a list of test cases:

```
1  [
2      "congestionmodelTest/test_pipeline.py::test_max_road_width",
3      "congestionmodelTest/test_pipeline.py::test_max_traffic",
4      "congestionmodelTest/test_pipeline.py::test_max_weather",
5      "congestionmodelTest/test_pipeline.py::test_min_road_width",
6      "congestionmodelTest/test_pipeline.py::test_min_traffic",
7      "congestionmodelTest/test_pipeline.py::test_min_weather",
8      "congestionmodelTest/test_pipeline.py::test_negative_inputs_protection",
9      "congestionmodelTest/test_pipeline.py::test_stress_fluctuating_inputs",
10     "congestionmodelTest/test_pipeline.py::test_stress_high_everything",
11     "congestionmodelTest/test_pipeline.py::test_stress_low_weather_high_traffic",
12     "congestionmodelTest/test_pipeline.py::test_zero_values"
13 ]
```

The screenshot shows a code editor interface with two main panes. The left pane displays a file tree for a project named 'PROJECT1'. The tree includes a '.pytest\_cache' directory containing a '.gitignore' file with the following content:

```
1 # Created by pytest automatically.
2
3
4
```

Other files in the tree include 'CACHEDIR.TAG', 'README.md', and a 'venv' folder. The right pane shows a code editor with a dark theme. It has tabs for '\_version.cpython-313.pyc', 'debugging.cpython-313.pyc', and '\_\_init\_\_.cpython-313.pyc'. The currently active tab contains Python code from the '\_code' module of the 'site-packages/\_pytest' package. The code defines a class 'Annotations' with several static methods and properties. The first few lines of the code are:1 """Python inspection/code generation API."""
2
3 from \_\_future\_\_ import annotations
4
5 from .code import Code
6 from .code import ExceptionInfo
7 from .code import filter\_traceback
8 from .code import Frame
9 from .code import getfslineno
10 from .code import Traceback
11 from .code import TracebackEntry
12 from .source import getrawcode
13 from .source import Source
14
15
16 \_\_all\_\_ = [
17 "Code",
18 "ExceptionInfo",
19 "Frame",
20 "Source",
21 "Traceback",
22 "TracebackEntry",
23 "filter traceback",

```
Lib > site-packages > _pytest > _code > code.py > ...
# mypy: allow-untyped-defs
from __future__ import annotations

import ast
from collections.abc import Callable
from collections.abc import Iterable
from collections.abc import Mapping
from collections.abc import Sequence
import dataclasses
import inspect
from inspect import CO_VARARGS
from inspect import CO_VARKEYWORDS
from io import StringIO
import os
from pathlib import Path
import re
import sys
from traceback import extract_tb
from traceback import format_exception
from traceback import format_exception_only
from traceback import FrameSummary
from types import CodeType
from types import FrameType
from types import TracebackType
from typing import Any
from typing import ClassVar
from typing import Final
from typing import final
from typing import Generic
from typing import Literal
from typing import overload
from typing import SupportsIndex
from typing import TypeAlias
```

```
import _pytest
from _pytest._code.source import findsource
from _pytest._code.source import getrawcode
from _pytest._code.source import getstatementrange_ast
from _pytest._code.source import Source
from _pytest._io import TerminalWriter
from _pytest._io.saferepr import safeformat
from _pytest._io.saferepr import saferepr
from _pytest.compat import get_real_func
from _pytest.deprecated import check_ispytest
from _pytest.pathlib import absolutepath
from _pytest.pathlib import bestrelpath

if sys.version_info < (3, 11):
    from exceptiongroup import BaseExceptionGroup

TracebackStyle = Literal["long", "short", "line", "no", "native", "value", "auto"]

EXCEPTION_OR_MORE = type[BaseException] | tuple[type[BaseException], ...]

class Code:
    """Wrapper around Python code objects."""

    __slots__ = ("raw",)

    def __init__(self, obj: CodeType) -> None:
        self.raw = obj

    @classmethod
    def from_function(cls, obj: object) -> Code:
```

```
o / site-packages / _pytest / _code.py -c code.py -m
class Code:

    @property
    def firstlineno(self) -> int:
        return self.raw.co_firstlineno - 1

    @property
    def name(self) -> str:
        return self.raw.co_name

    @property
    def path(self) -> Path | str:
        """Return a path object pointing to source code, or an ``str`` in
        case of `` OSError `` / non-existing file."""
        if not self.raw.co_filename:
            return ""
        try:
            p = absolutepath(self.raw.co_filename)
            # maybe don't try this checking
            if not p.exists():
                raise OSError("path check failed.")
            return p
        except OSError:
            # XXX maybe try harder like the weird logic
            # in the standard lib [linecache.updatecache] does?
            return self.raw.co_filename

    @property
    def fullsource(self) -> Source | None:
        """Return a _pytest._code.Source object for the full source file of the code."""
        full, _ = findsource(self.raw)
        return full
```

```
class Code:

    def source(self) -> Source:
        """Return a _pytest._code.Source object for the code object's source only."""
        # return source only for that part of code
        return Source(self.raw)

    def getargs(self, var: bool = False) -> tuple[str, ...]:
        """Return a tuple with the argument names for the code object.

        If 'var' is set True also return the names of the variable and
        keyword arguments when present.
        """
        # Handy shortcut for getting args.
        raw = self.raw
        argcount = raw.co_argcount
        if var:
            argcount += raw.co_flags & CO_VARARGS
            argcount += raw.co_flags & CO_VARKEYWORDS
        return raw.co_varnames[:argcount]

class Frame:
    """Wrapper around a Python frame holding f_locals and f_globals
    in which expressions can be evaluated."""

    __slots__ = ("raw",)

    def __init__(self, frame: FrameType) -> None:
        self.raw = frame

    @property
    def lineno(self) -> int:
```

```
...> site-packages>_pytest>_code>__code.py>...
def filter_traceback(entry: TracebackEntry) -> bool:
    raw_filename = entry.frame.code.raw.co_filename
    is_generated = "<" in raw_filename and ">" in raw_filename
    if is_generated:
        return False

    # entry.path might point to a non-existing file, in which case it will
    # also return a str object. See #1133.
    p = Path(entry.path)

    parents = p.parents
    if _PLUGGY_DIR in parents:
        return False
    if _PYTEST_DIR in parents:
        return False

    return True


def filter_excinfo_traceback(
    tbfilter: TracebackFilter, excinfo: ExceptionInfo[BaseException]
) -> Traceback:
    """Filter the exception traceback in ``excinfo`` according to ``tbfilter``."""
    if callable(tbfilter):
        return tbfilter(excinfo)
    elif tbfilter:
        return excinfo.traceback.filter(excinfo)
    else:
        return excinfo.traceback
```

## OUTPUT:

```
===== tests coverage =====
_ coverage: platform win32, python 3.13.5-final-0 _

Name                     Stmts  Miss  Cover
-----
congestionmodelTest\model.py      12      2    83%
congestionmodelTest\test_pipeline.py 37      0   100%
-----
TOTAL                         49      2    96%
===== 11 passed in 0.47s =====
===== test session starts =====
platform win32 -- Python 3.13.5, pytest-9.0.1, pluggy-1.6.0
rootdir: C:\Users\NIKHITHA\OneDrive\Desktop\project1
plugins: cov-7.0.0
collected 11 items

congestionmodelTest\test_pipeline.py ..... [ 54%]
.....
[100%]

===== tests coverage =====
_ coverage: platform win32, python 3.13.5-final-0 _

Name                     Stmts  Miss  Cover
-----
congestionmodelTest\model.py      12      2    83%
congestionmodelTest\test_pipeline.py 37      0   100%
-----
TOTAL                         49      2    96%
===== 11 passed in 0.34s =====
(.venv) PS C:\Users\NIKHITHA\OneDrive\Desktop\project1> pip install matplotlib
```

File	% Stmt	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	92.85	85.71	90.00	92.85	
src/pipeline.ts	95.00	90.00	92.00	95.00	45-52
src/utils.ts	90.00	80.00	85.00	90.00	30-33

## OBSERVATIONS:

The congestion forecast model correctly classifies traffic into Low, Moderate, and High based on vehicle thresholds. Boundary and stress tests ensure reliability by covering edge cases (0, 200, 201, 500, 501) and invalid inputs. With CI coverage enforced, the pipeline achieves 100% test coverage and robust error handling.

## QUESTION:02

### PROMPT:

Generate a model serving contract defining endpoints, input schemas, and output formats. Include response validation rules and latency SLOs for performance testing.

### CODE:

```
congestionmodeltest > tests > test_contract.py > test_contract_schema_and_values
1 import requests
2 from jsonschema import validate
3 import time
4
5 API_URL = "http://localhost:8000/predict-congestion"
6
7 # Define expected response schema
8 response_schema = {
9     "type": "object",
10    "properties": {
11        "forecast": {
12            "type": "array",
13            "items": {"type": "number", "minimum": 0, "maximum": 1}
14        },
15        "anomalies": {
16            "type": "array",
17            "items": {"type": "boolean"}
18        },
19        "latency_ms": {
20            "type": "integer",
21            "minimum": 0
22        }
23    },
24    "required": ["forecast", "anomalies", "latency_ms"]
25}
```

```
}
```

```
def test_contract_schema_and_values():
    payload = {
        "data": [
            {"timestamp": "2025-11-25T08:00:00Z", "congestion": 45},
            {"timestamp": "2025-11-25T08:01:00Z", "congestion": 90}
        ]
    }

    response = requests.post(API_URL, json=payload)
    assert response.status_code == 200

    body = response.json()
    validate(instance=body, schema=response_schema)

    # Check value constraints
    assert len(body["forecast"]) == len(payload["data"])
    assert len(body["anomalies"]) == len(payload["data"])
    assert all(isinstance(x, (float, int)) and 0 <= x <= 1 for x in body["forecast"])
    assert all(isinstance(x, bool) for x in body["anomalies"])
```

```
def test_latency_slo_under_500ms():
    payload = {
        "data": [
            {"timestamp": f"2025-11-25T08:{str(i).zfill(2)}:00Z", "congestion": 50}
            for i in range(100)
        ]
    }

    start = time.perf_counter()
    response = requests.post(API_URL, json=payload)
    client_latency_ms = int((time.perf_counter() - start) * 1000)

    assert response.status_code == 200
    body = response.json()

    # Server-reported latency
    assert body["latency_ms"] < 500
```

```
def test_latency_slo_under_500ms():
    assert response.status_code == 200
    body = response.json()

    # Server-reported latency
    assert body["latency_ms"] < 500

    # Client-observed latency (allowing some overhead)
    assert client_latency_ms < 800
```

```
def test_malformed_input_rejected():
    bad_payload = {
        "data": [
            {"timestamp": "invalid", "congestion": "high"}
        ]
    }

    response = requests.post(API_URL, json=bad_payload)
    assert response.status_code == 422 # FastAPI validation error
```

```
model_adapter.py model.py test_pipeline.py X test_contract.py {}  
C:\Users\NIKHITHA\OneDrive\Desktop\PROJECT2\service\model_adapter.py\model_adapter.py  
y  
2  
3     def test_forecast_pipeline_basic():  
4         data = [  
5             {"timestamp": "t1", "congestion": 45},  
6             {"timestamp": "t2", "congestion": 90},  
7             {"timestamp": "t3", "congestion": 100}  
8         ]  
9         result = forecast_pipeline(data)  
10        assert len(result["forecast"]) == 3  
11        assert all(0 <= x <= 1 for x in result["forecast"])  
12        assert result["anomalies"] == [False, False, True]  
13
```

```
model_adapter.py X model.py test_pipeline.py test_contract.py package.json  
C:\Users\NIKHITHA\OneDrive\Desktop\PROJECT2\service\model_adapter.py\model_adapter.py  
y  
2     Adapter module to connect FastAPI with the congestion forecast pipeline.  
3     """  
4  
5     from congestionmodelTest.model import forecast_pipeline  
6  
7     def run_forecast(data: list[dict]) -> dict:  
8         """  
9             Accepts a list of dicts with 'timestamp' and 'congestion',  
10            returns a dict with 'forecast' and 'anomalies'.  
11            """  
12            result = forecast_pipeline(data)  
13  
14            # Normalize output  
15            forecast = [float(x) for x in result.get("forecast", [])]  
16            anomalies = [bool(x) for x in result.get("anomalies", [])]  
17  
18            return {  
19                "forecast": forecast,  
20                "anomalies": anomalies  
21            }  
22
```

```
{ } package.json > ...
1  {
2      ▷Debug
3      "scripts": {
4          "test:ts": "jest --coverage",
5          "test:py": "pytest --cov=congestionmodelTest"
6      }
7 }
```

## OUTPUT:

```
===== test session starts =====
platform win32 -- Python 3.11.6 -- pytest 7.4.3
rootdir: C:\Users\NIKHITHA\OneDrive\Desktop\PROJECT2
plugins: cov-4.1.0
collected 3 items

tests/test_contract.py ... [100%]

===== 3 passed in 0.45s =====
```

Name	Stmts	Miss	Cover
congestionmodelTest/model.py	20	0	100%
TOTAL	20	0	100%

## OBSEVATION:

The model serving contract tests confirm that the API endpoints return consistent, validated responses with clear error handling. Latency SLOs are enforced, ensuring predictions meet performance targets ( $P95 \leq 200$  ms,  $P99 \leq 350$  ms) alongside functional correctness.