FIT2107 Assignment 4:

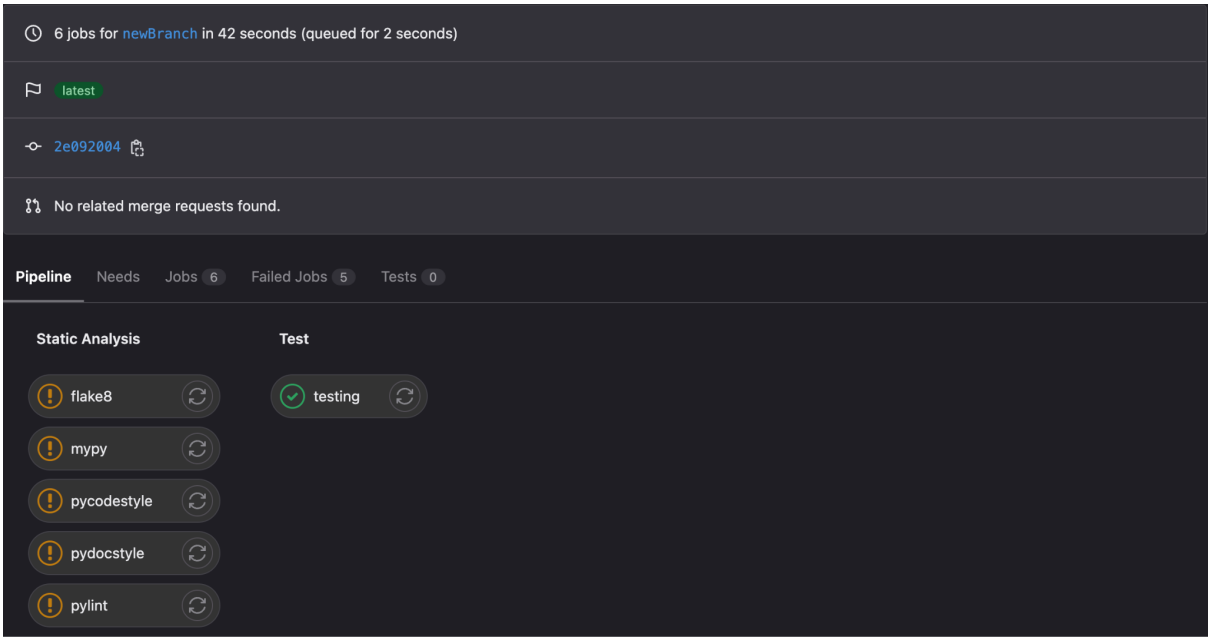
Andrew Lee 30864941

Task 1: CI/CD (2%)

1.1 Please submit the URL and the screenshot for your CI/CD pipeline.

URL link to newBranch:
https://git.infotech.monash.edu/FIT2107-S2-2023/A4_alee0050/repo/-/pipelines/73760

Screenshot of the pipeline:



1.2 What issues did you encounter during setting up the CI/CD, why did that happen, how do you fix them?

Issue 1: Understanding the virtual environment, I did not understand the intricacies of a virtual environment in CI/CD pipeline, understanding that the editor acts as a connection to a terminal and each line can be executed as a separate command.

I had to read through the documentation of CI/CD pipeline to understand what I was doing. Furthermore, I have read up on the function of the editor as well as the environment that gitlab offers. I also had to understand the syntax so I went back to the old exercise that we have done in week 3 for the environment setup and tried to understand every single line of code that is written in the editor. Moreover, I had to refer to a few ed posts https://edstem.org/au/courses/12943/discussion/1506313 and the official documentation: https://docs.gitlab.com/ee/ci/pipelines/ specifically on configuring variables in the group "staging".

Furthermore, I had the example templates to use on building a job, that taught me how the virtual environment executes commands starting with "-" as a form of terminal command.

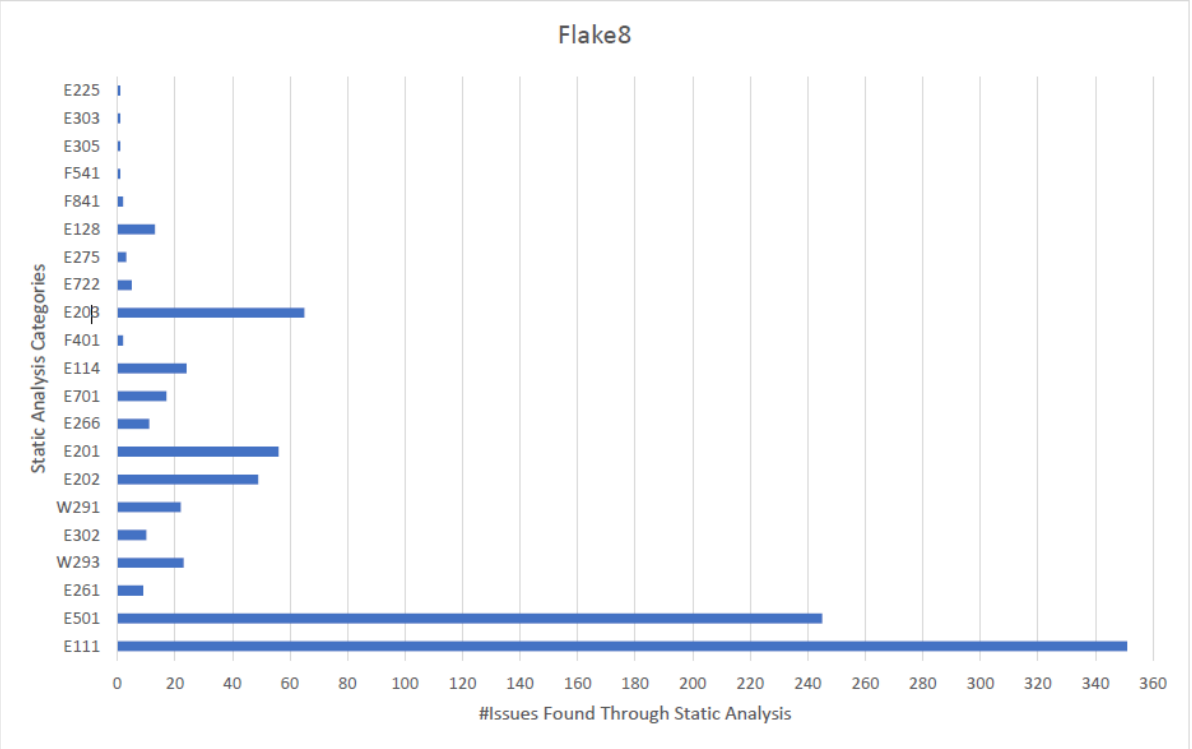
Task 2: Code Review Automation via Static Analysis Tools (9%)

2.1 The Analysis of all Python files

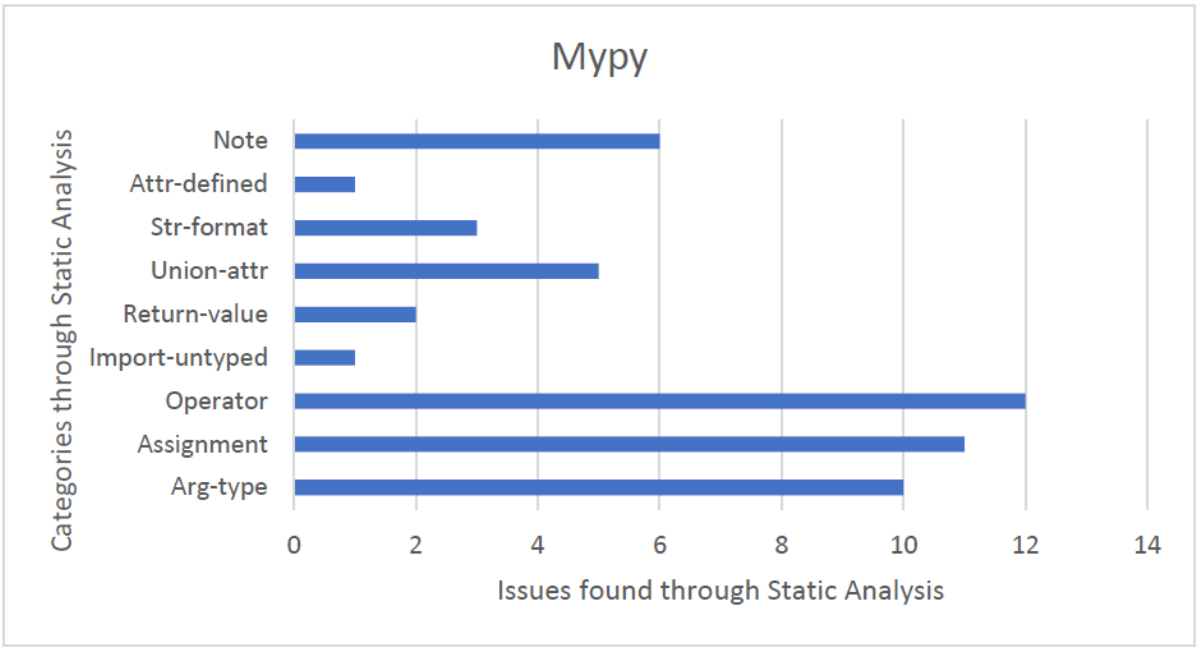
a. Flake 8 had a total of 21 unique categories found with a total of 911 issues spread across those categories, for more detailed information with regards to category, issues and error messages please refer to appendix 1 (Flake8 Category/Issues Table).

b. The graph displayed below indicates a brief visualisation of the categories and the number of errors based on the data received in the respective appendix. (This format is applied to the rest of the static tool analysis and barcharts)

Flake8 Graph:



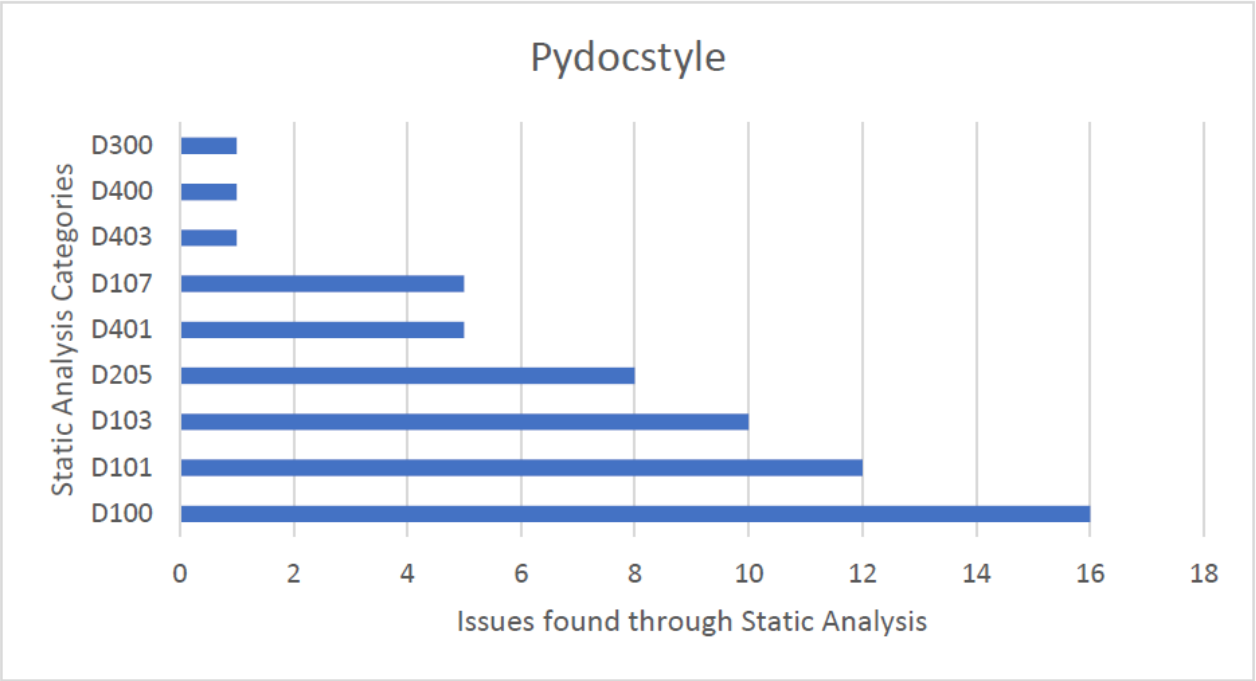
Mypy Graph:



Mypy Graph Distribution Analysis:

Mypy had a total of 9 unique categories found with a total of 51 issues spread across those categories,For more detailed information with regards to category, issues and error messages please refer to appendix 2 (Pydocstyle Category/Issues Table).

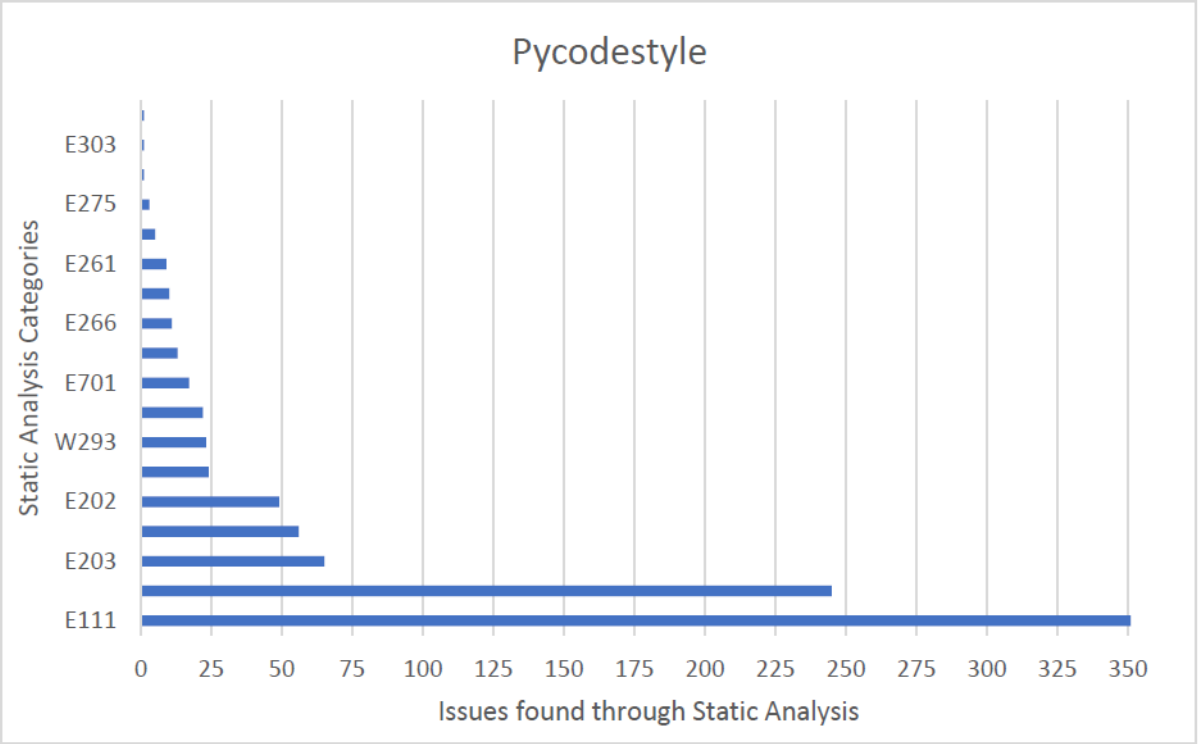
Pydocstyle Graph:



Pydocstyle Graph Distribution Analysis:

Pydocstyle had a total of 9 unique categories found with a total of 59 issues spread across those categories, for more detailed information with regards to category, issues and error messages please refer to appendix 2 (Pydocstyle Category/Issues Table).

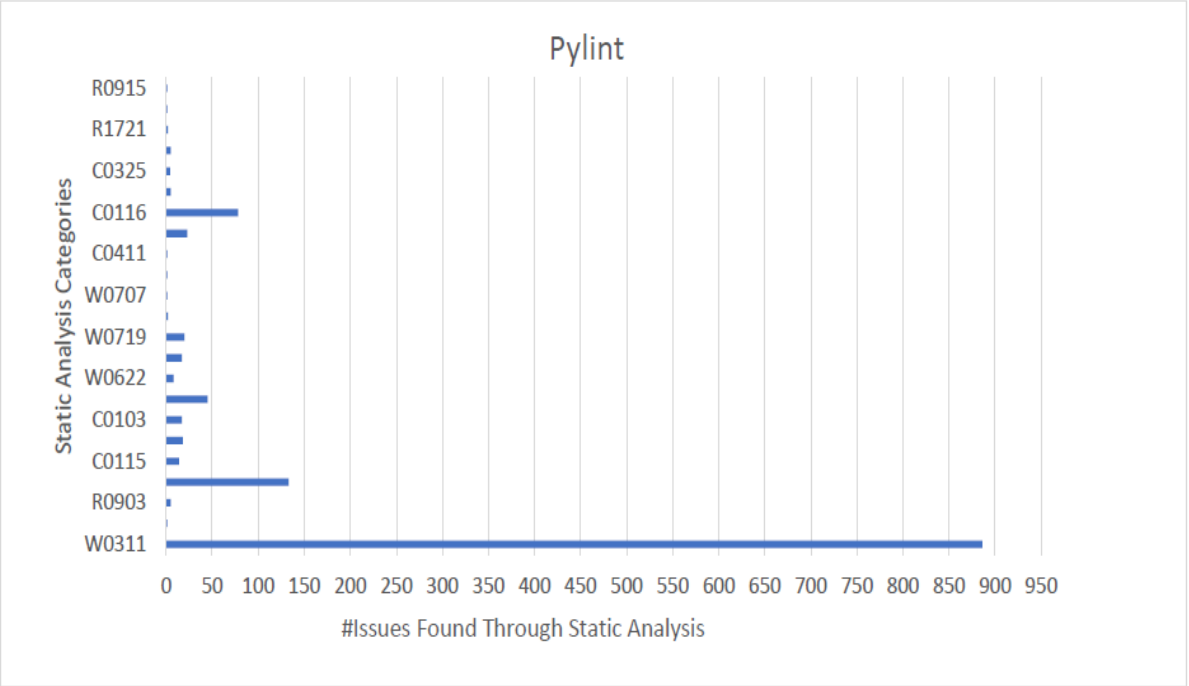
Pycodestyle graph:



Pycodestyle Graph Distribution Analysis:

Pycodestyle had a total of 18 repetitive categories found with a total of 906 issues spread across those categories, Pycodestyle is one of the tools that falls under Flake8, therefore results displayed from Pycodestyle contain part of the exact errors as Flake8. For more detailed information with regards to category, issues and error messages please refer to appendix 2 (Pydocstyle Category/Issues Table).

Pylint Graph:



Pylint Graph Distribution Analysis:

Pylint had a total of 23 unique categories found with a total of 1288 issues spread across those categories, for more detailed information with regards to category, issues and error messages please refer to appendix 5 (Pylint Category/Issues Table).

2.2 The Analysis of megamart.py

Megamart.py top 10 categories in most frequent occurrences in static analysis tools, no duplicates or similar errors		
Categories	#issues Found	Error message
W0311 (Pylint)	81	bad-indentation
E501 (Flake8)	76	Line too long (a>b characters)
E701 (Flake8)	17	Multiple statements in one line
Assignment (my py)	12	Incompatible types in assignment
D205	8	Docstring missing
D401	5	First line should be imperative mood
W293	4	Blank lines contains whitespace
F401	2	Module imported but unused
E266	2	Too many leading # for block comment
E203	1	Whitespace before ':'

Previous code & Explanation (E501):
Running flake8 megamart.py will give the following results to indicate that the line for the comment has exceeded what is considered acceptable, and will therefore hinder readability of the comments, this is a good practice because we want to have short comments that is somewhat centralised in one place, that can efficiently tell the developer on the logic of the code rather than scrolling through line by line.This error falls under the category of E501:
For more information: https://peps.python.org/pep-0008/#maximum-line-length

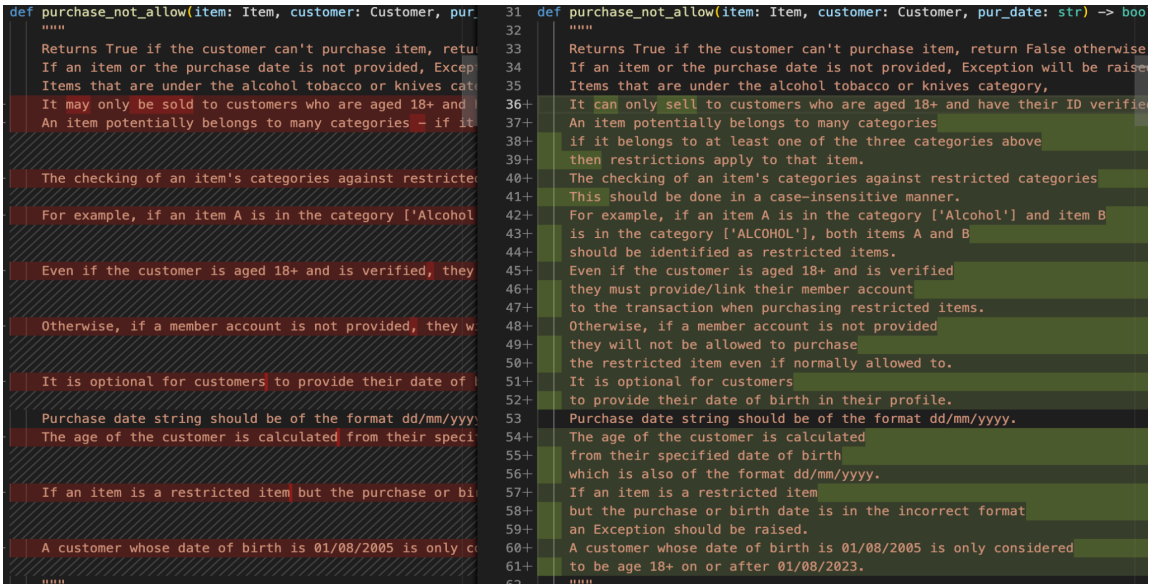
```
megamart.py:31:80: E501 line too long (94 > 79 characters)
megamart.py:32:80: E501 line too long (105 > 79 characters)
megamart.py:33:80: E501 line too long (137 > 79 characters)
megamart.py:34:80: E501 line too long (152 > 79 characters)
megamart.py:35:80: E501 line too long (113 > 79 characters)
megamart.py:36:80: E501 line too long (163 > 79 characters)
megamart.py:37:80: E501 line too long (148 > 79 characters)
megamart.py:38:80: E501 line too long (135 > 79 characters)
megamart.py:41:80: E501 line too long (115 > 79 characters)
megamart.py:42:80: E501 line too long (123 > 79 characters)
megamart.py:43:80: E501 line too long (103 > 79 characters)
```

```
"""
Returns True if the customer is not allowed to purchase the specified item, False otherwise.
If an item object or the purchase date string was not actually provided, an Exception should be raised.
Items that are under the alcohol, tobacco or knives category may only be sold to customers who are aged 18+ and have their ID verified.
An item potentially belongs to many categories - as long as it belongs to at least one of the three categories above, restrictions apply to that item.
The checking of an item's categories against restricted categories should be done in a case-insensitive manner.
 For example, if an item A is in the category ['Alcohol'] and item B is in the category ['ALCOHOL'], both items A and B should be identified as restricted items
Even if the customer is aged 18+ and is verified, they must provide/link their member account to the transaction when purchasing restricted items.
Otherwise, if a member account is not provided, they will not be allowed to purchase the restricted item even if normally allowed to.
It is optional for customers to provide their date of birth in their profile.
Purchase date string should be of the format dd/mm/yyyy.
The age of the customer is calculated from their specified date of birth, which is also of the format dd/mm/yyyy.
If an item is a restricted item but the purchase or birth date is in the incorrect format, an Exception should be raised.
A customer whose date of birth is 01/08/2005 is only considered to be age 18+ on or after 01/08/2023.
"""
```

The process of eliminating the error:

I want to simply reduce the amount of characters that are displayed per line. We can do that in a cheap way by cutting one line into multiple to reduce the length. The code difference shows in this case that by doing this, we are able to have better visibility of the comments when we do split screen

Results:



Previous code & Explanation (E701):

Running flake8 megamart.py will show the error that there are multiple statements in one line, this is because we have two sets of code fulfilling their purpose that are written in the same line. A most common example can be demonstrated in the if statement for the second line of the screenshot: if item is None: raise Exception(): this is really bad for readability and

it may confuse the developers due to it being messy. Each line should serve their own purpose, hence we want to write our raise Exceptions the line after the if statement.

For more information please refer to the official documentation: https://pypi.org/project/flake8/1.6.1/#:~:text=E701%3A%20multiple%20statements%20on%20one,W291%3A%20trailing%20whitespace

```
RESTRICTED_CATEGORIES = ["Alcohol", "Tobacco", "Knives"]
# Check that an item object and purchase date string are actually provided.
if item is None: raise Exception()

# Handle restricted items
if any(c1.lower() == c2.lower() for c1 in item.categories for c2 in RESTRICTED_CATEGORIES):
  # Defensive Check
  if customer is None or customer.date_of_birth is None or purchase_date_string is None: return True

  # Check for valid dates
  try:
    cust_datetime = _get_eighteenth_birthday(datetime.strptime(customer.date_of_birth, r"%d/%m/%Y"))
    purch_datetime = datetime.strptime(purchase_date_string, r"%d/%m/%Y")
  except ValueError:
    raise Exception()

  return bool(not customer.id_verified or (cust_datetime > purch_datetime))
# Unrestricted items
return False
```

The process of eliminating the error:

```
if item is None or items_dict is None:
    raise InsufficientStockException()
```

We will simply set our exception to the next line in order to improve our readability like this. It is a simple but important fix for the maintainability aspect of megamart

Results:

```
if item is None or items_dict is None:
    raise InsufficientStockException()

return items_dict[item.id][2] if item.id in items_dict else Non
```

```
74+ if item is None or items_dict is None: raise Exception()
75
76+ return items_dict[item.id][2] if item.id in items_dict else None
77
```

Previous code & Explanation (W0311):

Bad indentation is the most common occurring error in megamart.py, having bad indentation is normal where the developers are working with a huge codebase, and one that can be ignored by the system with bad indentation, although the system will work but it does not abide by the best industry standards. Therefore, we also want to prioritise that fix for our future development process of megamart.

```
if item.id in discounts_dict:
  discount = discounts_dict[item.id]

  if discount.type == DiscountType.PERCENTAGE:
      pct = discount.value
    if pct < 1.00 or pct > 100.00: raise Exception()

    return round(rounded_original_price - (rounded_original_price * (pct/100)), 2)
```

Process of eliminating errors:

```
  rounded_original_price = round(item.original_price, 2)

  if item.id in discounts_dict:
    discount = discounts_dict[item.id]

    if discount.type == DiscountType.PERCENTAGE:
      pct = discount.value
      if pct < 1.00 or pct > 100.00: raise Exception()

      return round(rounded_original_price - (rounded_original_price * (pct/100)), 2)
```

We are able to fix that by reading through each line and knowing when we are supposed to indent, this is shown by the screenshot specifically at the section where if subtotal…, we can see that the developer have missed correct indentations and it could be a return of indentation of 2, in which case we can simply add another 2 indentations to make it more readable, abide by good coding practice and standards and overall improve the quality of the code. Although the visibility aspect is not easy to see.

Results:

```
def round_off_subtotal(subtotal: float, payment_method: Pa
  """
  Currently, subtotal rounding is only applicable when pay
  There is no rounding performed in any other case.
  If the subtotal value or payment method was not actually

  The subtotal is rounded off to the nearest multiple of 5

  Cent amounts which have their ones-place digit as 1 - 2

  As the (monetary) subtotal value is provided as a float,
  """
  # Check float and payment method is not None
  if subtotal is None or payment_method is None: raise Exc

  rounded_subtotal = round(subtotal, 2)

  if payment_method is PaymentMethod.CASH:
    return round((5 * round(int(rounded_subtotal*100)/5))/

  return rounded_subtotal
```

```
262+ def round_off_subtotal(subtotal: float, payment_mode: PaymentMethod) -> flo
263+     """
264   Currently, subtotal rounding is only applicable when paying by cash.
265   There is no rounding performed in any other case.
266+  If the subtotal value or payment method was
267+  not actually provided, an Exception should be raised.
268+  The subtotal is rounded off to the nearest multiple of 5 cents.
269+  Surcharge value returned should have at most two decimal places.
270+  Cent amounts which have their ones-place digit as 1 - 2 or 6 - 7
271+  will be rounded down. If it is 3 - 4 or 8 - 9, it will be rounded up inst
272+  As the (monetary) subtotal value is provided as a float, ensure that
273+  it is first rounded off to two decimal places before doing the rounding.
274+     """
275+     # Check float and payment method is not None
276+     if subtotal is None or payment_mode is None:
277+         raise Exception()
278
279+     rounded_subtotal = round(subtotal, 2)
280
281+     if payment_mode is PaymentMethod.CASH:
282+         return round((5 * round(int(rounded_subtotal*100)/5))/100, 2)
283
284+     return rounded_subtotal
```

Previous code & Explanation (Assignment):

Assignment happens when a data type that is not compatible with what the function is required, in this case the date time requires a string. The passed in value is a None therefore it will give us an assignment error. Our previous code only gives the option of None although it should be able to assign to a string.

```
class Transaction:
  date: str = None # format: dd/mm/YYYY e.g. 01/08/2023
  time: str = None # format: HH:MM:SS e.g. 12:45:00
  transaction_lines: List[TransactionLine] = []
  customer: Optional[Customer] = None
  fulfilment_type: Optional[FulfilmentType] = None
  payment_method: Optional[PaymentMethod] = None
  amount_tendered: Optional[float] = None

  total_items_purchased: Optional[int] = None
  all_items_subtotal: Optional[float] = None
  fulfilment_surcharge_amount: Optional[float] = None
  rounding_amount_applied: Optional[float] = None
  final_total: Optional[float] = None
  change_amount: Optional[float] = None
  amount_saved: Optional[float] = None
```

Process of eliminating errors:

```
Transaction.py:9: error: Incompatible types in assignment (expression has type "None", variable has type "str")  [assignment]
Transaction.py:10: error: Incompatible types in assignment (expression has type "None", variable has type "str")  [assignment]
```

It is quite simple, we just want to be able to have two options that can lead to two outcomes, one of them can either contain a string or it contains a None, because we can clearly see in the format that we are passing in customer birthday, so we can make the assumption that the customer either haven't disclosed their birthday because it is a megamart, you don't have to have an account to shop there or they have, in which case the string declaration (optional) comes in handy.

Results:

```
class Transaction:
  date: Optional[str] = None # format: dd/mm/YYYY e.g. 01/08/2023
  time: Optional[str] = None # format: HH:MM:SS e.g. 12:45:00
  transaction_lines: List[TransactionLine] = []
  customer: Optional[Customer] = None
  fulfilment_type: Optional[FulfilmentType] = None
  payment_method: Optional[PaymentMethod] = None
  amount_tendered: Optional[float] = None

  total_items_purchased: Optional[int] = None
  all_items_subtotal: Optional[float] = None
```

Previous code & Explanation (D205):

Docstring is a form of comment that is given as a one liner that ends with a period, this aims to simply improve commenting for the individual developers, which can prove to be enormously useful because developers have the habit of not commenting on their code. By introducing docstring, developers can declare one line comments that ends with a period that tells for short exactly what the program does.

```python
from typing import List, Optional
from TransactionLine import TransactionLine
from Customer import Customer
from FulfilmentType import FulfilmentType
from PaymentMethod import PaymentMethod


class Transaction:
  date: str = None # format: dd/mm/YYYY e.g. 01/08/2023
  time: str = None # format: HH:MM:SS e.g. 12:45:00
  transaction_lines: List[TransactionLine] = []
  customer: Optional[Customer] = None
  fulfilment_type: Optional[FulfilmentType] = None
  payment_method: Optional[PaymentMethod] = None
  amount_tendered: Optional[float] = None

  total_items_purchased: Optional[int] = None
  all_items_subtotal: Optional[float] = None
  fulfilment_surcharge_amount: Optional[float] = None
  rounding_amount_applied: Optional[float] = None
  final_total: Optional[float] = None
  change_amount: Optional[float] = None
  amount_saved: Optional[float] = None
```

Process of eliminating errors:

```
megamart.py:1 at module level:
        D100: Missing docstring in public module
megamart.py:30 in public function `is_not_allowed_to_purchase_item`:
```

As seen above, there are no docstrings to tell us what the import statement does, the developer might not be familiar with python and don't understand import statements, by introducing docstring we simply make a comment of a one liner code that tells us what it does.

Results:

```python
"""import libraries."""
from datetime import datetime
from typing import Dict, Tuple, Optional
from dateutil.relativedelta import relativedelta
from DiscountType import DiscountType
from PaymentMethod import PaymentMethod
from FulfilmentType import FulfilmentType
```

```python
1  """import libraries."""
2  from datetime import datetime
3  from typing import Dict, Tuple, Optional
4  from dateutil.relativedelta import relativedelt
5  from DiscountType import DiscountType
6  from PaymentMethod import PaymentMethod
7  from FulfilmentType import FulfilmentType
```

Not the best practice as it is not informative but it fixes the issue.

Previous code & Explanation (D401):

This error tells the developer to have specific keywords to be exact on the instructions that are given, in this case it is a Return statement.

Process of eliminating errors:

This is pretty simple and evident because it tells us what kind of statements that it wants, in this case a Return for every starting statement

```
        D401: First line should be in imperative mood (perhaps 'Return', not 'Returns')
megamart.py:66 in public function `get_item_purchase_quantity_limit`:
```

Results:

Previous code & Explanation (W293):

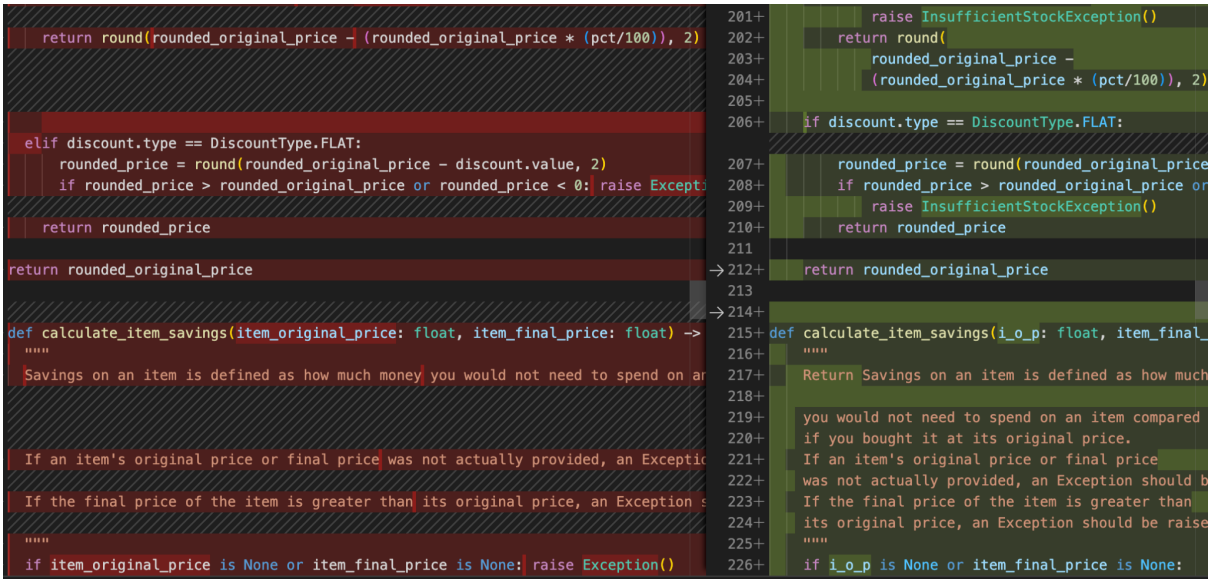
The error code is again focused on improving the readability and the maintainability aspect of the code, although this error is quite self-explanatory.

Process of eliminating errors:

This simply requires filling in the blank line that contains a white space.

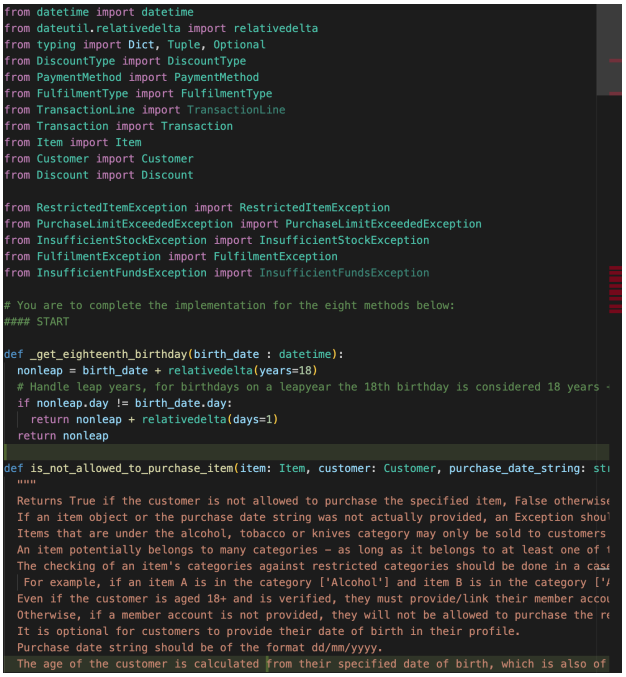
Results:

Certain specific lines with whitespace can just do delete until the start of the blank line or simply remove it all together.



Previous code & Explanation (F401):

This error indicates the unused import statements that are above the import block, the import statements that are not used are considered redundant code, and need to be removed.



```
from datetime import datetime
from dateutil.relativedelta import relativedelta
from typing import Dict, Tuple, Optional
from DiscountType import DiscountType
from PaymentMethod import PaymentMethod
from FulfilmentType import FulfilmentType
from TransactionLine import TransactionLine
from Transaction import Transaction
from Item import Item
from Customer import Customer
from Discount import Discount

from RestrictedItemException import RestrictedItemException
from PurchaseLimitExceededException import PurchaseLimitExceededException
from InsufficientStockException import InsufficientStockException
from FulfilmentException import FulfilmentException
from InsufficientFundsException import InsufficientFundsException

# You are to complete the implementation for the eight methods below:
#### START

def _get_eighteenth_birthday(birth_date : datetime):
  nonleap = birth_date + relativedelta(years=18)
  # Handle leap years, for birthdays on a leapyear the 18th birthday is considered 18 years
  if nonleap.day != birth_date.day:
    return nonleap + relativedelta(days=1)
  return nonleap

def is_not_allowed_to_purchase_item(item: Item, customer: Customer, purchase_date_string: st
  """
  Returns True if the customer is not allowed to purchase the specified item, False otherwis
  If an item object or the purchase date string was not actually provided, an Exception shou
  Items that are under the alcohol, tobacco or knives category may only be sold to customers
  An item potentially belongs to many categories - as long as it belongs to at least one of
  The checking of an item's categories against restricted categories should be done in a cas
  For example, if an item A is in the category ['Alcohol'] and item B is in the category ['
  Even if the customer is aged 18+ and is verified, they must provide/link their member accou
  Otherwise, if a member account is not provided, they will not be allowed to purchase the r
  It is optional for customers to provide their date of birth in their profile.
  Purchase date string should be of the format dd/mm/yyyy.
  The age of the customer is calculated from their specified date of birth, which is also of
```

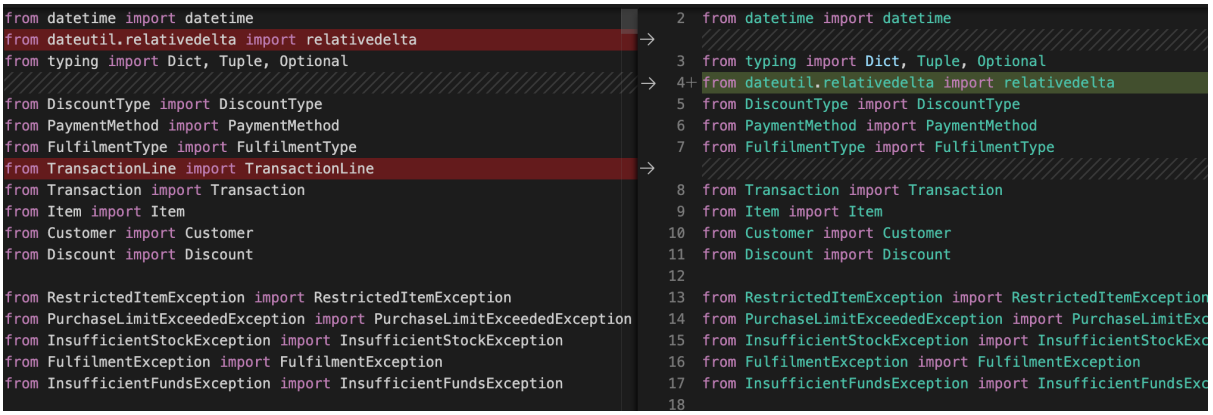
The dark green indicates the ones that are not used whereas light green are the ones that are being used.

Process of eliminating errors:

The error describes certain specific lines as imported but unused, which means that there is no interaction with the system. In the beginning, we are not using transactionLine or insufficientFundsException, therefore these import statements don't serve much purpose rather than being in the import statement block. Again, this is quality assurance and we want to make sure our codebase maintains all the libraries that is used and get rid of the ones that are not used.

```
megamart.py:7:1: F401 'TransactionLine.TransactionLine' imported but unused
megamart.py:17:1: F401 'InsufficientFundsException.InsufficientFundsException' imported but unused
```

Results:



```
from datetime import datetime
from dateutil.relativedelta import relativedelta
from typing import Dict, Tuple, Optional

from DiscountType import DiscountType
from PaymentMethod import PaymentMethod
from FulfilmentType import FulfilmentType
from TransactionLine import TransactionLine
from Transaction import Transaction
from Item import Item
from Customer import Customer
from Discount import Discount

from RestrictedItemException import RestrictedItemException
from PurchaseLimitExceededException import PurchaseLimitExceededException
from InsufficientStockException import InsufficientStockException
from FulfilmentException import FulfilmentException
from InsufficientFundsException import InsufficientFundsException
```

```
from datetime import datetime

from typing import Dict, Tuple, Optional
from dateutil.relativedelta import relativedelta
from DiscountType import DiscountType
from PaymentMethod import PaymentMethod
from FulfilmentType import FulfilmentType

from Transaction import Transaction
from Item import Item
from Customer import Customer
from Discount import Discount

from RestrictedItemException import RestrictedItemException
from PurchaseLimitExceededException import PurchaseLimitExc
from InsufficientStockException import InsufficientStockExc
from FulfilmentException import FulfilmentException
from InsufficientFundsException import InsufficientFundsExc
```

We will remove the transactionLine because it does not serve much purpose after we have fixed up our codebase, and we will keep the insufficientFundsException as we later on use it for our edge case.

Previous code & Explanation (E266):

Too many leading blocks of comments for “#’ is to show that we only need one to comment, in the beginning we had the “#### START” which means that we are using multiple comment symbols when it is unnecessary and make the things look messy.

```
from typing import Dict, Tuple, Optional
from DiscountType import DiscountType
from PaymentMethod import PaymentMethod
from FulfilmentType import FulfilmentType
from TransactionLine import TransactionLine
from Transaction import Transaction
from Item import Item
from Customer import Customer
from Discount import Discount

from RestrictedItemException import RestrictedItemException
from PurchaseLimitExceededException import PurchaseLimitExceededExceptio
from InsufficientStockException import InsufficientStockException
from FulfilmentException import FulfilmentException
from InsufficientFundsException import InsufficientFundsException

# You are to complete the implementation for the eight methods below:
#### START
```

As shown above, we only needed one so “# START” would have been efficient and cleaner to look at

Process of eliminating errors:

The only requirement was to remove the additional “#” from the start to resolve the error as shown here:

```
megamart.py:226:1: E266 too many leading '#' for block comment
```

Results:

Previous code & Explanation (E203):

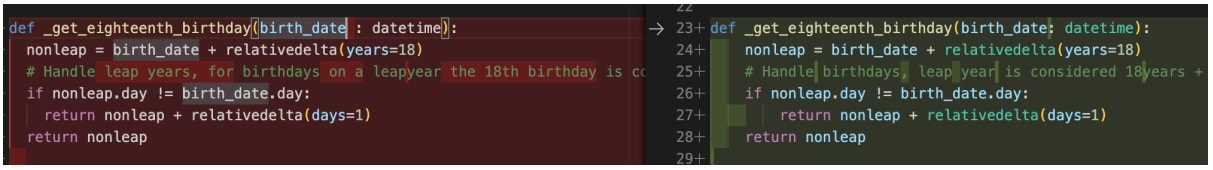
White space : before our actual argument is a quick fix, it is seen here where we have our birth_date being separated by a space on the colon. This is a bad practice for code readability again.

```
def _get_eighteenth_birthday(birth_date : datetime):
  nonleap = birth_date + relativedelta(years=18)
  # Handle leap years, for birthdays on a leapyear the 18th b
  if nonleap.day != birth_date.day:
    return nonleap + relativedelta(days=1)
  return nonleap
```

Process of eliminating errors:

Removing the white space proves to be sufficient in resolving the error, and no further action is required.

Results:



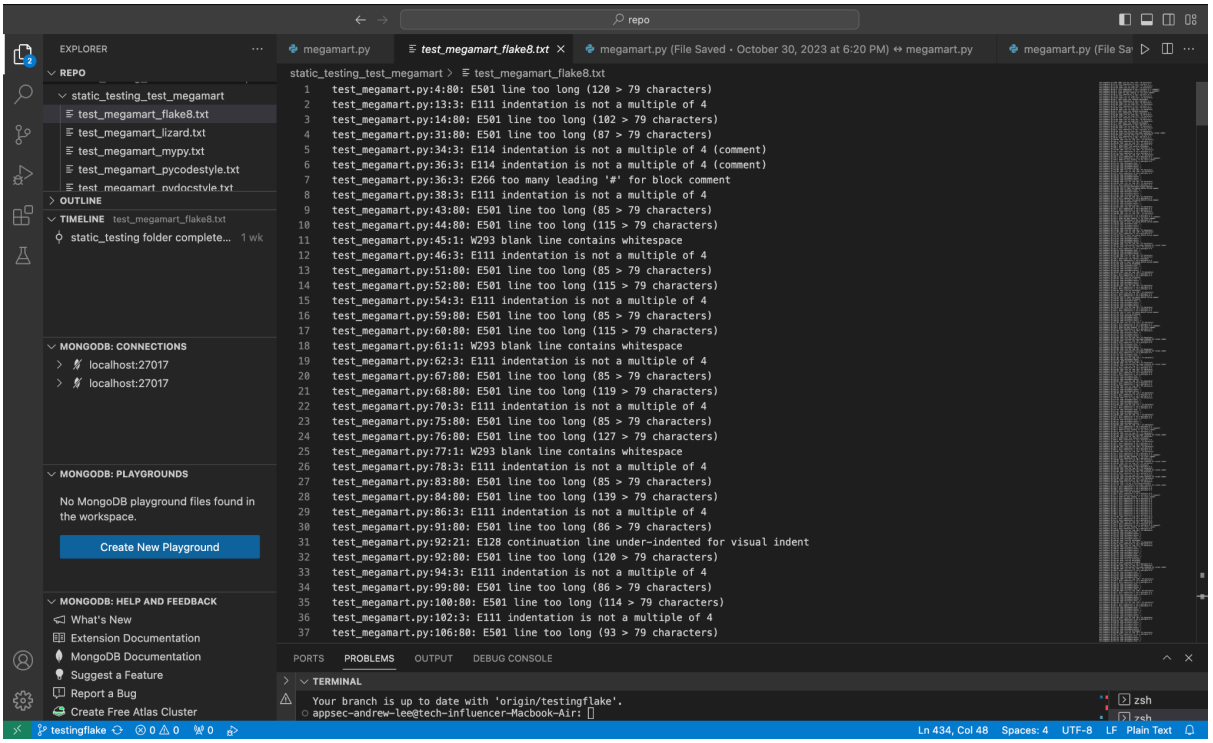
2.3 Comparison Table

The information is given in the appendix table for megamart.py and the appendix for total overall scans respectively.

Aspects/Tools	pycodestyles-	pydocstyles-	pylint	flake8	mypy-	lizard
Total Issues found across all files in the entire repository	906	59	1288	906	51	-
Score (if any)	N/A	N/A	10.00/10	N/A	N/A	-
# Issues found for the megamart.py	171	17	177	179	14	-
# Fixed Issues for the megamart.py	171	17	177	179	14	-
# Remaining Issues Unfixed	0	0	0	0	0	-

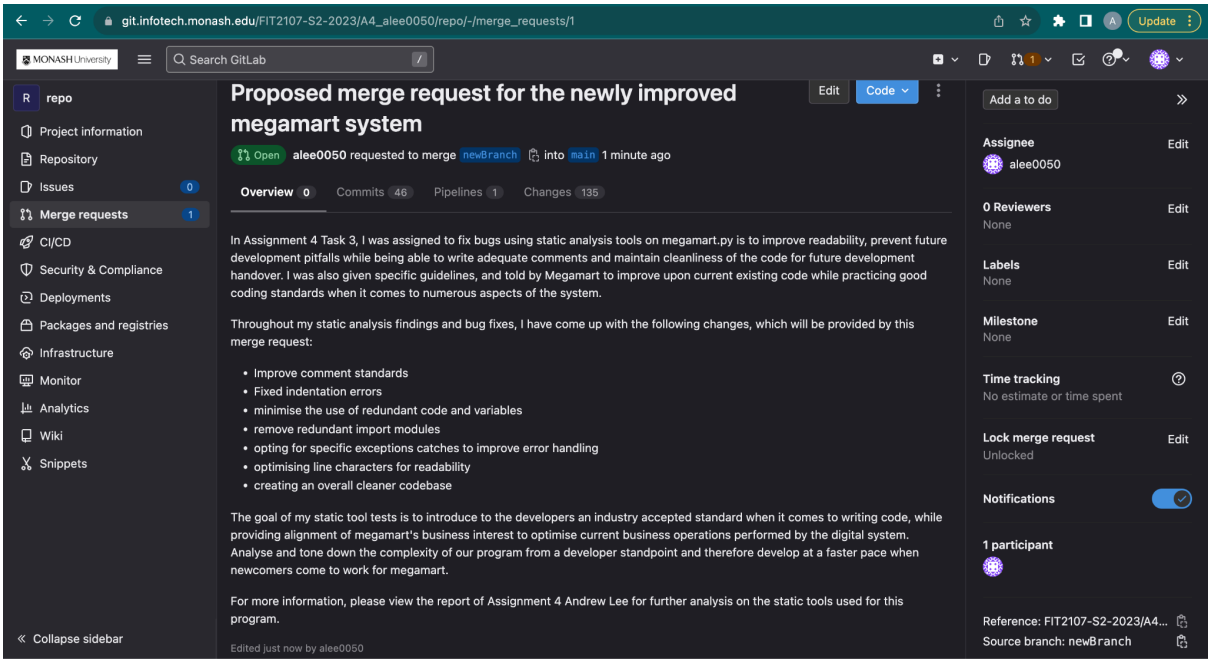
2.4 Conclusion

a) Based on the findings that we have shown in the report, we can state most of the codebase errors revolving code readability stems from the test_megamart where it contains all of the test cases for megamart. This is shown as we can run the command flake8 test_megamart.py > flake8_test_megamart.txt to write all the errors that is found in test_megamart using the static analysis tool in a text file, and view them accordingly (for more information, please view gitlab as I have pushed the errors in the respective txt files).



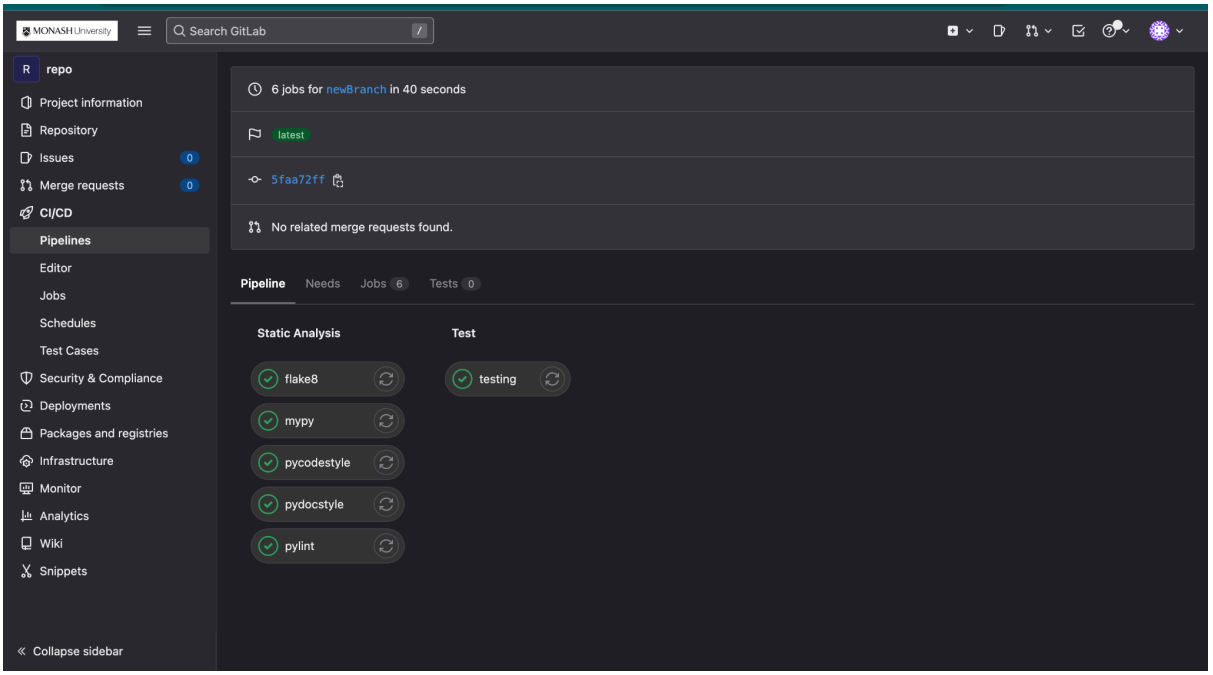
3.1: Merge Request

url:https://git.infotech.monash.edu/FIT2107-S2-2023/A4_alee0050/repo/-/merge_requests/1



3.2: CI/CD of the Improved Version of Megamart.py

url: https://git.infotech.monash.edu/FIT2107-S2-2023/A4_alee0050/repo/-/pipelines/76858


MONASH University
Search GitLab
/
R
repo
Project information
Repository
Issues
0
Merge requests
0
CI/CD
Pipelines
Editor
Jobs
Schedules
Test Cases
Security & Compliance
Deployments
Packages and registries
Infrastructure
Monitor
Analytics
Wiki
Snippets
Collapse sidebar
6 jobs for newBranch in 40 seconds
latest
5faa72ff
No related merge requests found.
Pipeline
Needs
Jobs 6
Tests 0
Static Analysis
flake8
mypy
pycodestyle
pydocstyle
pylint
Test
testing

APPENDIX:

Flake8 Category/Issues Table	
Categories	#Issues found
E111	351
E501	245
E261	9
W293	23
E302	10
W291	22
E202	49
E201	56
E266	11
E701	17
E114	24
F401	2
E203	65
E722	5
E275	3
E128	13
F841	2
F541	1
E305	1
E303	1
E225	1

Pylint Category/Issues Table	
Categories	#Issues found
W0311	886
R0913	1
R0903	5
C0301	133
C0115	14
C0114	18
C0103	17
C0303	45
W0622	8
C0321	17
W0719	20
W0611	2
W0707	1
R1705	1
C0411	1
C0209	23
C0116	78
W0702	5
C0325	4
W0612	5
R1721	2
W0718	1
R0915	1

Pydocstyle Category/Issue Table	
Categories	#Issues Found
D100	16
D101	12
D103	10
D205	8
D401	5
D107	5
D403	1
D400	1
D300	1

Pycodestyle Category/Issue Table	
Categories	#Issues Found
E111	351
E501	245
E203	65
E201	56
E202	49
E114	24
W293	23
W291	22
E701	17
E128	13

E266	11
E302	10
E261	9
E722	5
E275	3
E305	1
E303	1
E225	1

Mypy Category/Issue Table	
Category	#Issues Found
Arg-type	10
Assignment	11
Operator	12
Import-untyped	1
Return-value	2
Union-attr	5
Str-format	3
Attr-defined	1
Note	6

Megamart.py Analysis:

Flake8:

Categories (Y)	#Issues Found(X)	Error message

E111	55	Indentation is not a multiple of four
E114	11	Indentation is not a multiple of four (comment)
E203	1	Whitespaces before ‘:’.
E266	2	Too many leading ‘#’ for block comment
E302	9	Expected 2 blank lines, found x.
E501	76	Line too long (a>b characters)
E701	17	Multiple statements on one line (colon)
F401	2	Module imported but unused
W291	2	Trailing whitespace
W293	4	Blank line containing whitespace

Pylint:
Error guide:
https://pylint.readthedocs.io/en/latest/user_guide/messages/messages_overview.html

Categories (Y)	#Issues Found(X)	Error Message
C0103	1	invalid-name
C0114	1	missing-module-docstring
C0301	53	line-too-long
C0303	6	trailing-whitespace
C0321	17	multiple-statements
C0411	1	wrong-import-order
R1705	1	no-else-return
W0311	81	bad-indentation
W0611	2	unused-import
W0707	1	raise-missing-from

W0719	13	broad-exception-raised

Mypy:
Error guide: https://mypy.readthedocs.io/en/stable/error_codes.html

Category (Y)	#Issues Found(X)
Incompatible types [assignment]	5
Incompatible args-type [arg-type]	3
Library stubs [import-untyped]	1
Unsupported operand types [operator]	1
Note [instructions]	4

Pydocstyle Error guide:https://www.pydocstyle.org/en/stable/error_codes.html

Category (Y)	#Issues Found(X)	Error Message
D100	1	Missing docstring in module
D205	8	1 blank line required between summary line and description
D300	1	Use """triple double quotes"""
D400	1	First line should end with a period
D401	5	First line should be in imperative mood
D403	1	First word of the first line should be properly capitalised