CONTENTS

i. PROJECT OVERVIEW

Credit scoring models are statistical analysis used by banks that evaluate the worthiness to receive credit. The agencies select statistical characteristics found in a person's credit payment patterns, analyze them and come up with a credit score.

Scoring calculations are based on ***payment record, frequency of payments, amount of debts, credit charge-offs, and number of credit cards*** held. A weight is assigned to each factor considered in the model's formula and a credit score is assigned based on the evaluation.

Lenders use credit scores to determine the risk involved in making a loan, the terms of the loan, and the interest rate. The higher the score, the better the terms of a loan will be offered.

Credit scoring is perhaps ***one of the most classic applications for predictive modeling*** to predict whether or not the credit extended to an applicant will likely result in profit or loss for the lending institution. Credit scoring is the set of decision models and their underlying techniques that aid lenders in the granting of consumer credit. These techniques determine who will get credit, how much credit they should get, and what operational strategies will enhance the profitability of the borrowers to the lenders. Further, they help to assess the risk in lending. Credit scoring is a dependable assessment of a person's credit worthiness since it is based on actual data.

- The target variable to predict credit score for current credit card customer is ***"Bad_label"*.**

  If Bad_label = 0, the customer has good credit history.

  Bad_label = 1, the customer has bad credit history.

- So, it is a ***binary classification*** that is used to predict the probability of a categorical dependent variable. The dependent variable is a binary variable that contains data coded as 1 (no) or 0 (yes).

ii. REQUIREMENTS

a. Dataset of credit history of customer
b. Python 3
c. Python libraries
   - Numpy
   - Pandas
   - Sklearn
   - Matplotlib
   - Seaborn
d. SonarPython
e. Pylint
f. Pybuilder
g. Travis CI

iii. UML (Unified Modeling Language)

UML is a modern approach to modeling and documenting software. It is based on ***diagrammatic representations*** of software components.

Sequence Diagram of the "credit score modelling" system is shown in Fig 1.

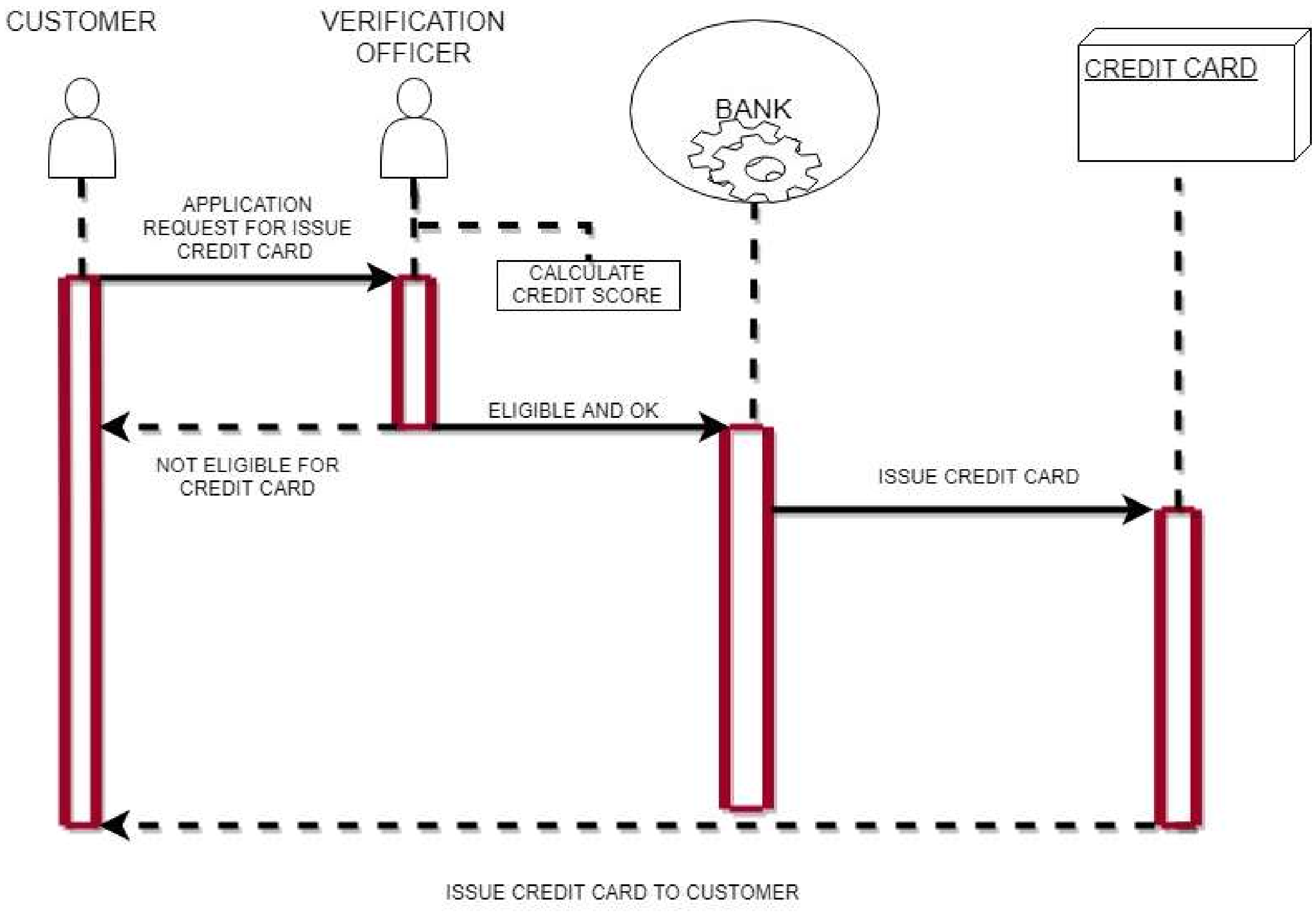

CUSTOMER
VERIFICATION OFFICER
BANK
CREDIT CARD
APPLICATION REQUEST FOR ISSUE CREDIT CARD
CALCULATE CREDIT SCORE
ELIGIBLE AND OK
NOT ELIGIBLE FOR CREDIT CARD
ISSUE CREDIT CARD
ISSUE CREDIT CARD TO CUSTOMER

Fig 1: Sequence Diagram of credit score modelling

Class diagram of the system is shown in Fig 2.

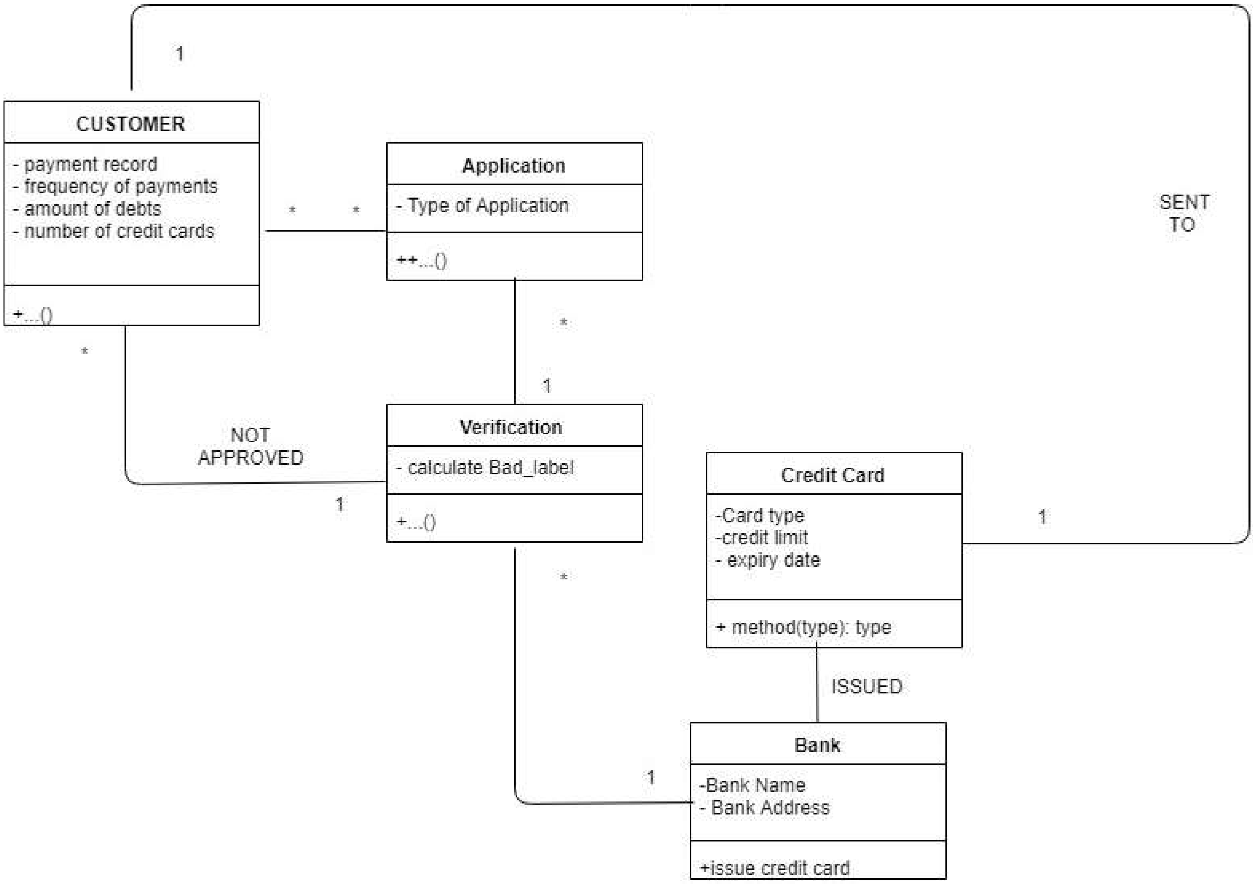


Fig 2: Class Diagram of credit score modelling

Use case diagram of the system is shown in Fig 3.

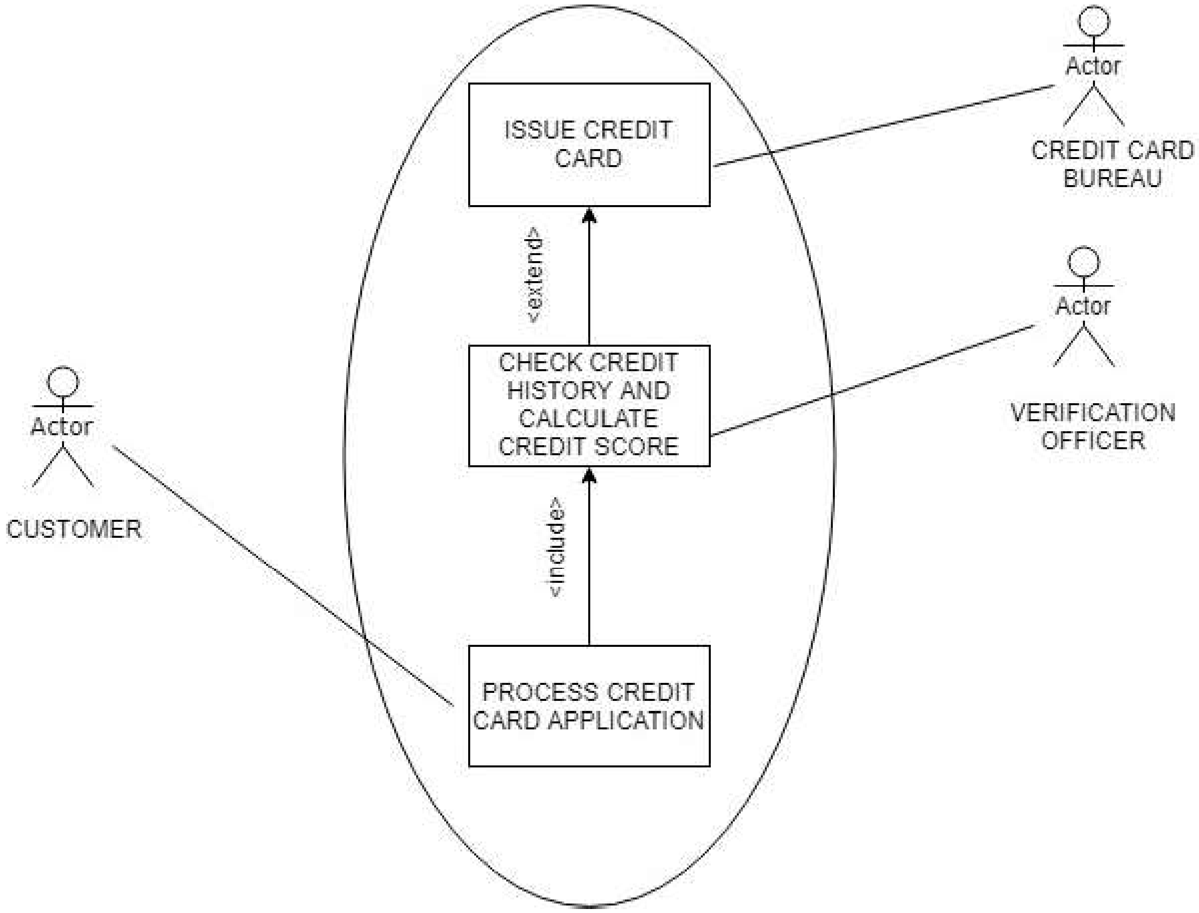


Fig 3: Use Case Diagram of credit score modelling

iv. Metrics

A software metric is a measure that says something about the software. There are many types of metrics related to software development, for example:

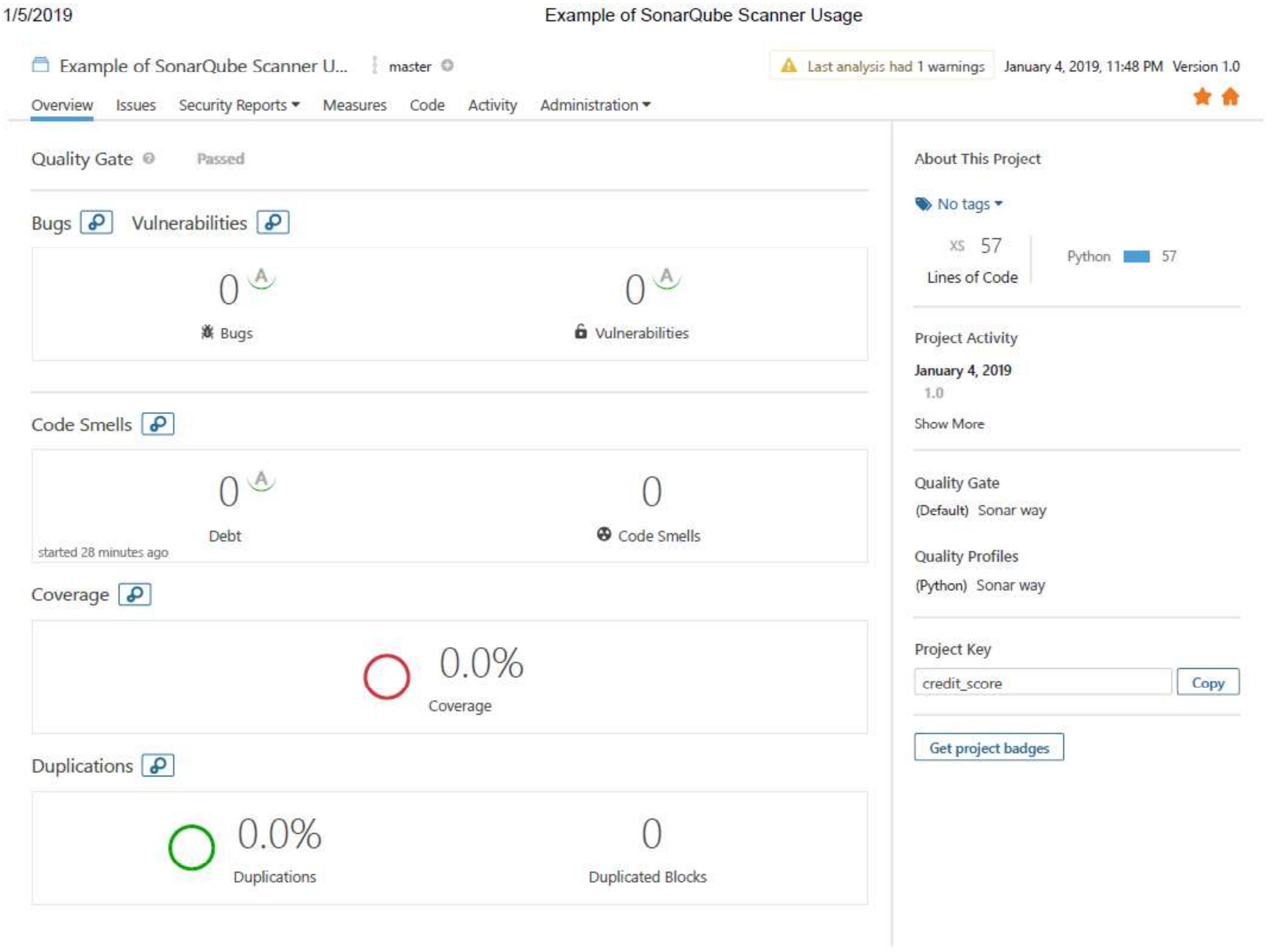
- sourcecode
- binary code
- external Libs
- buildinformation
- versioning information
- forms of meta- and additional Information (e.g. Annotations)

It is helpful to have an overview of very simple code metrics. These include the following metrics:

- lines of code
- comments
- methods/functions
- classes
- packages
- files
- duplication

To measure these simple code metrices, we can use SonarQube. SonarQube is static code analyzer to detect bugs, code smells, and security vulnerabilities on different programming languages. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities.

Analysis of the python code written for credit score modelling is done by ***SonarPython*** scanner which is available at - https://docs.sonarqube.org/display/PLUG/SonarPython. The report is shown in Fig 4 &5.



1/5/2019 Example of SonarQube Scanner Usage

Example of SonarQube Scanner U... master

⚠ Last analysis had 1 warnings January 4, 2019, 11:48 PM Version 1.0

Quality Gate: Passed

Bugs, Vulnerabilities

0 A Bugs

0 A Vulnerabilities

Code Smells

0 A Debt

started 28 minutes ago

0 Code Smells

Coverage

0.0% Coverage

Duplications

0.0% Duplications

0 Duplicated Blocks

About This Project

No tags ▾

XS 57 Lines of Code

Python 57

Project Activity

January 4, 2019

1.0

Show More

Quality Gate

(Default) Sonar way

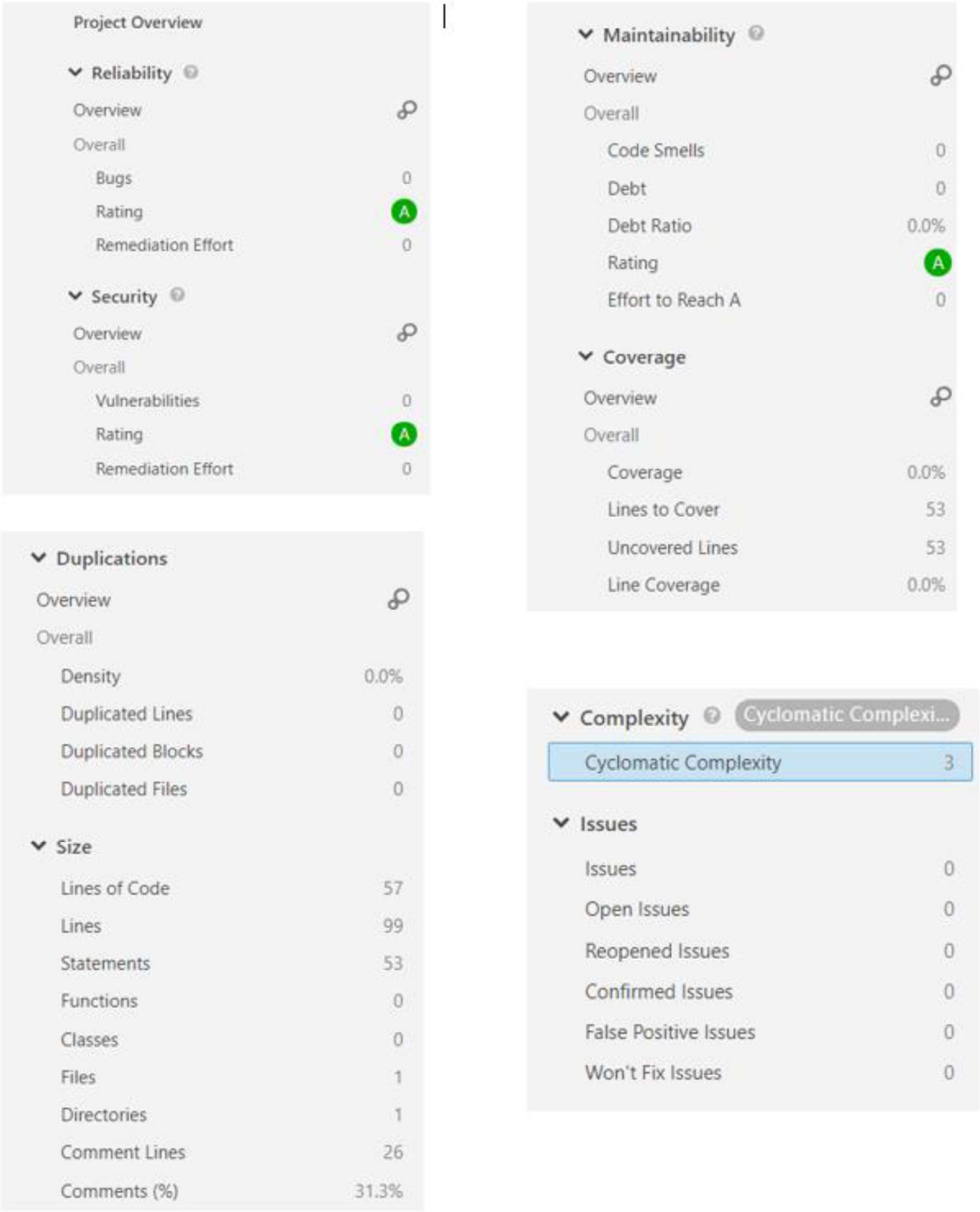
Quality Profiles

(Python) Sonar way

Project Key

credit_score

Fig 4: Analysis of basic metrics by SonarPython



Project Overview

Reliability

Overview

Overall

Bugs	0
Rating	A
Remediation Effort	0

Security

Overview

Overall

Vulnerabilities	0
Rating	A
Remediation Effort	0

Duplications

Overview

Overall

Density	0.0%
Duplicated Lines	0
Duplicated Blocks	0
Duplicated Files	0

Size

Lines of Code	57
Lines	99
Statements	53
Functions	0
Classes	0
Files	1
Directories	1
Comment Lines	26
Comments (%)	31.3%

Maintainability

Overview

Overall

Code Smells	0
Debt	0
Debt Ratio	0.0%
Rating	A
Effort to Reach A	0

Coverage

Overview

Overall

Coverage	0.0%
Lines to Cover	53
Uncovered Lines	53
Line Coverage	0.0%

Complexity

Cyclomatic Complexity	3

Issues

Issues	0
Open Issues	0
Reopened Issues	0
Confirmed Issues	0
False Positive Issues	0
Won't Fix Issues	0

Fig 5: Analysis of basic metrics by SonarPython

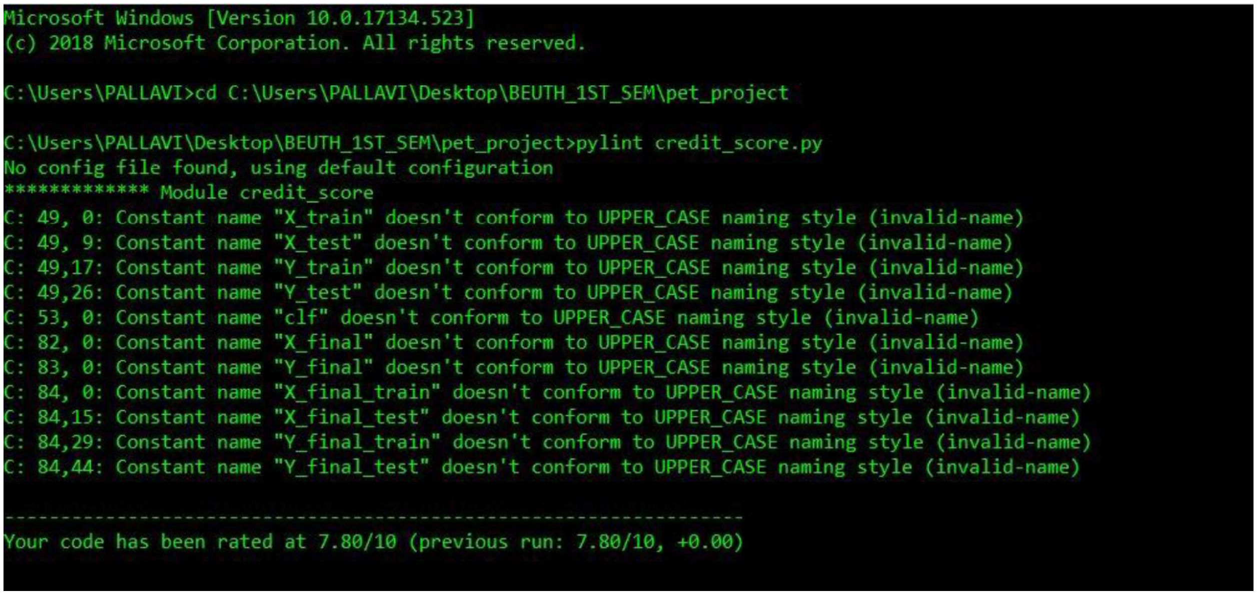
V. Clean Code Development

Clean code is code that is ***easy to understand*** (i.e. easy to understand the execution flow of the entire application, easy to understand how the different objects collaborate with each other, easy to understand the role and responsibility of each class, easy to understand what each method does, easy to understand what is the purpose of each expression and variable) and ***easy to change*** (i.e. code is easy to extend and refactor, and it's easy to fix bugs in the codebase.

Pylint has been used here for clean code development. Pylint is a source-code, bug and quality checker for the Python programming language. It is similar to Pychecker and Pyflakes, but includes the following features:

- Checking the length of each line
- Checking that variable names are well-formed according to the project's coding standard
- Checking that declared interfaces are truly implemented.

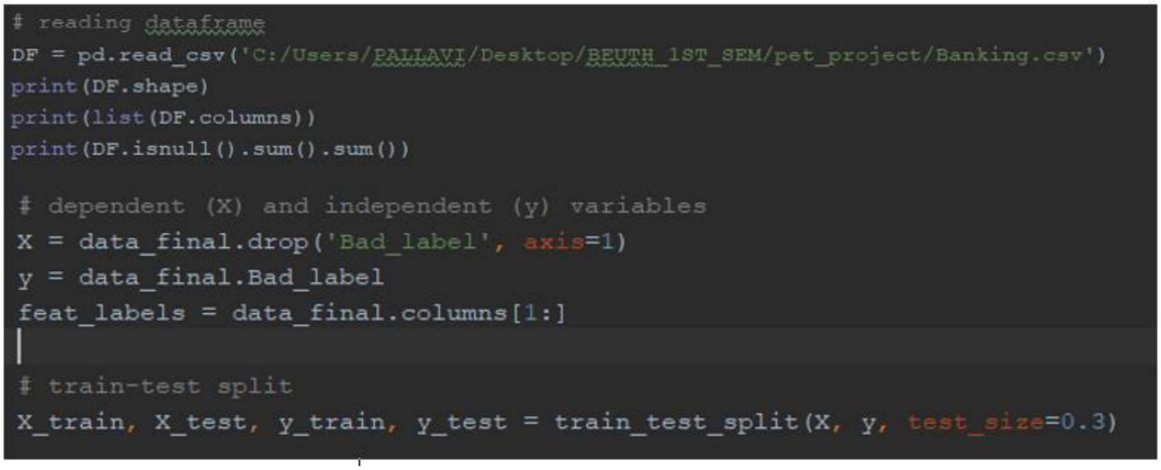
The pylint report of the above-mentioned codebase is attached below (Fig 6).



```
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\PALLAVI>cd C:\Users\PALLAVI\Desktop\BEUTH_1ST_SEM\pet_project

C:\Users\PALLAVI\Desktop\BEUTH_1ST_SEM\pet_project>pylint credit_score.py
No config file found, using default configuration
************* Module credit_score
C: 49, 0: Constant name "X_train" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 49, 9: Constant name "X_test" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 49,17: Constant name "Y_train" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 49,26: Constant name "Y_test" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 53, 0: Constant name "clf" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 82, 0: Constant name "X_final" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 83, 0: Constant name "Y_final" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 84, 0: Constant name "X_final_train" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 84,15: Constant name "X_final_test" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 84,29: Constant name "Y_final_train" doesn't conform to UPPER_CASE naming style (invalid-name)
C: 84,44: Constant name "Y_final_test" doesn't conform to UPPER_CASE naming style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 7.80/10 (previous run: 7.80/10, +0.00)
```

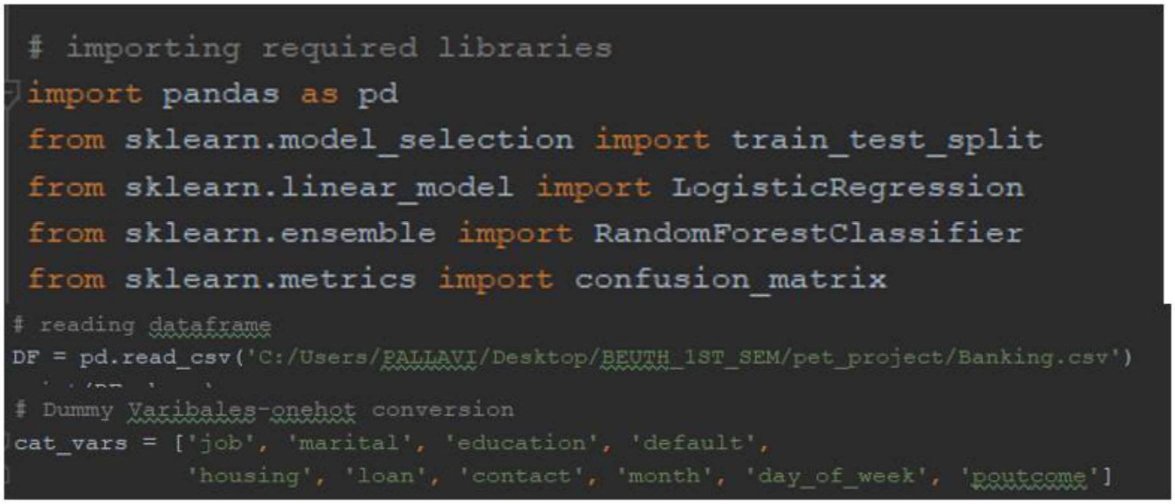
Fig 6: Pylint Report

i. Meaningful Names



```
# reading dataframe
DF = pd.read_csv('C:/Users/PALLAVI/Desktop/BEUTH_1ST_SEM/pet_project/Banking.csv')
print(DF.shape)
print(list(DF.columns))
print(DF.isnull().sum().sum())

# dependent (X) and independent (y) variables
X = data_final.drop('Bad_label', axis=1)
y = data_final.Bad_label
feat_labels = data_final.columns[1:]

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

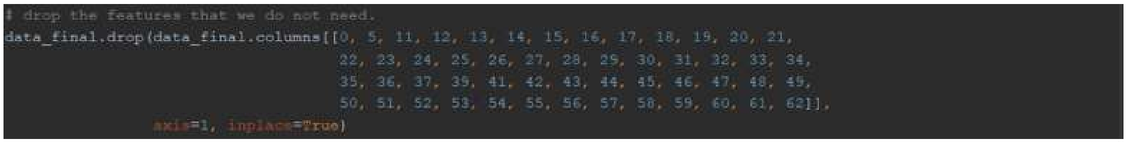
ii. Comments



```
# importing required libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
# reading dataframe
DF = pd.read_csv('C:/Users/PALLAVI/Desktop/BEUTH_1ST_SEM/pet_project/Banking.csv')
# Dummy Varibales-onehot conversion
cat_vars = ['job', 'marital', 'education', 'default',
            'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome']
```

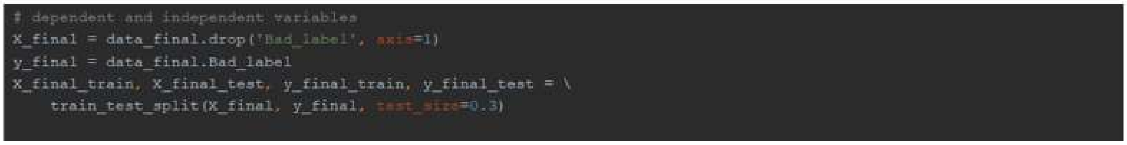
iii. Formatting

Setting a limit of characters (120) per line of code.



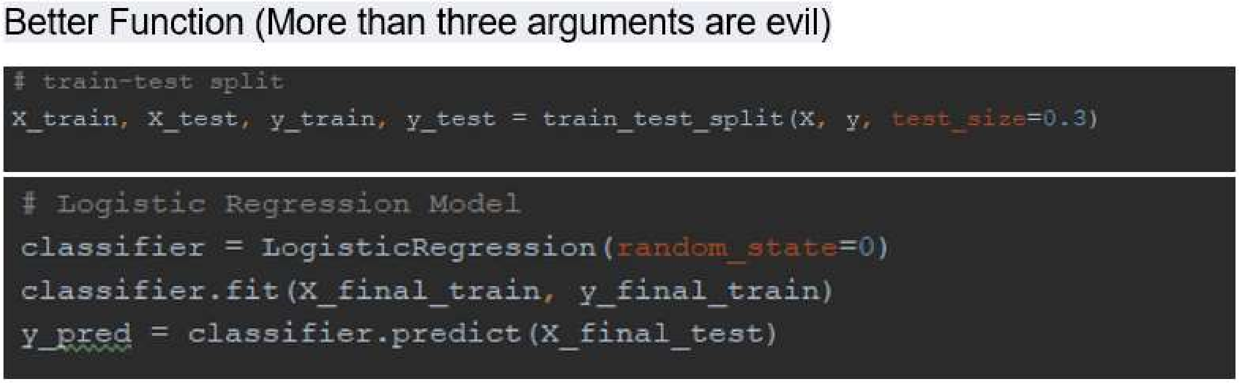
```
# drop the features that we do not need.
data_final.drop(data_final.columns[[0, 5, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
                                    22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
                                    35, 36, 37, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49,
                                    50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62]],
                axis=1, inplace=True)
```

Use spaces between operators, parameters, and commas.



```
# dependent and independent variables
X_final = data_final.drop('Bad_label', axis=1)
y_final = data_final.Bad_label
X_final_train, X_final_test, y_final_train, y_final_test = \
    train_test_split(X_final, y_final, test_size=0.3)
```

iv. Less arguments in function

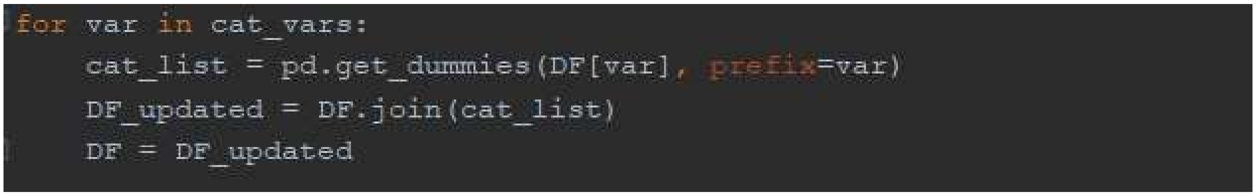


Better Function (More than three arguments are evil)

```
# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
# Logistic Regression Model
classifier = LogisticRegression(random_state=0)
classifier.fit(X_final_train, y_final_train)
y_pred = classifier.predict(X_final_test)
```

v. Use correct Indentation



```
for var in cat_vars:
    cat_list = pd.get_dummies(DF[var], prefix=var)
    DF_updated = DF.join(cat_list)
    DF = DF_updated
```

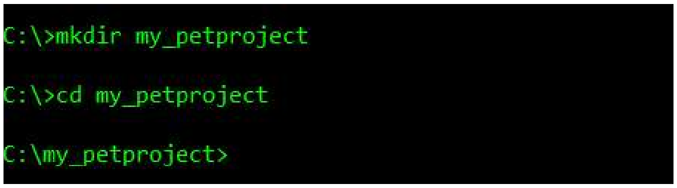
VI. Build Management

For build management, ***PyBuilder*** has been used in this project. PyBuilder is a multi-purpose software build tool. Most commonly it targets the building and management of software with a strong focus on Python. The advantages of PyBuilder are –

- Automatic execution of unit and integration tests on every build
- Automatic analysis of the code coverage
- Automatic execution and result interpretation of analysis tools, such as flake8
- Automatic generation of distutils script setup.py

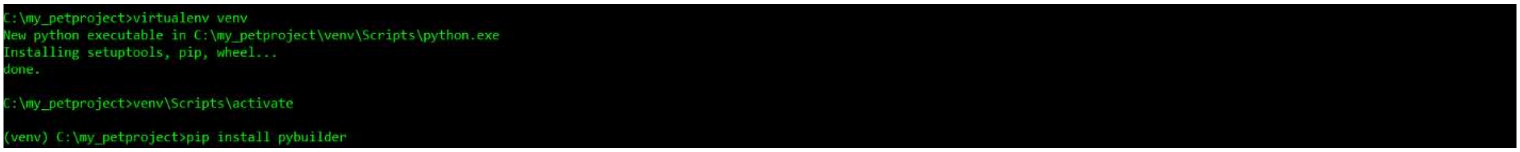
A complete walkthrough for the pybuilder project is given as follows:

i. Installing PyBuilder:

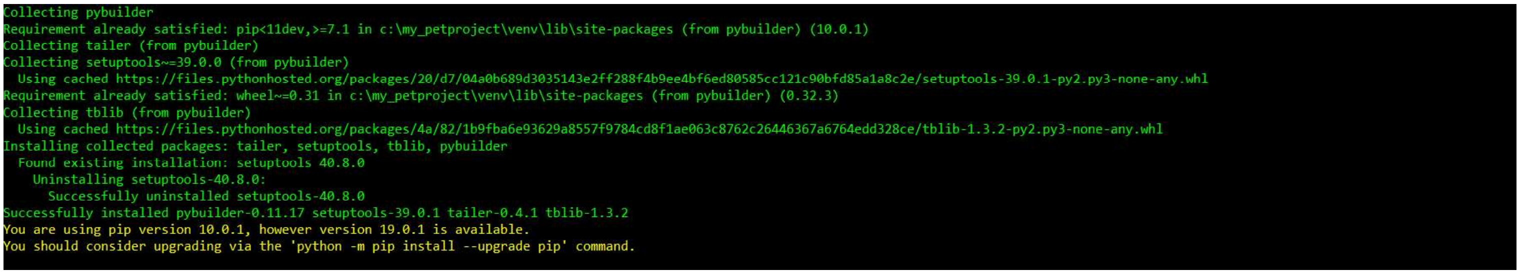


```
C:\>mkdir my_petproject

C:\>cd my_petproject

C:\my_petproject>
```

ii. Create Virtualenvironment and installing PyBuild:

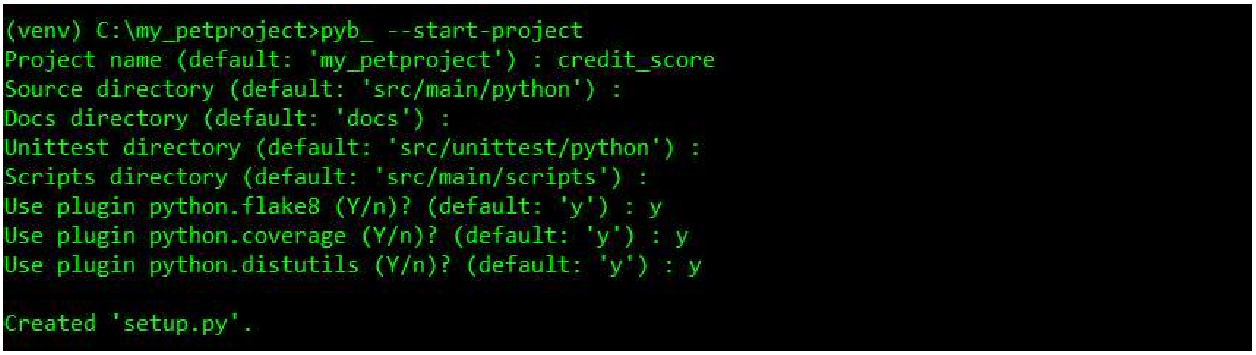


```
C:\my_petproject>virtualenv venv
New python executable in C:\my_petproject\venv\Scripts\python.exe
Installing setuptools, pip, wheel...
done.

C:\my_petproject>venv\Scripts\activate

(venv) C:\my_petproject>pip install pybuilder
```



```
Collecting pybuilder
Requirement already satisfied: pip<11dev,>=7.1 in c:\my_petproject\venv\lib\site-packages (from pybuilder) (10.0.1)
Collecting tailer (from pybuilder)
Collecting setuptools~=39.0.0 (from pybuilder)
  Using cached https://files.pythonhosted.org/packages/20/d7/04a0b689d3035143e2ff288f4b9ee4bf6ed80585cc121c90bfd85a1a8c2e/setuptools-39.0.1-py2.py3-none-any.whl
Requirement already satisfied: wheel~=0.31 in c:\my_petproject\venv\lib\site-packages (from pybuilder) (0.32.3)
Collecting tblib (from pybuilder)
  Using cached https://files.pythonhosted.org/packages/4a/82/1b9fba6e93629a8557f9784cd8f1ae063c8762c26446367a6764edd328ce/tblib-1.3.2-py2.py3-none-any.whl
Installing collected packages: tailer, setuptools, tblib, pybuilder
  Found existing installation: setuptools 40.8.0
    Uninstalling setuptools-40.8.0:
      Successfully uninstalled setuptools-40.8.0
Successfully installed pybuilder-0.11.17 setuptools-39.0.1 tailer-0.4.1 tblib-1.3.2
You are using pip version 10.0.1, however version 19.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

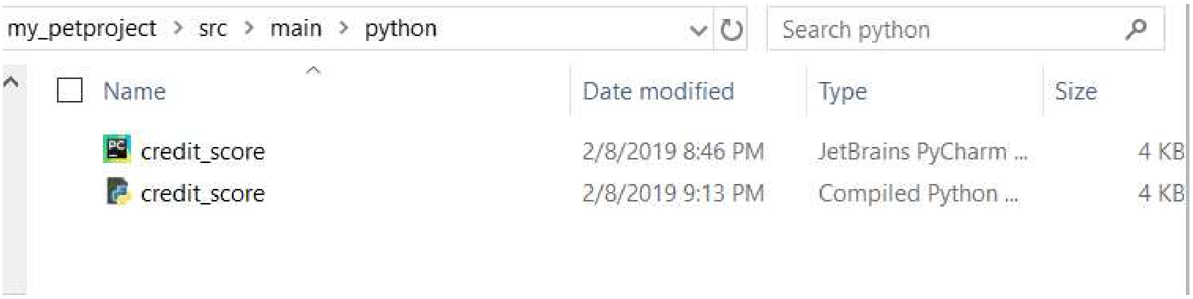
iii. Scaffolding:



```
(venv) C:\my_petproject>pyb_ --start-project
Project name (default: 'my_petproject') : credit_score
Source directory (default: 'src/main/python') :
Docs directory (default: 'docs') :
Unittest directory (default: 'src/unittest/python') :
Scripts directory (default: 'src/main/scripts') :
Use plugin python.flake8 (Y/n)? (default: 'y') : y
Use plugin python.coverage (Y/n)? (default: 'y') : y
Use plugin python.distutils (Y/n)? (default: 'y') : y

Created 'setup.py'.
```

iv. Adding Python Source Files:

The next step is to add a Python module that contains our sources. PyBuilder separates source files and expects them in different directories based on their meaning. The default location for main python sources is src/main/python.

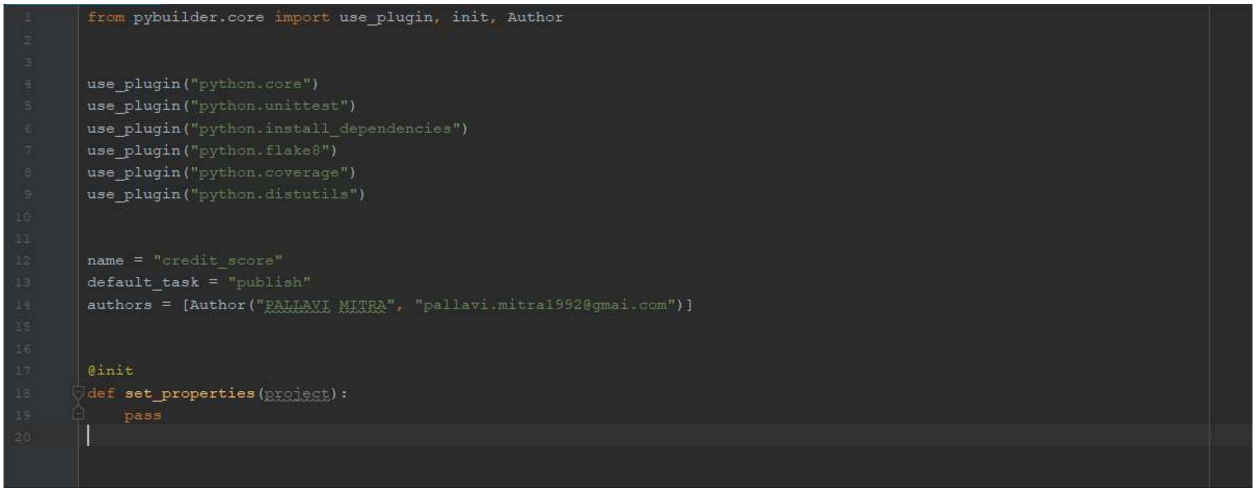


my_petproject > src > main > python

Name	Date modified	Type	Size
credit_score	2/8/2019 8:46 PM	JetBrains PyCharm ...	4 KB
credit_score	2/8/2019 9:13 PM	Compiled Python ...	4 KB

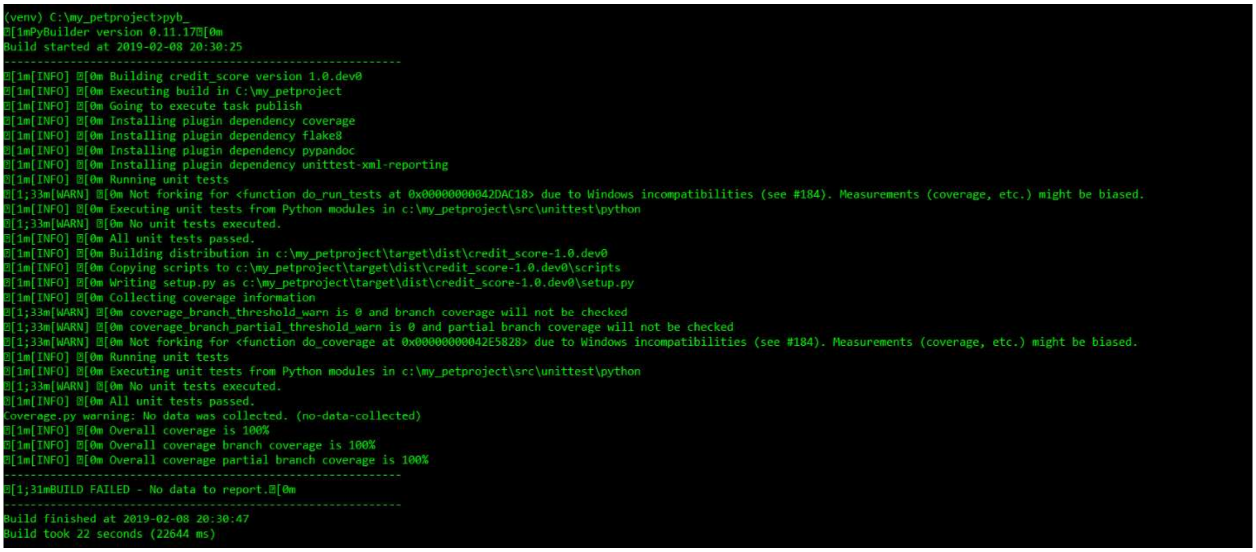
v. New build.py:

build.py is the centralized project description for the pet project.



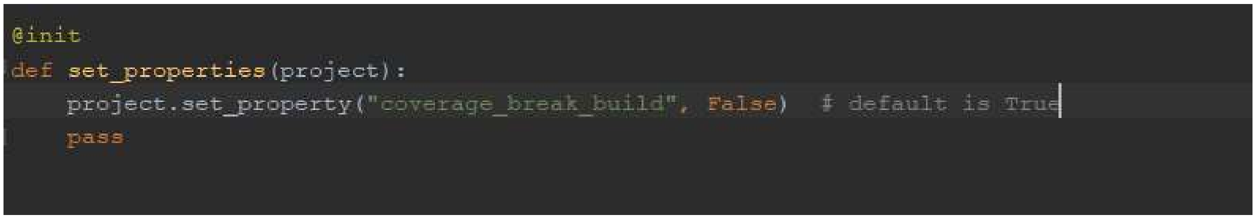
```python
from pybuilder.core import use_plugin, init, Author


use_plugin("python.core")
use_plugin("python.unittest")
use_plugin("python.install_dependencies")
use_plugin("python.flake8")
use_plugin("python.coverage")
use_plugin("python.distutils")


name = "credit_score"
default_task = "publish"
authors = [Author("PALLAVI MITRA", "pallavi.mitra1992@gmai.com")]


@init
def set_properties(project):
    pass

```

Let’s run PyBuilder and see what happens:



```
(venv) C:\my_petproject>pyb_
▯[1mPyBuilder version 0.11.17▯[0m
Build started at 2019-02-08 20:30:25
------------------------------------------------------------
▯[1m[INFO] ▯[0m Building credit_score version 1.0.dev0
▯[1m[INFO] ▯[0m Executing build in C:\my_petproject
▯[1m[INFO] ▯[0m Going to execute task publish
▯[1m[INFO] ▯[0m Installing plugin dependency coverage
▯[1m[INFO] ▯[0m Installing plugin dependency flake8
▯[1m[INFO] ▯[0m Installing plugin dependency pypandoc
▯[1m[INFO] ▯[0m Installing plugin dependency unittest-xml-reporting
▯[1m[INFO] ▯[0m Running unit tests
▯[1;33m[WARN] ▯[0m Not forking for <function do_run_tests at 0x00000000042DAC18> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.
▯[1m[INFO] ▯[0m Executing unit tests from Python modules in c:\my_petproject\src\unittest\python
▯[1;33m[WARN] ▯[0m No unit tests executed.
▯[1m[INFO] ▯[0m All unit tests passed.
▯[1m[INFO] ▯[0m Building distribution in c:\my_petproject\target\dist\credit_score-1.0.dev0
▯[1m[INFO] ▯[0m Copying scripts to c:\my_petproject\target\dist\credit_score-1.0.dev0\scripts
▯[1m[INFO] ▯[0m Writing setup.py as c:\my_petproject\target\dist\credit_score-1.0.dev0\setup.py
▯[1m[INFO] ▯[0m Collecting coverage information
▯[1;33m[WARN] ▯[0m coverage_branch_threshold_warn is 0 and branch coverage will not be checked
▯[1;33m[WARN] ▯[0m coverage_branch_partial_threshold_warn is 0 and partial branch coverage will not be checked
▯[1;33m[WARN] ▯[0m Not forking for <function do_coverage at 0x00000000042E5828> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.
▯[1m[INFO] ▯[0m Running unit tests
▯[1m[INFO] ▯[0m Executing unit tests from Python modules in c:\my_petproject\src\unittest\python
▯[1;33m[WARN] ▯[0m No unit tests executed.
▯[1m[INFO] ▯[0m All unit tests passed.
Coverage.py warning: No data was collected. (no-data-collected)
▯[1m[INFO] ▯[0m Overall coverage is 100%
▯[1m[INFO] ▯[0m Overall coverage branch coverage is 100%
▯[1m[INFO] ▯[0m Overall coverage partial branch coverage is 100%
------------------------------------------------------------
▯[1;31mBUILD FAILED - No data to report.▯[0m
------------------------------------------------------------
Build finished at 2019-02-08 20:30:47
Build took 22 seconds (22644 ms)
```

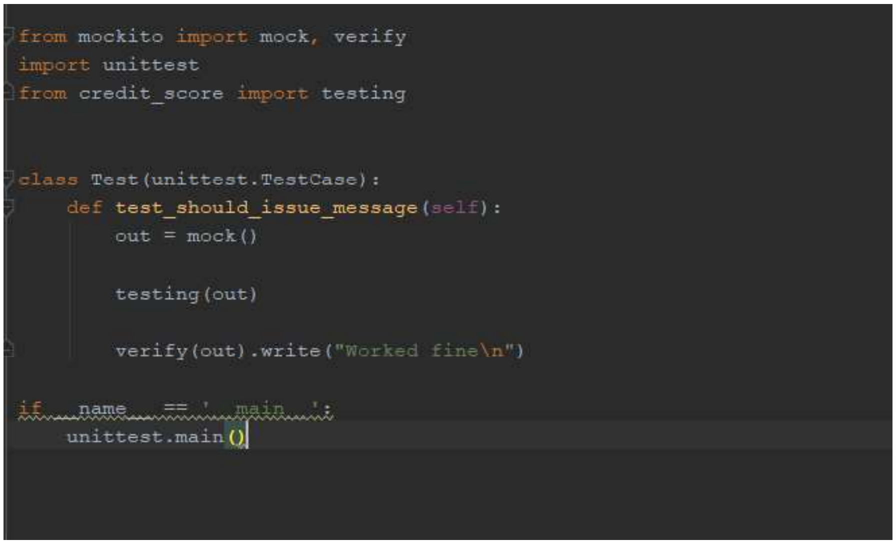
We don’t have any tests right now. To avoid the break in the build, we need to modify build.py.



```python
@init
def set_properties(project):
    project.set_property("coverage_break_build", False)  # default is True
    pass
```

vi. Unittest Plugin:

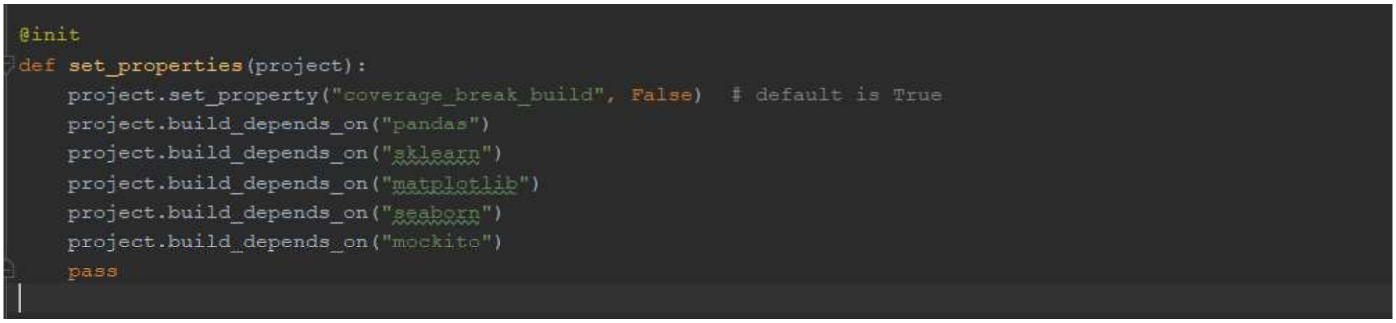
Let’s start with a test at src/unittest/python/credit_score_tests.py:



```python
from mockito import mock, verify
import unittest
from credit_score import testing


class Test(unittest.TestCase):
    def test_should_issue_message(self):
        out = mock()

        testing(out)

        verify(out).write("Worked fine\n")

if __name__ == '__main__':
    unittest.main()
```

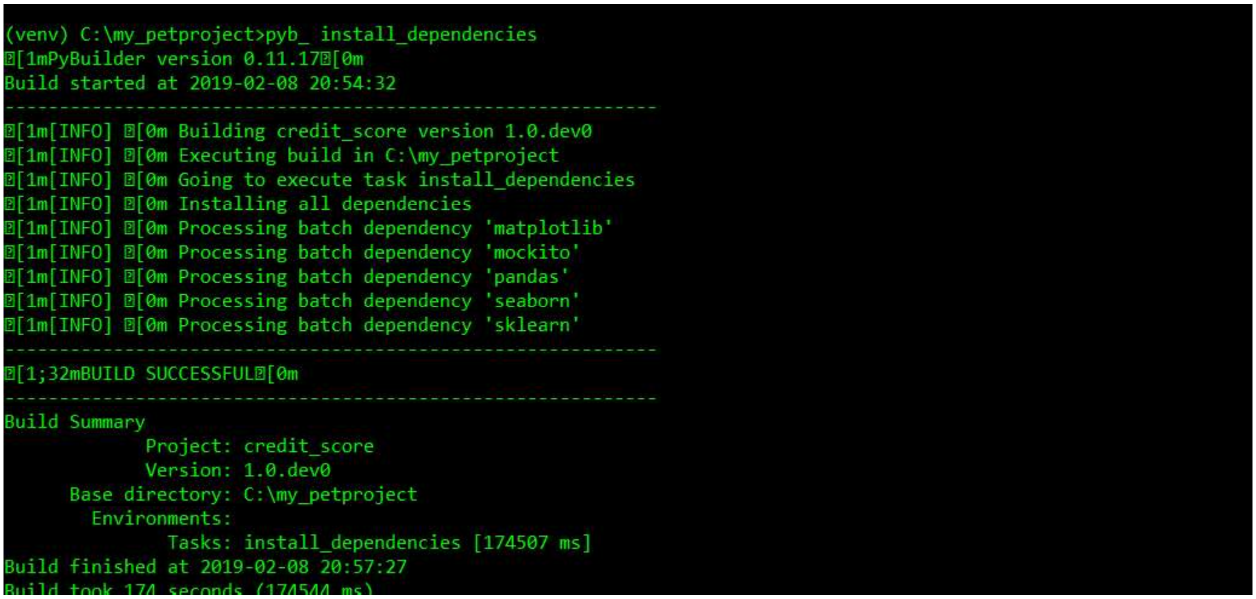
vii. Dependency:

Since we're using various libraries of python, we'll have to install it by telling our initializer in build.py about it:



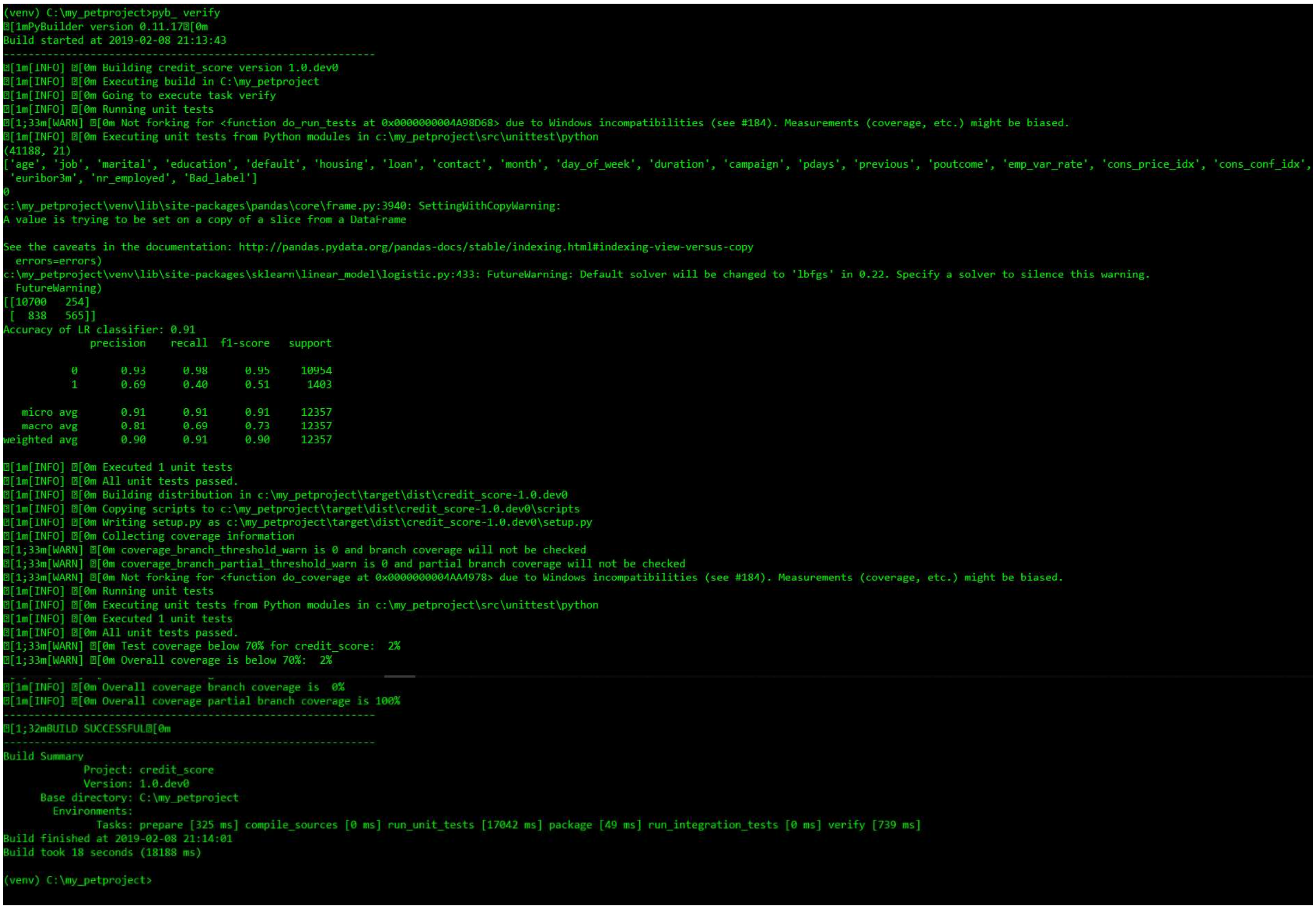
```
@init
def set_properties(project):
    project.set_property("coverage_break_build", False)  # default is True
    project.build_depends_on("pandas")
    project.build_depends_on("sklearn")
    project.build_depends_on("matplotlib")
    project.build_depends_on("seaborn")
    project.build_depends_on("mockito")
    pass
```

We can install our dependency by running PyBuilder with the corresponding task



```
(venv) C:\my_petproject>pyb_ install_dependencies
□[1mPyBuilder version 0.11.17□[0m
Build started at 2019-02-08 20:54:32
------------------------------------------------------------
□[1m[INFO] □[0m Building credit_score version 1.0.dev0
□[1m[INFO] □[0m Executing build in C:\my_petproject
□[1m[INFO] □[0m Going to execute task install_dependencies
□[1m[INFO] □[0m Installing all dependencies
□[1m[INFO] □[0m Processing batch dependency 'matplotlib'
□[1m[INFO] □[0m Processing batch dependency 'mockito'
□[1m[INFO] □[0m Processing batch dependency 'pandas'
□[1m[INFO] □[0m Processing batch dependency 'seaborn'
□[1m[INFO] □[0m Processing batch dependency 'sklearn'
------------------------------------------------------------
□[1;32mBUILD SUCCESSFUL□[0m
------------------------------------------------------------
Build Summary
             Project: credit_score
             Version: 1.0.dev0
      Base directory: C:\my_petproject
        Environments:
               Tasks: install_dependencies [174507 ms]
Build finished at 2019-02-08 20:57:27
Build took 174 seconds (174544 ms)
```

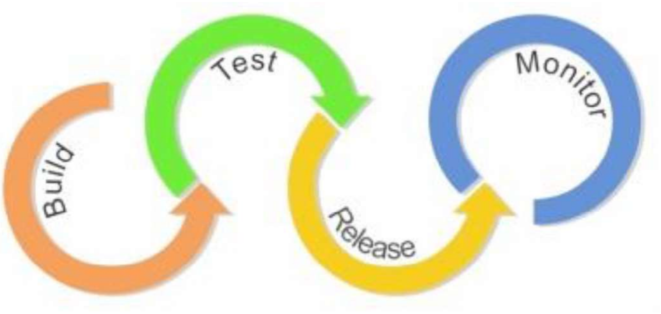
viii. Running the test:



```
(venv) C:\my_petproject>pyb_ verify
⍰[1mPyBuilder version 0.11.17⍰[0m
Build started at 2019-02-08 21:13:43
------------------------------------------------------------
⍰[1m[INFO] ⍰[0m Building credit_score version 1.0.dev0
⍰[1m[INFO] ⍰[0m Executing build in C:\my_petproject
⍰[1m[INFO] ⍰[0m Going to execute task verify
⍰[1m[INFO] ⍰[0m Running unit tests
⍰[1;33m[WARN] ⍰[0m Not forking for <function do_run_tests at 0x0000000004A98D68> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.
⍰[1m[INFO] ⍰[0m Executing unit tests from Python modules in c:\my_petproject\src\unittest\python
(41188, 21)
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx',
 'euribor3m', 'nr_employed', 'Bad_label']
0
c:\my_petproject\venv\lib\site-packages\pandas\core\frame.py:3940: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  errors=errors)
c:\my_petproject\venv\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
[[10700   254]
 [  838   565]]
Accuracy of LR classifier: 0.91
              precision    recall  f1-score   support

           0       0.93      0.98      0.95     10954
           1       0.69      0.40      0.51      1403

   micro avg       0.91      0.91      0.91     12357
   macro avg       0.81      0.69      0.73     12357
weighted avg       0.90      0.91      0.90     12357

⍰[1m[INFO] ⍰[0m Executed 1 unit tests
⍰[1m[INFO] ⍰[0m All unit tests passed.
⍰[1m[INFO] ⍰[0m Building distribution in c:\my_petproject\target\dist\credit_score-1.0.dev0
⍰[1m[INFO] ⍰[0m Copying scripts to c:\my_petproject\target\dist\credit_score-1.0.dev0\scripts
⍰[1m[INFO] ⍰[0m Writing setup.py as c:\my_petproject\target\dist\credit_score-1.0.dev0\setup.py
⍰[1m[INFO] ⍰[0m Collecting coverage information
⍰[1;33m[WARN] ⍰[0m coverage_branch_threshold_warn is 0 and branch coverage will not be checked
⍰[1;33m[WARN] ⍰[0m coverage_branch_partial_threshold_warn is 0 and partial branch coverage will not be checked
⍰[1;33m[WARN] ⍰[0m Not forking for <function do_coverage at 0x0000000004AA4978> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.
⍰[1m[INFO] ⍰[0m Running unit tests
⍰[1m[INFO] ⍰[0m Executing unit tests from Python modules in c:\my_petproject\src\unittest\python
⍰[1m[INFO] ⍰[0m Executed 1 unit tests
⍰[1m[INFO] ⍰[0m All unit tests passed.
⍰[1;33m[WARN] ⍰[0m Test coverage below 70% for credit_score:  2%
⍰[1;33m[WARN] ⍰[0m Overall coverage is below 70%:  2%
⍰[1m[INFO] ⍰[0m Overall coverage branch coverage is  0%
⍰[1m[INFO] ⍰[0m Overall coverage partial branch coverage is 100%
------------------------------------------------------------
⍰[1;32mBUILD SUCCESSFUL⍰[0m
------------------------------------------------------------
Build Summary
             Project: credit_score
             Version: 1.0.dev0
      Base directory: C:\my_petproject
        Environments:
               Tasks: prepare [325 ms] compile_sources [0 ms] run_unit_tests [17042 ms] package [49 ms] run_integration_tests [0 ms] verify [739 ms]
Build finished at 2019-02-08 21:14:01
Build took 18 seconds (18188 ms)

(venv) C:\my_petproject>
```

VII. CONTINUOUS INTEGRATION

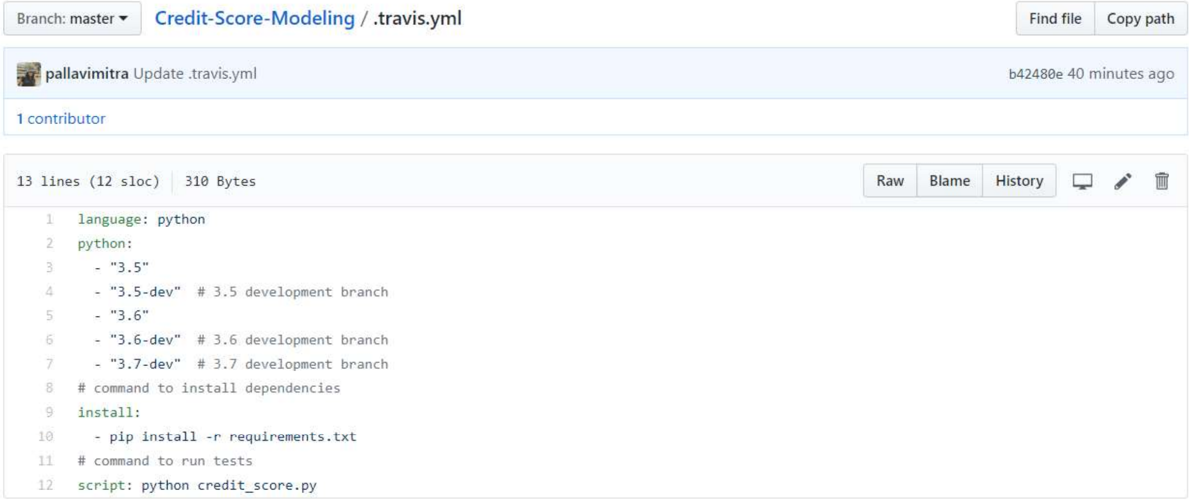
Continuous Integration is a development practice of integrating code into a single build multiple times per day. Builds are usually automated and depend on thorough automatic test to ensure they complete properly and without issue.



Here, "***Travis CI"*** has been used for continuous integration. It's built to communicate with github and can be set up to build for every push and pull in request that comes in for the repository.

To get started with Travis Ci followings steps must be taken.

1. Go to Travis-ci.com and *Sign up with GitHub*.
2. Accept the Authorization of Travis CI and redirect to GitHub.
3. Click the green *Activate* button and select the repository that want to use with Travis CI.
4. Add a .travis.yml file to the repository to tell Travis CI what to do.



Credit-Score-Modeling / .travis.yml

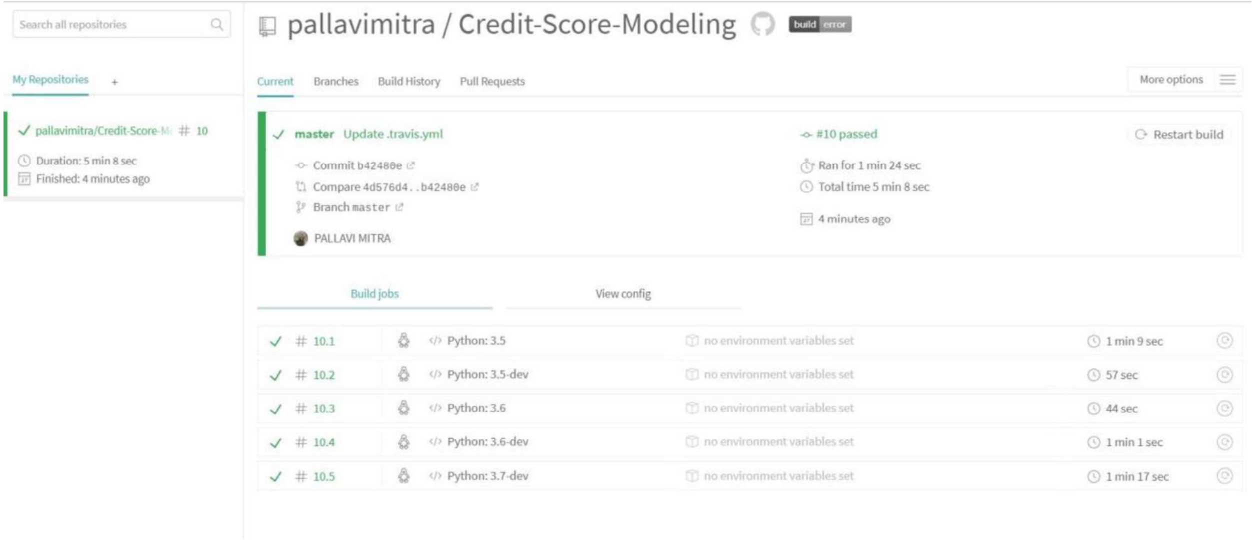
pallavimitra Update .travis.yml b42480e 40 minutes ago

1 contributor

13 lines (12 sloc) | 310 Bytes

```
language: python
python:
  - "3.5"
  - "3.5-dev"  # 3.5 development branch
  - "3.6"
  - "3.6-dev"  # 3.6 development branch
  - "3.7-dev"  # 3.7 development branch
# command to install dependencies
install:
  - pip install -r requirements.txt
# command to run tests
script: python credit_score.py
```

5. Add the .travis.yml file to git, commit and push, to trigger a Travis CI build.
6. Check the build status page to see if your build passes or fails.



pallavimitra / Credit-Score-Modeling

✓ pallavimitra/Credit-Score-M # 10

Duration: 5 min 8 sec

Finished: 4 minutes ago

✓ master Update .travis.yml

Commit b42480e

Compare 4d576d4..b42480e

Branch master

PALLAVI MITRA

#10 passed

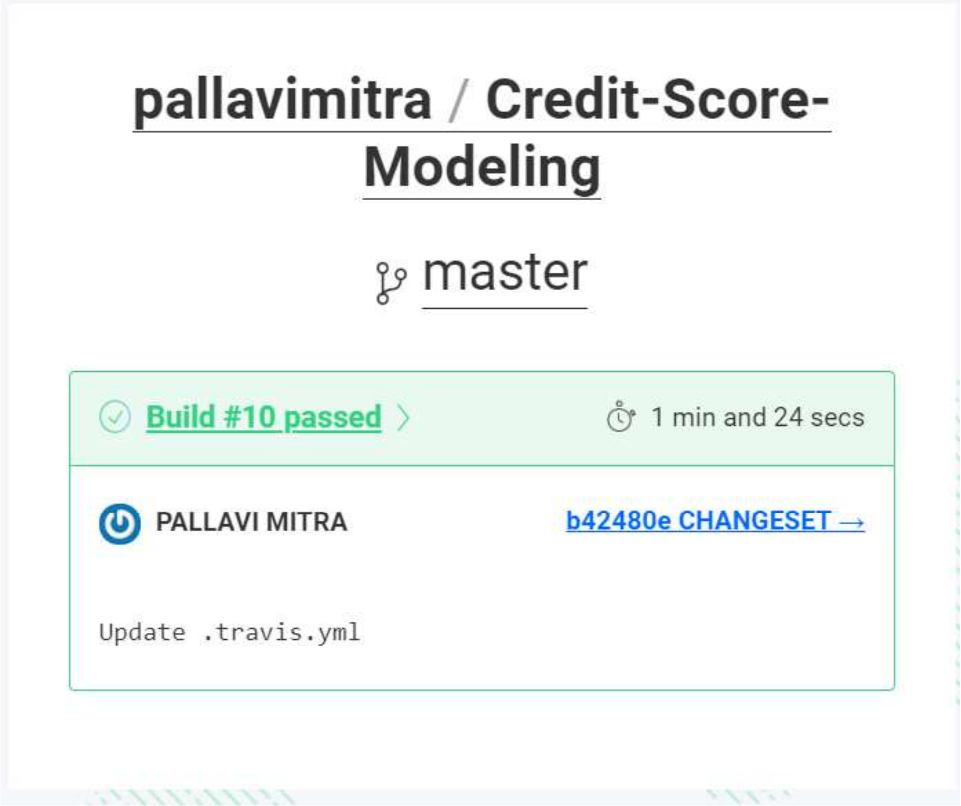
Ran for 1 min 24 sec

Total time 5 min 8 sec

4 minutes ago

Build jobs | View config

✓	# 10.1	Python: 3.5	no environment variables set	1 min 9 sec
✓	# 10.2	Python: 3.5-dev	no environment variables set	57 sec
✓	# 10.3	Python: 3.6	no environment variables set	44 sec
✓	# 10.4	Python: 3.6-dev	no environment variables set	1 min 1 sec
✓	# 10.5	Python: 3.7-dev	no environment variables set	1 min 17 sec



pallavimitra / Credit-Score-Modeling

master

Build #10 passed

1 min and 24 secs

PALLAVI MITRA

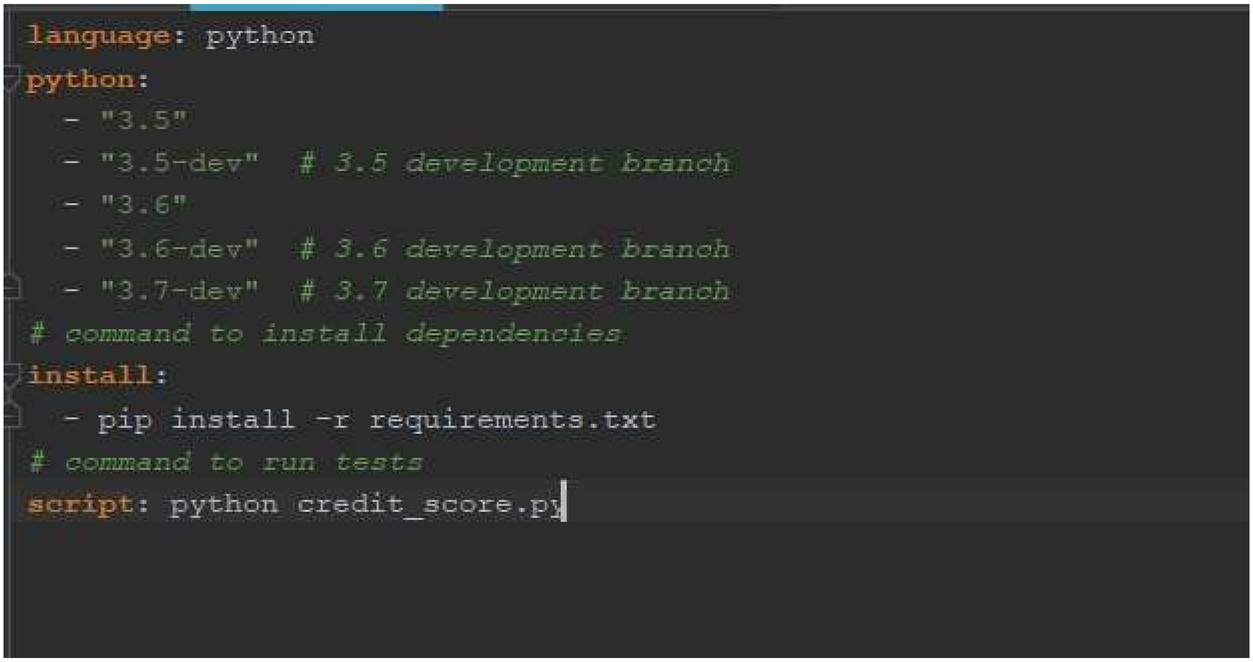
b42480e CHANGESET →

```
Update .travis.yml
```

VIII. DOMAIN SPECIFIC LANGUAGES

A ***domain-specific language*** *(DSL)* is a computer language specialized to a particular application domain. There are a wide variety of DSLs, ranging from widely used languages for common domains, such as HTML for web pages, down to languages used by only one or a few pieces of software.

In the previous section i.e. in continuous integration, we have to use a yml code to trigger Travis CI build. So, this .travis.yml can be considered as DSL for this pet project.



```
language: python
python:
  - "3.5"
  - "3.5-dev"  # 3.5 development branch
  - "3.6"
  - "3.6-dev"  # 3.6 development branch
  - "3.7-dev"  # 3.7 development branch
# command to install dependencies
install:
  - pip install -r requirements.txt
# command to run tests
script: python credit_score.py
```

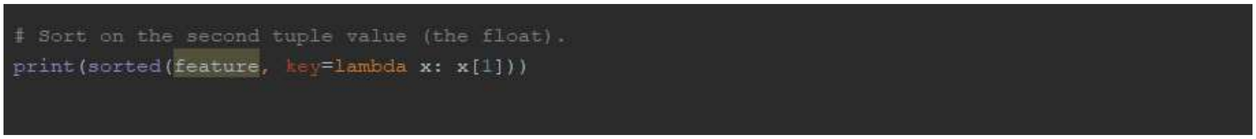
IX. FUNCTIONAL PROGRAMMING

Functional programming languages are specially designed to handle symbolic computation and list processing applications. Functional programming is based on mathematical functions. Some of the popular functional programming languages include: Lisp, Python, Erlang, Haskell, Clojure, etc.

Following functional programming have been utilized in the pet project.

h. The Lambda Expression:

Instead of the def syntax for function declaration, we can use a lambda expression to write Python functions. The lambda expression takes in a comma separated sequences of inputs (like def). Then, immediately following the colon, it returns the expression without using an explicit return statement. Finally, when assigning the lambda expression to a variable, it acts exactly like a Python function, and can be called using the function call syntax.

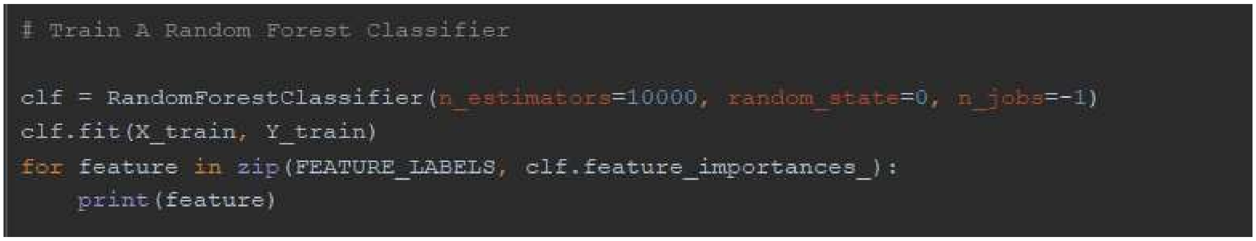


```
# Sort on the second tuple value (the float).
print(sorted(feature, key=lambda x: x[1]))
```

Here, the sorted() function takes in an optional key argument (a function) that describes how the items in a list should be sorted.

ii. Zip Function:

zip is a common functional programming method like map or fold. They are designed to perform common batch operations on lists. The zip() function take iterables (can be zero or more), makes iterator that aggregates elements based on the iterables passed, and returns an iterator of tuples.



```
# Train A Random Forest Classifier

clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)
clf.fit(X_train, Y_train)
for feature in zip(FEATURE_LABELS, clf.feature_importances_):
    print(feature)
```