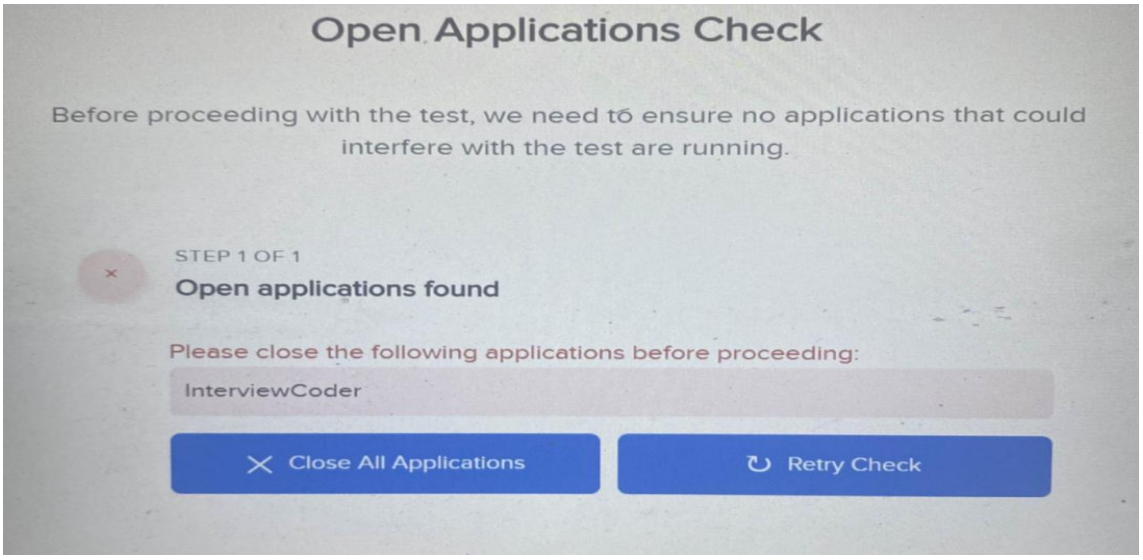
Common Issues-Summative Exam

HackerEarth Platform and Coding

HackerEarth platform has no issues as such. The issues raised by the learners are the configuration on the environment setup for the examination.

Few issues which are frequently raised by the learners are captured and are addressed below.

Issue 1

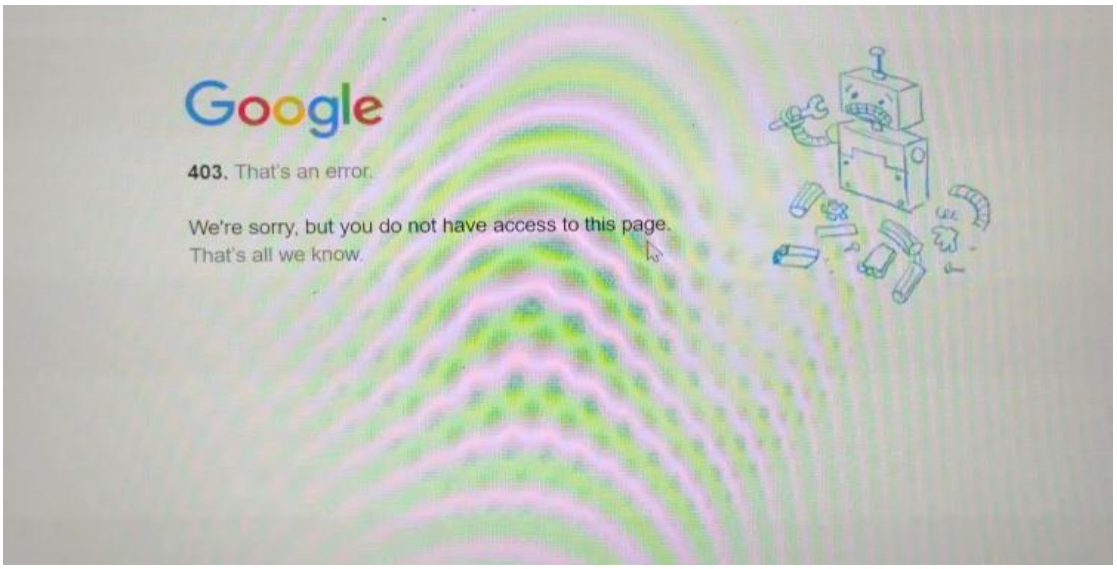


Close all the applications opened and if they have added any extensions in google chrome which it is specifying as in the above image “Please close the following applications before proceeding” **InterviewCoder**(it is an extension) remove the extension.

Issue 2

click on **More Info** and then **run anyway**

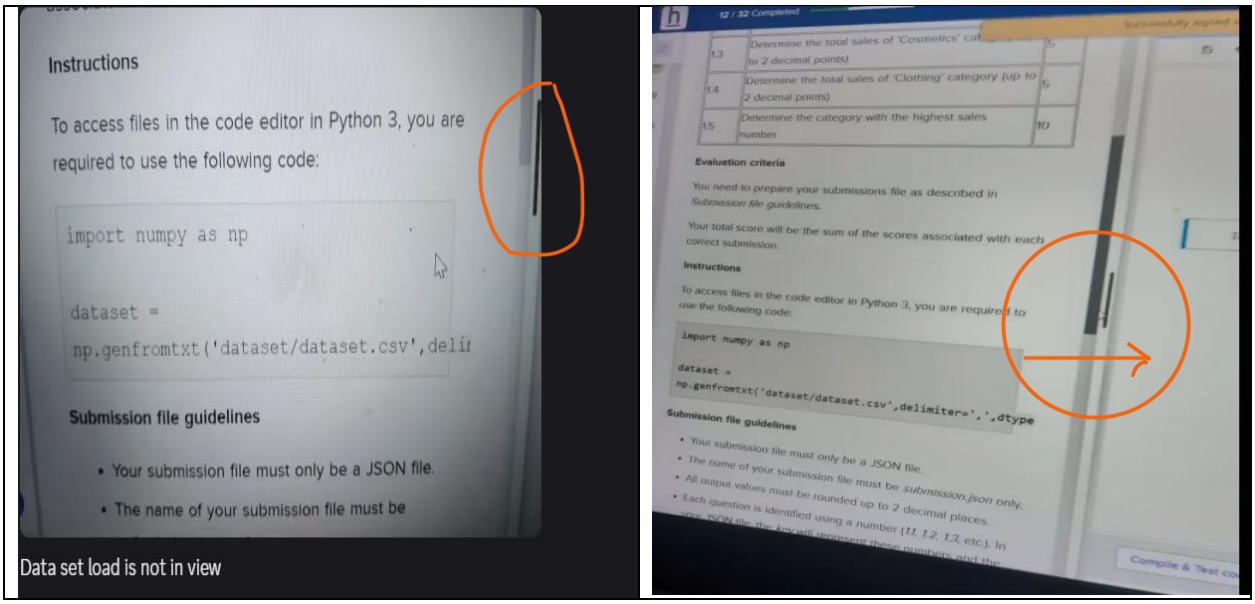
Issue 3



Instead of sign in with Google. Try giving email and password in the input boxes and try login else reset the password.

Still same issue they need to repeat the process 2 to 3 times.

Issue 4

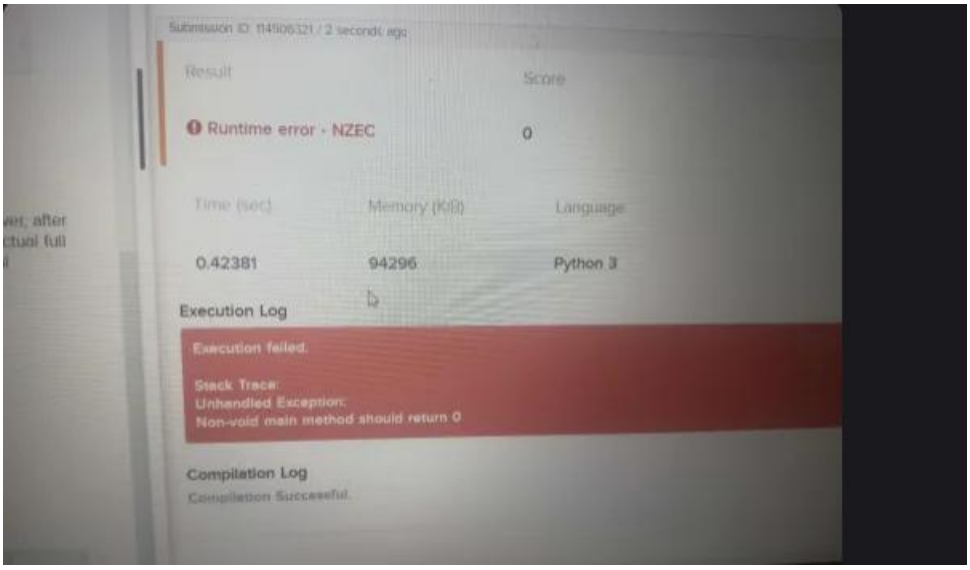


Scroll it the code will be visible.

Don’t copy and paste it will not work

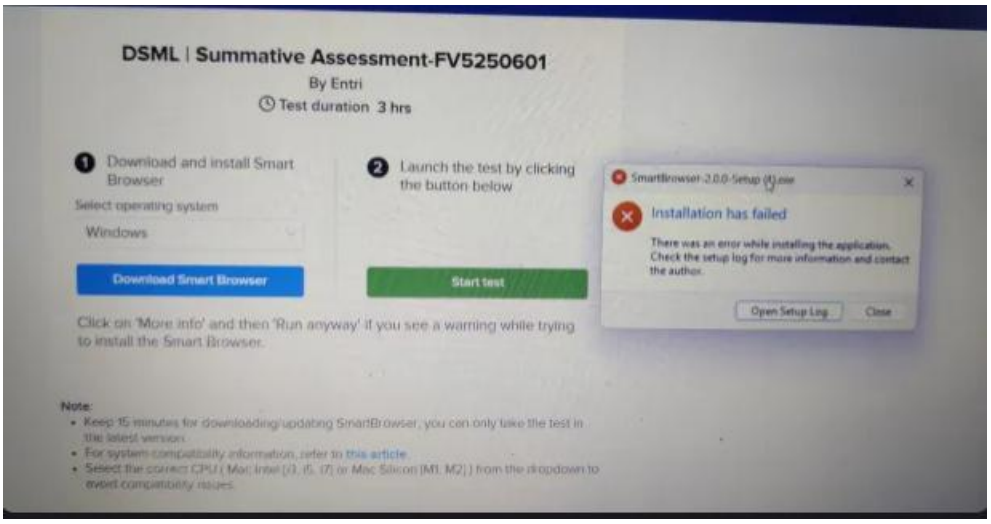
You need to type it

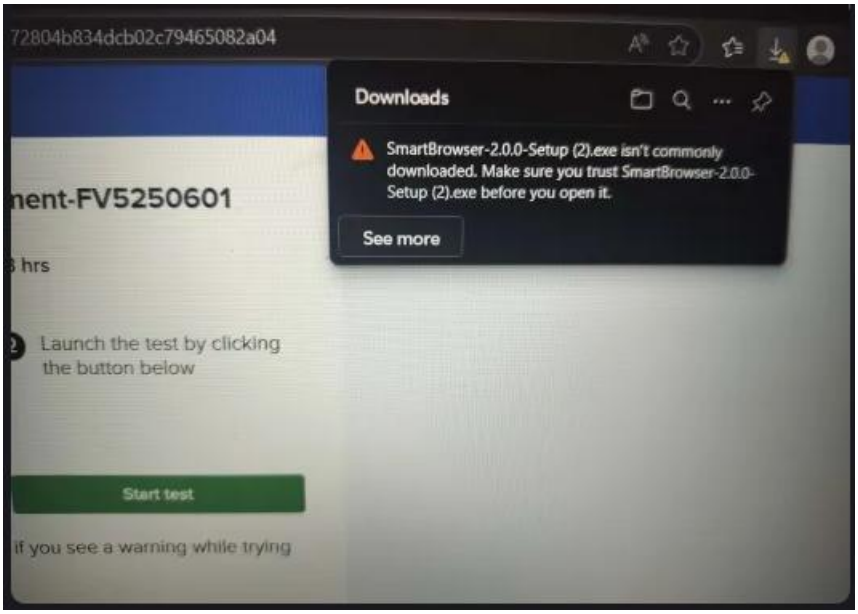
Issue 5



Something went wrong in your code.Try to rectify it

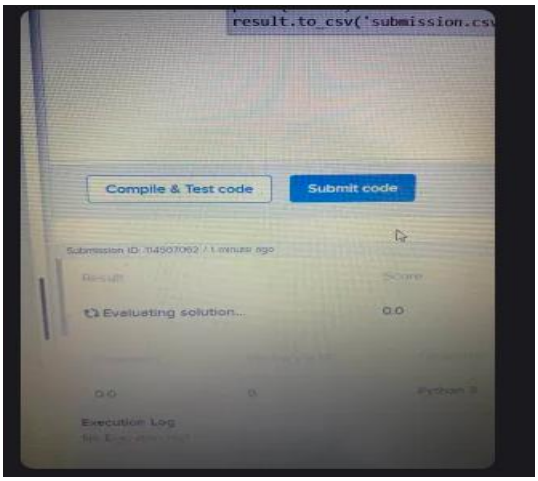
Issue 6





Uninstall the antivirus which has been installed in your PC and the install the smart browser.

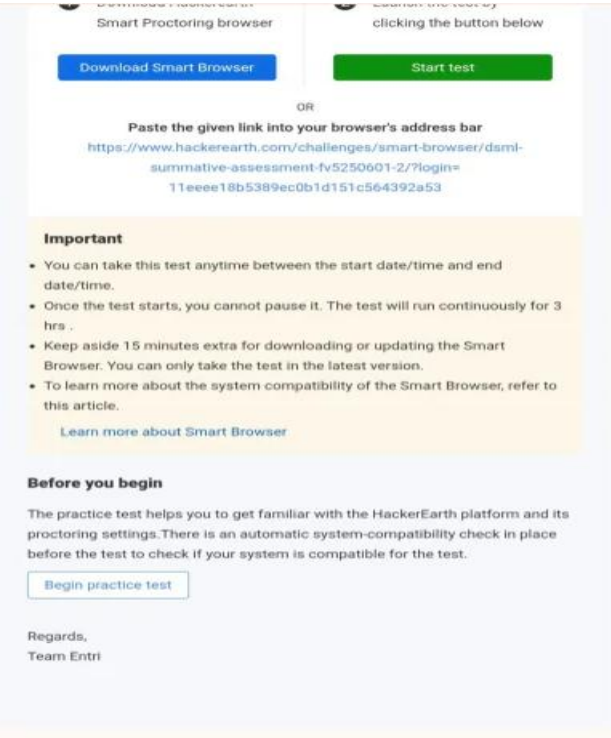
Issue 7



In case if the button submit is not visible you may be in full screen mode come out of it will be visible.

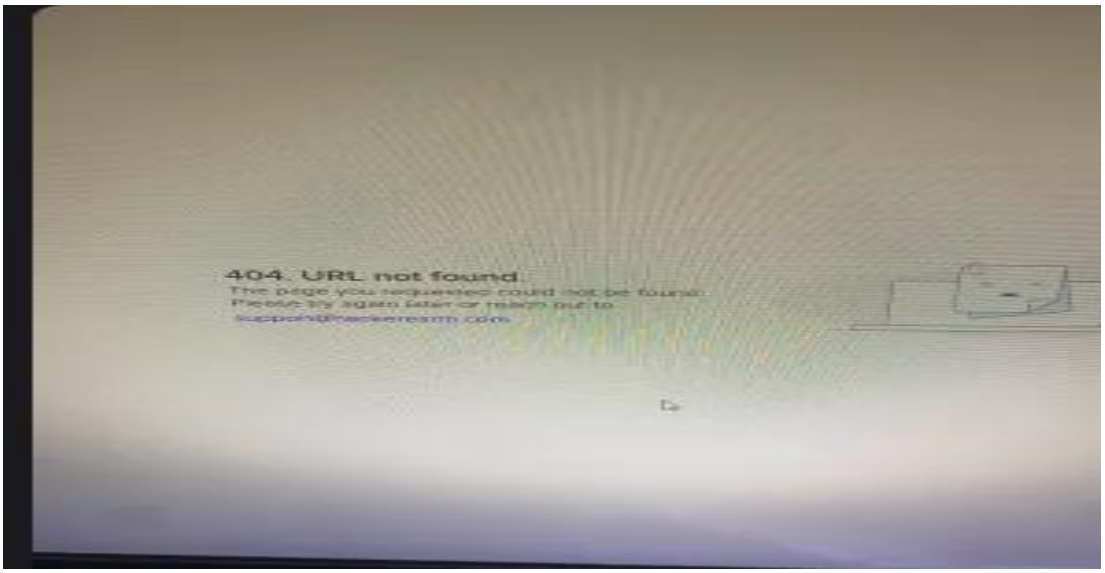
In case execution is slow or if evaluating solution is still active it may be network issue or coding issue which takes few minutes.

Issue 8



Practice test will work after the installation of smart browser.do download and install it and then start the practice exam.

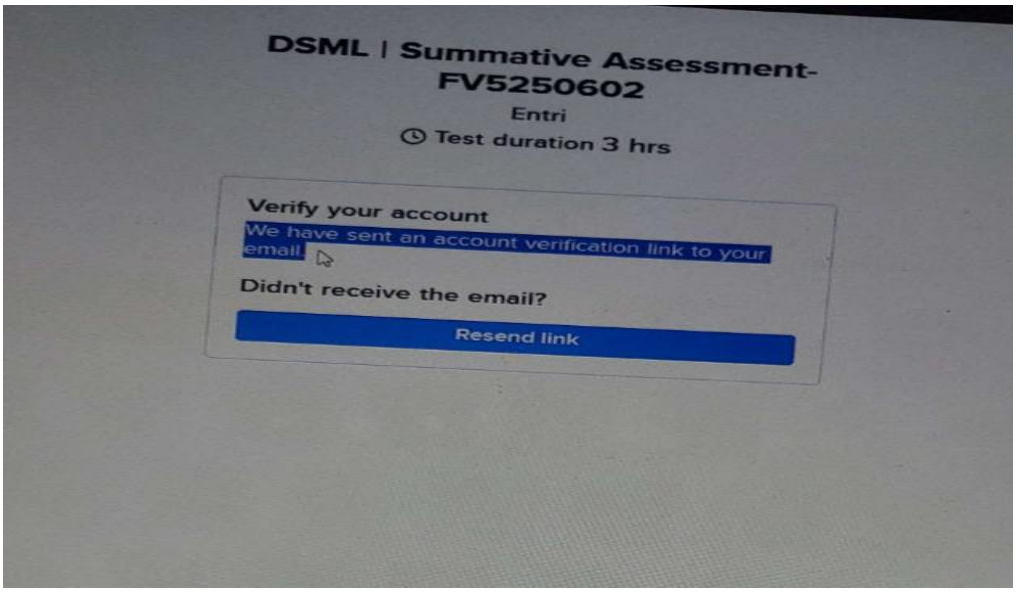
Issue 9



Incorrect link

After installing smart browser if the above screen is displayed restart the system and login again.

Issue 10



Open your mail and click on the verification link if not received click on resend link

Check in inbox,updates and spam also if not received in inbox

Coding Issues

```
model=LogisticRegression()
model.fit(X_train,Y_train)
```

```
/opt/conda/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
▾ LogisticRegression
LogisticRegression()
```

How to fix it:

The error message explicitly gives you the solution: **"Increase the number of iterations (max_iter) or scale the data as shown in..."**

the primary solution here is to **increase max_iter** in your LogisticRegression model.

model = LogisticRegression(max_iter=1000)

```
submission=pd.DataFrame({
    "StudentID":X_test["StudentID"],
    "GraduationOnTime":y_pred
})
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-62-acd9e35d4e82> in <module>
      1 submission=pd.DataFrame({
----> 2     "StudentID":X_test["StudentID"],
      3     "GraduationOnTime":y_pred
      4 })

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`) and integer or boolean arrays
are valid indices
```

```
submission.to_csv("submission,csv",index=False)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-59-fd6b74b62bcb> in <module>
----> 1 submission.to_csv("submission,csv",index=False)

NameError: name 'submission' is not defined
```

How to fix it:

The traceback clearly states: IndexError: only integers, slices (':'), ellipsis ('...'), numpy.newaxis ('None') and integer or boolean arrays are valid indices.

This error occurs on the line: X_test["StudentID"].

X_test is likely a NumPy array or a similar structure that doesn't support string-based column indexing like Pandas DataFrames.

```
submission = pd.DataFrame({
        "Student ID":StudentID,
        "GraduationOnTime": y_pred
        })
```

The traceback states: NameError: name 'submission' is not defined.

This error occurs on the line: submission.to_csv("submission,csv", index=False).

This means that the variable submission was not created or assigned a value before you tried to use it. This error is a **consequence** of the IndexError. Because the cell with your pd.DataFrame creation failed due to the IndexError, the submission variable was never successfully created.

submission.to_csv("submission.csv", index=False)

The instructions given in the question

Evaluation criteria

You need to prepare your submissions file as described in *Submission file guidelines.*

Your total score will be the sum of the scores associated with each correct submission.

Instructions

To access files in the code editor in Python 3, you are required to use the following code:

```
import numpy as np

dataset =
np.genfromtxt('dataset/dataset.csv',delimiter=',',dtype='
str')
```

Learners mistakes

```
import numpy as np
```

```
dataset=np.genfromtxt('dataset/dataset.csv',delimiter=',')
```

```
dataset
```

```
array([[    nan,     nan,  704.39],
       [    nan,     nan,  108.3 ],
       [    nan,     nan, 1988.4 ],
       [    nan,     nan, 1506.78],
       [    nan,     nan, 1861.89],
       [    nan,     nan, 2036.48],
       [    nan,     nan, 2347.46],
       [    nan,     nan,  216.76]
```

The "nan" values you're seeing in your NumPy array dataset after using np.genfromtxt() indicate "Not a Number." This typically happens when np.genfromtxt() encounters non-numeric data in your CSV file where it expects numbers, or if it encounters empty cells.

How to fix it

The answer is in question only

dataset = np.genfromtxt('dataset/dataset.csv', delimiter=',',dtype='str')

```
numerical_transformer=Pipeline(steps=[
    ('imputer',SimpleImputer(Strategy='median')),
    ('scalar',StandardScaler())])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-46-203840e6a377> in <module>
----> 1 numerical_transformer=Pipeline(steps=[('imputer',SimpleImputer(Strategy='median')),('scalar',StandardSca
ler())])

TypeError: __init__() got an unexpected keyword argument 'Strategy'
```

```
catrgorical_transformer=Pipeline(steps=[
    ('imputer',SimpleImputer(Strategy='most_frequent')),
    ('onehot',OneHotEncoder(handle_unknown='ignore'))
])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-e8847fc2c2cd> in <module>
      1 catrgorical_transformer=Pipeline(steps=[
----> 2     ('imputer',SimpleImputer(Strategy='most_frequent')),
      3     ('onehot',OneHotEncoder(handle_unknown='ignore'))
      4 ])

TypeError:   init  () got an unexpected keyword argument 'Strategy'
```

The error message __init__() got an unexpected keyword argument 'Strategy' indicates that the SimpleImputer class does not accept Strategy as a keyword argument for its initialization.

In scikit-learn's SimpleImputer, the parameter to specify the imputation strategy is strategy (lowercase 's'), not Strategy (uppercase 'S').

To fix the error

('imputer', SimpleImputer(strategy='median'))

('imputer', SimpleImputer(strategy='most_frequent'))

```
pipeline=Pipeline(steps=[('preprocessor',preprocessor),

                         'classifier'RandomForestClassifier(n_estimators100,random_state=42)
                        ])
```

```
  File "<ipython-input-49-8008fe175965>", line 3
    'classifier'RandomForestClassifier(n_estimators100,random_state=42)
                ^
SyntaxError: invalid syntax
```

```
pipeline.fit(X_train,y_train)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-50-512c1e5af757> in <module>
----> 1 pipeline.fit(X_train,y_train)

NameError: name 'pipeline' is not defined
```

```
prediction=pipeline.predict(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-51-0c2a3a6694ae> in <module>
----> 1 prediction=pipeline.predict(X_test)

NameError: name 'pipeline' is not defined
```

SyntaxError: invalid syntax in RandomForestClassifier line

- **Error:** classifier' RandomForestClassifier(n_estimators100, random_state=42) is causing a SyntaxError.

    1. There's no comma between n_estimators and its value 100.

Pipeline(steps=[('preprocessor', preprocessor),

('classifier', RandomForestClassifier(n_estimators=100, random_state=42)) # Corrected line

])

2. NameError: name 'pipeline' is not defined in pipeline.fit(X_train, y_train)

- **Error:** You're trying to use pipeline.fit(...) but Python says name 'pipeline' is not defined.
- **Reason:** In your Pipeline definition, you assigned the created pipeline object to a variable named Pipeline (with a capital 'P' at the beginning), not pipeline (lowercase 'p'). Python is case-sensitive.

How to fix it

pipeline = Pipeline(steps=[('preprocessor', preprocessor), # Make sure 'preprocessor' is defined ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))])

Name error

A NameError is a common error in Python that occurs when you try to use a variable or function name that hasn't been defined or recognized by the interpreter. Happens when working with libraries like scikit-learn, TensorFlow, or PyTorch, or when dealing with your own defined functions and variables.

What is a NameError?

A NameError is a type of LookupError in Python. It means that the Python interpreter cannot find a name (variable, function, class, module) that you're trying to reference in your code.

When is it Caused?

A NameError is typically caused when:

1. **Typo in Variable/Function Name:** You've misspelled the name of a variable, function, or class.
2. **Using a Variable Before Definition:** You're attempting to use a variable before it has been assigned a value.
3. **Unimported Module/Function:** You're trying to use a function or class from a module without first importing that module (or the specific function/class).

4. **Incorrect Scope:** A variable is defined in one scope (e.g., inside a function) and you're trying to access it from another scope where it's not visible.
5. **Deleted or Redeclared Variable:** A variable might have been deleted or overwritten, leading to its original name becoming undefined.

Why is it Caused in ML?

In Machine Learning, NameError is particularly common due to:

- **Reliance on Libraries:** ML heavily relies on external libraries. Forgetting to import numpy, pandas, sklearn, matplotlib.pyplot, etc., or specific components from them (e.g., train_test_split from sklearn.model_selection) is a frequent cause.
- **Complex Workflows:** ML projects often involve many steps: data loading, preprocessing, model definition, training, evaluation, and visualization. It's easy to make a typo or forget to define a variable at an earlier stage that's needed later.
- **Variable Naming Conventions:** With multiple datasets (training, testing, validation), models, and metrics, consistency in naming is crucial.
- **Notebook Environments (Jupyter):** In Jupyter notebooks, if you run cells out of order or restart the kernel without re-running all necessary cells, previously defined variables might become undefined.

How to Fix it in ML with an Example:

Let's illustrate with an example related to a common ML workflow.

Scenario: You're building a simple linear regression model.

```
# Cause of NameError: Unimported module

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error


# 1. Data Loading (Assume 'data.csv' exists with 'X' and 'y' columns)

data = pd.read_csv('data.csv')

X = data[['X']]
```

```
y = data['y']


# 2. Splitting the data (Potential NameError here)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# 3. Model Training (Potential NameError here)
model = LinearRegressor() # Typo: Should be LinearRegression


# 4. Prediction
predictions = model.predict(X_test)


# 5. Evaluation (Potential NameError here)
mse = mean_squared_error(y_test, predictions)
print(f"Mean Squared Error: {mse}")


# 6. Plotting (Potential NameError here)
plt.scatter(X_test, y_test)
plt.plot(X_test, predictions, color='red')
plt.xlabel("Features")
plt.ylabel("Target")
plt.title("Linear Regression")
plt.show()
```

Common NameErrors and their Fixes in the above code:

1. NameError: name 'train_test_split' is not defined

- **Cause:** You forgot to import train_test_split from sklearn.model_selection.

- **Fix:** Add from sklearn.model_selection import train_test_split at the beginning of your script.

Python

from sklearn.model_selection import train_test_split # Fix

2. NameError: name 'LinearRegressor' is not defined

- **Cause:** This is a common typo. The correct class name for linear regression in scikit-learn is LinearRegression.
- **Fix:** Correct the spelling.

Python

model = LinearRegression() # Fix

3. NameError: name 'plt' is not defined

- **Cause:** You're trying to use plt (the conventional alias for matplotlib.pyplot) without importing it.
- **Fix:** Import matplotlib.pyplot.

Python

import matplotlib.pyplot as plt # Fix

4. NameError: name 'data' is not defined (If data = pd.read_csv('data.csv') was commented out or not run)

- **Cause:** You're trying to use the data variable before it has been assigned the result of reading the CSV file.
- **Fix:** Ensure the line data = pd.read_csv('data.csv') is present and executed before any operations on data.

General Troubleshooting Steps for NameError:

1. **Read the Error Message Carefully:** Python's NameError messages are usually very clear, telling you exactly which name is not defined.
2. **Check for Typos:** Double-check the spelling of the variable or function name.
3. **Verify Imports:** Ensure all necessary modules, functions, and classes are correctly imported at the top of your script.

4. **Check Variable Definition:** Make sure the variable you're trying to use has been assigned a value before its use.
5. **Understand Scope:** If you're working with functions or classes, be mindful of where variables are defined and accessible. Global variables are accessible everywhere, but local variables are only within their defined scope.

Run Time Error

A "runtime error" in Jupyter is a general term that means something went wrong while your code was executing. It's like your program hit a snag and couldn't continue. Unlike syntax errors (which prevent your code from even starting due to incorrect grammar), runtime errors occur *during* the execution.

To help you troubleshoot, here's a breakdown of common causes and how to approach them:

Common Causes of Runtime Errors in Jupyter:

1. **NameError:**
    - **Meaning:** You're trying to use a variable or function that hasn't been defined or imported.
    - **Example:** print(my_variable) when my_variable hasn't been assigned a value.
    - **Jupyter Specifics:** Sometimes you might run cells out of order, or restart the kernel without re-running all necessary initialization cells.
2. **TypeError:**
    - **Meaning:** An operation is being performed on an object of an inappropriate type.
    - **Example:** "hello" + 5 (trying to add a string and an integer).
    - **Jupyter Specifics:** Can happen if data types change unexpectedly after a previous cell execution or if you're working with dataframes and assume a column's type without verifying.
3. **IndexError / KeyError:**
    - **Meaning:** You're trying to access an index in a list/tuple or a key in a dictionary that doesn't exist.
    - **Example:** my_list = [1, 2, 3]; print(my_list[3]) (list has only indices 0, 1, 2).

- **Jupyter Specifics:** Often seen when iterating or processing data, and your assumptions about the size or content of a data structure are incorrect.

4. **ValueError:**
    - **Meaning:** A function receives an argument of the correct type, but an inappropriate value.
    - **Example:** int("hello") (you can't convert the string "hello" to an integer).
    - **Jupyter Specifics:** Common when parsing user input, reading data from files, or converting between data types.
5. **ZeroDivisionError:**
    - **Meaning:** Attempting to divide a number by zero.
    - **Example:** 10 / 0
6. **FileNotFoundError:**
    - **Meaning:** Your code is trying to open a file that doesn't exist at the specified path.
    - **Jupyter Specifics:** Ensure your file paths are correct relative to your Jupyter notebook's location. Using os.getcwd() can help verify your current working directory.
7. **MemoryError:**
    - **Meaning:** Your program has run out of available memory, often due to trying to load very large datasets or creating excessively large data structures.
    - **Jupyter Specifics:** More common when dealing with big data or complex computations.
8. **RecursionError:**
    - **Meaning:** A function calls itself too many times, exceeding Python's recursion depth limit.
    - **Jupyter Specifics:** Indicates an issue with recursive function design.

How to Debug a Runtime Error in Jupyter:

1. **Read the Traceback:** This is the most crucial step! When an error occurs, Jupyter will display a "traceback" (also known as a stack trace). It tells you:

- **The type of error:** (e.g., NameError, TypeError).
- **The error message:** A brief explanation of what went wrong.
- **The file/cell and line number where the error occurred:** This points directly to the problematic line of code. Work your way *up* the traceback from the error line to see where the function calls originated.

Example Traceback Snippet:

NameError Traceback (most recent call last)

<ipython-input-1-abcdef123456> in <module>

----> 1 print(undefined_variable)

NameError: name 'undefined_variable' is not defined

2. **Locate the Error:** Go to the specific cell and line number indicated in the traceback.
3. **Inspect Variables:**
   - Use print() statements strategically before the error line to check the values and types of variables that are involved in the operation.
   - You can also simply type the variable name in a new cell and run it to see its current value and type.
4. **Step-by-Step Execution (or Mental Walkthrough):**
   - Break down the problematic line or block of code.
   - Imagine yourself as the computer executing each step. What are the values of the variables at each point? Does this match your expectation?
5. **Reproduce the Error with Minimal Code:** If the error is complex, try to isolate the problematic part of your code into a smaller, simpler example. This can help pinpoint the exact cause.