**Problem Statement:**

Developers often need to identify potential anomalies or unusual behavior in code to ensure its quality, maintainability, and reliability. However, manually analyzing the code to detect such anomalies can be a time-consuming and error-prone task, especially in large codebases.

The goal of this project is to create a code visualization tool that can help developers identify potential anomalies in code by highlighting patterns of unusual behavior or code changes. The tool should be able to analyze the code and identify specific areas that need attention, such as code that has changed significantly or code that has a higher-than-usual number of bugs.

**Datasets**

1. We trained the machine using data set we got from Kaggle. <https://www.kaggle.com/datasets/veeralakrishna/python-code-data>
2. We have taken 50 source codes from the kaggle for training the machine

**Methodologies**

To create a code visualization tool that can help developers identify potential anomalies in code by highlighting patterns of unusual behavior or code changes, the following methodology can be used:

1. Determine the scope and requirements of the tool: Define the programming languages and environments the tool will support, the types of anomalies the tool will identify, the level of detail required for the visualization, and the intended audience for the tool.
2. Identify the data sources: Determine the sources of data required for the visualization, such as code repositories, issue tracking systems, and continuous integration tools. Identify the format and structure of the data, as well as any required preprocessing steps.
3. Design the visualization: Develop a design for the visualization that effectively communicates the relevant information to the user. Consider the layout, color scheme, and interactive features of the visualization.
4. Choose the visualization library: Select a suitable visualization library that can effectively represent the data and meet the requirements of the design. Consider the available features, performance, and ease of use of the library.
5. Implement the visualization: Implement the visualization using the chosen library, incorporating the data sources and preprocessing steps as required. Ensure that the visualization is scalable and can handle large amounts of data.
6. Test and validate the tool: Test the tool to ensure that it accurately identifies anomalies and effectively communicates the information to the user. Validate the tool with a group of representative users to obtain feedback and identify any areas for improvement.
7. Iterate and refine the tool: Use the feedback obtained from users to refine the tool and improve its performance and usability. Iterate through the design and implementation process as required to achieve the desired results.
8. Document and maintain the tool: Document the tool's functionality, limitations, and usage instructions. Maintain the tool by updating it to support new programming languages and environments, addressing any bugs or issues that arise, and incorporating new features as required.

**Code**

import numpy as np

import tensorflow as tf

from sklearn.ensemble import IsolationForest

# Define the vectorize\_program function

def vectorize\_program(program):

# Tokenize and vectorize the program

tokenizer = tf.keras.preprocessing.text.Tokenizer()

tokenizer.fit\_on\_texts([program])

sequence = tokenizer.texts\_to\_sequences([program])[0]

padded\_sequence = tf.keras.preprocessing.sequence.pad\_sequences([sequence])

vectorized\_program = tf.one\_hot(padded\_sequence, depth=len(tokenizer.word\_index)+1)

# Pad the program vector with zeros along the second dimension

max\_len = max(len(sequence), 1)

padded\_program = tf.keras.preprocessing.sequence.pad\_sequences(vectorized\_program, maxlen=max\_len, padding='post')

vectorized\_program\_2d = np.reshape(padded\_program, (-1, padded\_program.shape[-1]))

return vectorized\_program\_2d

# Define the list of filenames

filenames = ['code1.py', 'code2.py', 'code3.py', 'code4.py', 'code5.py', 'code6.py', 'code7.py', 'code8.py', 'code9.py', 'code10.py', 'code11.py', 'code12.py', 'code13.py', 'code14.py', 'code15.py', 'code16.py', 'code17.py', 'code18.py', 'code19.py', 'code20.py','code21.py', 'code22.py', 'code23.py', 'code24.py', 'code25.py', 'code26.py', 'code27.py', 'code28.py','code29.py', 'code30.py', 'code31.py', 'code32.py', 'code33.py', 'code34.py', 'code35.py', 'code36.py', 'code37.py', 'code38.py', 'code39.py', 'code40.py','code42.py','code43.py','code44.py']

# Loop over the filenames

for filename in filenames:

# Read the program from the file

with open(filename, 'r') as file:

program = file.read()

# Vectorize the program

vectorized\_program = vectorize\_program(program)

# Split the data into training and testing sets

train\_size = int(0.8 \* vectorized\_program.shape[0])

train\_data = vectorized\_program[:train\_size]

test\_data = vectorized\_program[train\_size:]

# Fit the IsolationForest model on the training set

model = IsolationForest(n\_estimators=100, contamination=0.01, random\_state=42)

model.fit(train\_data)

# Use the trained model to predict anomalies in the test set

predictions = model.predict(test\_data)

# Print the number of anomalies detected by the model

print(f"Number of anomalies detected in {filename}: {sum(predictions == -1)}")

**Isolation Forests Anamoly Detection**

Isolation Forest similar to Random Forests, are build based on decision trees. And since there are no pre-defined labels here, it is an unsupervised model.

Isolation Forests were built based on the fact that anomalies are the data points that are “few and different”.

In an Isolation Forest, randomly sub-sampled data is processed in a tree structure based on randomly selected features. The samples that travel deeper into the tree are less likely to be anomalies as they required more cuts to isolate them. Similarly, the samples which end up in shorter branches indicate anomalies as it was easier for the tree to separate them from other observations.

**Let us look at the complete algorithm step by step:**

When given a dataset, a random sub-sample of the data is selected and assigned to a binary tree.

Branching of the tree starts by selecting a random feature (from the set of all N features) first. And then branching is done on a random threshold ( any value in the range of minimum and maximum values of the selected feature).

If the value of a data point is less than the selected threshold, it goes to the left branch else to the right. And thus a node is split into left and right branches.

This process from step 2 is continued recursively till each data point is completely isolated or till max depth(if defined) is reached.

The above steps are repeated to construct random binary trees.

After an ensemble of iTrees(Isolation Forest) is created, model training is complete. During scoring, a data point is traversed through all the trees which were trained earlier. Now, an ‘anomaly score’ is assigned to each of the data points based on the depth of the tree required to arrive at that point. This score is an aggregation of the depth obtained from each of the iTrees. An anomaly score of -1 is assigned to anomalies and 1 to normal points based on the contamination (percentage of anomalies present in the data) parameter provided

**Python libraries used**

**1.Pandas:-**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.

Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**Key Features of Pandas**

Fast and efficient Data Frame object with default and customized indexing.

Tools for loading data into in-memory data objects from different file formats.

Data alignment and integrated handling of missing data.

Reshaping and pivoting of date sets.

Label-based slicing, indexing and subsetting of large data sets.

Columns from a data structure can be deleted or inserted.

Group by data for aggregation and transformations.

High performance merging and joining of data.

Time Series functionality.

**2.numpy**

NumPy (Numerical Python) is an open source Python library that’s used in almost every field of science and engineering. It’s the universal standard for working with numerical data in Python, and it’s at the core of the scientific Python and PyData ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

The NumPy library contains multidimensional array and matrix data structures (you’ll find more information about this in later sections). It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

**3.scikit learn**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Origin of Scikit-Learn

It was originally called scikits.learn and was initially developed by David Cournapeau as a Google summer of code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation), took this project at another level and made the first public release (v0.1 beta) on 1st Feb. 2010.

**4.matplotlib**

Matplotlib is a Python library for data visualization that provides a wide range of tools for creating high-quality, publication-ready plots and figures. It is one of the most popular data visualization libraries in Python, and is widely used in the scientific community for creating visualizations of data from various fields such as physics, biology, and finance.

Matplotlib is designed to be easy to use and customizable, with a simple and intuitive interface that allows users to create a variety of plot types, including line plots, scatter plots, bar charts, histograms, and many more. It is built on top of the NumPy and SciPy libraries, and integrates seamlessly with other data analysis and visualization tools in the Python ecosystem.

Matplotlib can be used to create both static and interactive plots, and provides a range of options for customizing the appearance of plots, including colors, labels, markers, and fonts. It also includes support for a wide range of file formats, including PNG, PDF, SVG, and more, making it easy to export high-quality plots for use in presentations, papers, and other documents.

**5.tensor flow**

TensorFlow is an open-source software library developed by Google that is used for building and training machine learning models. It is one of the most popular and widely used machine learning frameworks, and is known for its flexibility, scalability, and ease of use.

TensorFlow is designed to work with both CPUs and GPUs, and allows developers to create and train a wide range of machine learning models, including deep neural networks, convolutional neural networks, recurrent neural networks, and more. It provides a high-level API that makes it easy to build and train models, and also includes a lower-level API that allows for more fine-grained control over the model building process.

In addition to its core functionality for building and training models, TensorFlow also includes a range of tools and libraries for data preprocessing, visualization, and deployment. It also integrates with a wide range of other libraries and tools in the Python ecosystem, making it easy to use in conjunction with other data analysis and machine learning tools.

Overall, TensorFlow is a powerful and versatile machine learning framework that is widely used by developers and researchers around the world for a variety of applications, from image and speech recognition to natural language processing and more.

**6.keras**

Keras is an open-source software library written in Python that provides a high-level API for building and training deep learning models. It is built on top of lower-level deep learning frameworks such as TensorFlow and Theano, and provides a simple and intuitive interface that makes it easy to build complex neural networks.

Keras is designed to be user-friendly and easy to use, with a focus on enabling rapid experimentation and prototyping of deep learning models. It provides a wide range of building blocks for constructing neural networks, including layers, activations, loss functions, and optimizers. These building blocks can be combined and configured in a variety of ways to create different types of neural networks, including feedforward networks, convolutional networks, and recurrent networks.

One of the key features of Keras is its focus on modularity and extensibility. It allows users to easily define their own custom layers and loss functions, and provides a range of tools for visualizing and debugging neural networks. It also includes support for a wide range of data formats and preprocessing techniques, making it easy to work with different types of data.

Overall, Keras is a powerful and versatile deep learning library that is widely used by researchers and developers around the world for a variety of applications, including image and speech recognition, natural language processing, and more.

**APPROACHES**

**1. tokenisation**

In natural language processing, tokenization is the process of breaking up text into smaller units, known as tokens. These tokens can be words, phrases, sentences, or even individual characters, depending on the specific task and context.

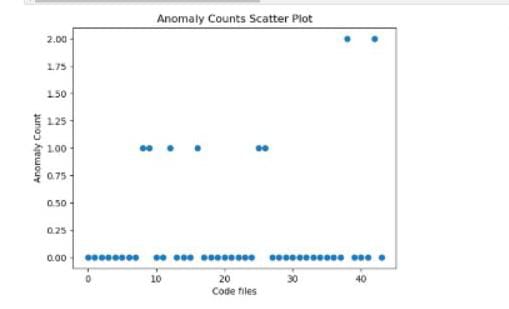
In Python, there are several libraries and tools available for tokenization, including the built-in string methods, regular expressions, and external libraries such as NLTK (Natural Language Toolkit) and spaCy.

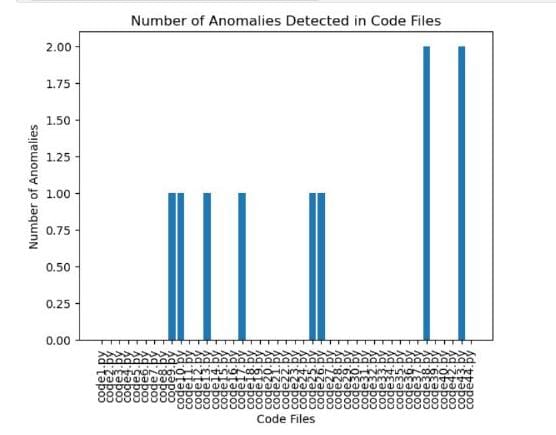
**2. vectorizarition**

Vectorization is the process of converting non-numeric data into numeric form, so that it can be processed by machine learning algorithms. In the context of natural language processing, vectorization is used to represent text data as numerical vectors, which can then be used as input for various machine learning models.

In Python, there are several libraries and tools available for vectorization, including NumPy, SciPy, and scikit-learn.

**Results and Visualisation :-**

****

****