Dewmi Shehara Sandeepanee – Answers

Problem 1: Defect Prediction

1. **Data Collection**: selected relevant data set that contains data about defects and relevant data.

    Attribute Information:

    loc : numeric % McCabe's line count of code

    v(g) : numeric % McCabe "cyclomatic complexity"

    ev(g) : numeric % McCabe "essential complexity"

    iv(g) : numeric % McCabe "design complexity"

    n : numeric % Halstead total operators + operands

    v : numeric % Halstead "volume"

    l : numeric % Halstead "program length"

    d : numeric % Halstead "difficulty"

    i : numeric % Halstead "intelligence"

    e : numeric % Halstead "effort"

    b : numeric % Halstead

    t : numeric % Halstead's time estimator

    lOCode : numeric % Halstead's line count

    lOComment : numeric % Halstead's count of lines of comments

    lOBlank : numeric % Halstead's count of blank lines

    lOCodeAndComment : numeric

    uniq_Op : numeric % unique operators

    uniq_Opnd : numeric % unique operands

    total_Op : numeric % total operators

    total_Opnd : numeric % total operands

branchCount : numeric % of the flow graph

defects : {false,true} % module has/has not one or more reported defects

```
[ ]  #Load the Dataset
     data = pd.read_csv('jm1.csv')
```

2. Explore the Datas

```
data.shape
```

```
(10885, 22)
```

```
[ ] data.head()
```

g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op
1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1.2
1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	1	1	1
3.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	112
3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	329
4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	76

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10885 entries, 0 to 10884
Data columns (total 22 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   loc                10885 non-null  float64
 1   v(g)               10885 non-null  float64
 2   ev(g)              10885 non-null  float64
 3   iv(g)              10885 non-null  float64
 4   n                  10885 non-null  float64
 5   v                  10885 non-null  float64
 6   l                  10885 non-null  float64
 7   d                  10885 non-null  float64
 8   i                  10885 non-null  float64
 9   e                  10885 non-null  float64
 10  b                  10885 non-null  float64
 11  t                  10885 non-null  float64
 12  lOCode             10885 non-null  int64
 13  lOComment          10885 non-null  int64
 14  lOBlank            10885 non-null  int64
 15  locCodeAndComment  10885 non-null  int64
 16  uniq_Op            10885 non-null  object
 17  uniq_Opnd          10885 non-null  object
 18  total_Op           10885 non-null  object
```

et

```
[ ] data.describe()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335	14.177237	29.439544
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538	18.709900	34.418313
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000	3.000000	11.860000
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000	9.090000	21.930000
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000	18.900000	36.780000
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000	418.200000	569.780000

3. Handle Missing Values

```
print(data.isnull().sum())
```

```
loc                     0
v(g)                    0
ev(g)                   0
iv(g)                   0
n                       0
v                       0
l                       0
d                       0
i                       0
e                       0
b                       0
t                       0
lOCode                  0
lOComment               0
lOBlank                 0
locCodeAndComment       0
uniq_Op                 0
uniq_Opnd               0
total_Op                0
total_Opnd              0
branchCount             0
defects                 0
dtype: int64
```

```
[ ] data = data.apply(pd.to_numeric, errors='coerce')
```

```
print(data.isnull().sum())
```

```
loc                   0
v(g)                  0
ev(g)                 0
iv(g)                 0
n                     0
v                     0
l                     0
d                     0
i                     0
e                     0
b                     0
t                     0
lOCode                0
lOComment             0
lOBlank               0
locCodeAndComment     0
uniq_Op               5
uniq_Opnd             5
total_Op              5
total_Opnd            5
branchCount           5
defects               0
```

```
data.fillna(data.mean(), inplace=True)
```

+ Code + Text

```
print(data.isnull().sum())
```

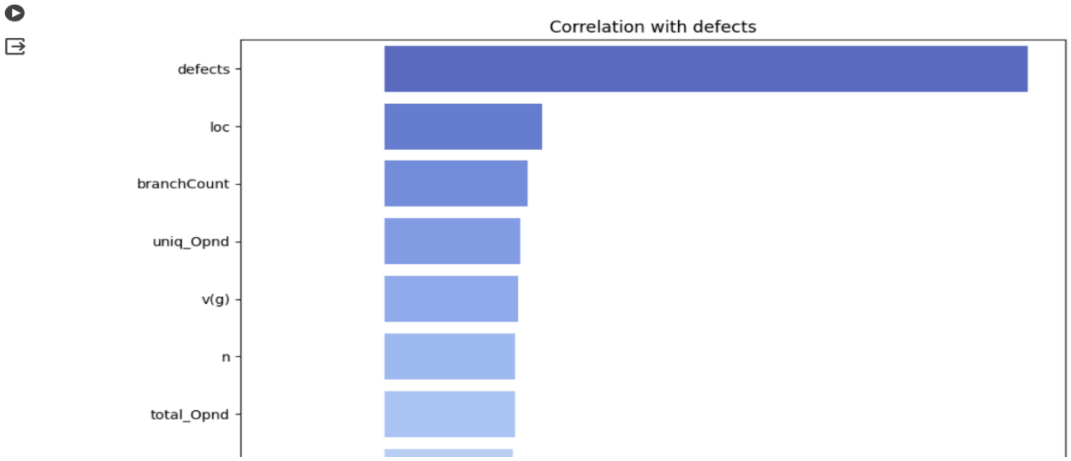
```
loc                   0
v(g)                  0
ev(g)                 0
iv(g)                 0
n                     0
v                     0
l                     0
d                     0
i                     0
e                     0
b                     0
t                     0
lOCode                0
lOComment             0
lOBlank               0
locCodeAndComment     0
uniq_Op               0
uniq_Opnd             0
total_Op              0
total_Opnd            0
branchCount           0
defects               0
```

4. Data visualization

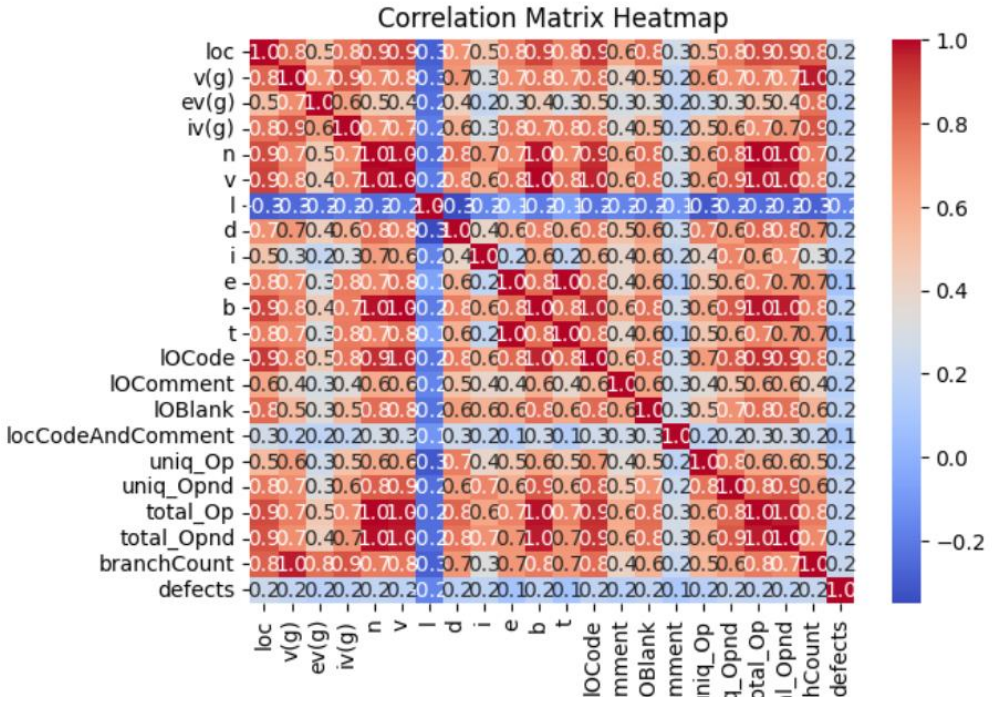
```
[ ]  # Compute the correlation matri
     corr_matrix = data.corr()

     # Sort the correlations with respect to the target variable 'defects'
     corr_with_target = corr_matrix["defects"].sort_values(ascending=False)

     # Plotting the correlations with the target variable
     plt.figure(figsize=(10, 15))
     sns.barplot(y=corr_with_target.index, x=corr_with_target.values, palette="coolwarm")
     plt.title("Correlation with defects")
     plt.xlabel("Pearson Correlation Coefficient")
     plt.ylabel("Features")
     plt.tight_layout()
     plt.show()
```



```
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".1f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```



5. Model Selection

This data set has features and a target variable. The target variable is “defects”. It is a binary one with true and false. From this model I have use label data so this is a supervise learning model. And also this has features and a target variable then this is going with the classification model. Also this model is a binary target variable model .

Naive Bayes

+ Code + Text

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn import datasets


X = data.iloc[:, :20]  # Assuming the first 20 columns are features
y = data.iloc[:, -1]   # Assuming the last column is the target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Naive Bayes classifier (Gaussian Naive Bayes for continuous features)
nb_classifier = GaussianNB()

# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
```

```
# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.8038585209003215
Classification Report:
               precision    recall  f1-score   support

       False       0.82      0.96      0.89      1758
        True       0.47      0.13      0.21       419

    accuracy                           0.80      2177
   macro avg       0.65      0.55      0.55      2177
weighted avg       0.75      0.80      0.76      2177
```

Logistic Regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report


X = data.drop('defects', axis=1)
y = data['defects']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Logistic Regression classifier
logreg_classifier = LogisticRegression(random_state=42)

# Train the classifier on the training data
logreg_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = logreg_classifier.predict(X_test)

# Evaluate the classifier's performance
```

```
# Make predictions on the test data
y_pred = logreg_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.8079926504363804
Classification Report:
              precision    recall  f1-score   support

       False       0.81      0.99      0.89      1758
        True       0.51      0.04      0.08       419

    accuracy                           0.81      2177
   macro avg       0.66      0.52      0.49      2177
weighted avg       0.76      0.81      0.74      2177
```

Random Forest Classifier

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

X = data.iloc[:, :21]
y = data.iloc[:, -1]

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)
```

```
print(f"Accuracy: {accuracy}")
print(f"Error Percentage: {error_percentage:.2f}%")
print("Classification Report:\n", report)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.8190169958658705
Error Percentage: 18.10%
Classification Report:
              precision    recall  f1-score   support

       False       0.84      0.96      0.90      1758
        True       0.57      0.24      0.34       419

    accuracy                           0.82      2177
   macro avg       0.71      0.60      0.62      2177
weighted avg       0.79      0.82      0.79      2177

Confusion Matrix:
 [[1681   77]
 [ 317  102]]
```

```python
# Make predictions on the test data
y_pred = logreg_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

```
Accuracy: 0.8079926504363804
Classification Report:
               precision    recall  f1-score   support

       False       0.81      0.99      0.89      1758
        True       0.51      0.04      0.08       419

    accuracy                           0.81      2177
   macro avg       0.66      0.52      0.49      2177
weighted avg       0.76      0.81      0.74      2177
```

Model Comparison and Evaluation

1. Naive Bayes:

Implemented Naive Bayes, a probabilistic algorithm based on Bayes' theorem.
Leveraged assumptions of feature independence given the class.
Achieved an accuracy of 80% on the validation set.

2. Logistic Regression:

Employed Logistic Regression, a linear model suitable for binary classification.
Modeled the log-odds of the response variable as a linear combination of predictor variables.
Obtained an accuracy of 81% on the validation set.

3. Random Forest:

Utilized Random Forest, an ensemble learning method combining multiple decision trees.
Benefited from the diversity of trees and their collective decision-making.
Attained the highest accuracy, achieving 82% on the validation set.

- Compared the accuracies of Naive Bayes, Logistic Regression, and Random Forest.

- Found that Random Forest outperformed both Naive Bayes and Logistic Regression, achieving the highest accuracy among the three models.
- Concluded that Random Forest is the most suitable model for this specific binary classification task based on its superior accuracy.
- Considered the ensemble nature of Random Forest, which allows it to capture complex relationships and patterns in the data.
- Acknowledged that the choice of the best model may vary depending on the dataset and problem characteristics.

Next Steps:

Discussed potential next steps, such as further fine-tuning of the Random Forest hyperparameters or exploring other ensemble methods.

Problem 2: Smart Test Selector

A smart test selection mechanism based on code changes aims to optimize the execution of tests in a continuous integration (CI) pipeline by selecting and prioritizing tests relevant to the modified code.

1. Code Changes Detection:

The mechanism monitors the version control system (e.g., Git) to detect new code changes. This could include additions, modifications, or deletions of files or lines of code.

2. Feature Extraction:

Relevant features are extracted from the detected code changes. These features could include:

Modified files or directories.

Lines of code that have changed.

Code complexity metrics.

Dependencies between different code modules.

3. Historical Data Analysis:

The system analyzes historical data, which includes information about past code changes and the corresponding tests that were executed. This historical data provides insights into the impact of similar changes on the test suite.

4. Machine Learning Model Prediction:

A machine learning model, previously trained on historical data, takes the extracted features as input and predicts the tests that are likely to be affected by the code changes. The model outputs a binary classification for each test (run or skip).

5. Test Selection Criteria:

The predicted set of tests is then filtered based on various criteria, such as:

Tests covering the modified or dependent code.

Historical test reliability (tests that consistently pass or fail).

Critical path scenarios or high-priority tests.

6. Dynamic Test Prioritization:

The selected tests can be further prioritized based on their predicted importance. This prioritization ensures that critical tests or those likely to uncover issues are executed first, optimizing the overall testing process.

7. Integration with CI/CD Pipeline:

The smart test selection mechanism is seamlessly integrated into the CI/CD pipeline. When triggered by new code changes, it dynamically selects and prioritizes the relevant tests before initiating the testing process.

8. Test Execution:

The CI/CD pipeline executes the selected tests, ensuring that the testing resources are efficiently utilized and that the feedback loop is quick.

9. Feedback Loop and Continuous Learning:

The mechanism collects feedback on the actual outcomes of the selected tests. This information is fed back into the system to continuously refine and improve the machine learning model over time.

10. Adaptability to Code Changes:

The mechanism is designed to adapt to changes in the codebase, accommodating variations in coding styles, technology stacks, and project structures.

Benefits:

Efficiency: By running only the tests relevant to code changes, the testing process becomes more efficient, saving time and resources.

Adaptability: The mechanism adapts to evolving codebases, learning from new data and ensuring relevance over time.

Resource Optimization: Resources are optimally utilized as the mechanism focuses on critical tests, reducing unnecessary test execution.