

# TEST PLAN DESIGN

**Objective:** The objective of this test plan is to ensure the reliability and functionality of the code developed by another group through comprehensive testing. This report outlines the process of reviewing the provided code, identifying test scenarios, defining test objectives, creating test cases, planning test execution, and documenting the test plan.

## Understanding the Code

The provided code consists of main classifier functions: B\_Classifier, C\_Classifier, and O\_Classifier. The b classifier classifies the data based on a threshold value to b0 and bn. The C classifier classifies the data based on a threshold value to cw and cb. The O classifier function should classify the sample data to three groups o0, o1 and ox.

## Identifying Test Scenarios

For all three types of classifiers, we tested the functions using the dataset given to our group. Our dataset for b classifier contained 2000 records. Since the number of records wasn't too many, we didn't feel the need to sample it. For C classifier we sampled 30% data from each of the 6 files. For o classifier we did not sample the data.

**Defining Test Objectives:** The test objective is that each classifier of all 10 groups should perform with an accuracy greater than 80 percent. We intend to share feedback with the groups according to the accuracies we obtained for their respective functions.

## Test Cases

### B Classifier

#### Group1

Group 1's B classifier has an accuracy of 99.84% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took 2 seconds to execute.

The screenshot shows a Jupyter Notebook workspace titled "Untitled (Workspace)". The code cell contains the following Python script:

```
value=0.6
class_label = classify_b(value)
print(class_label)

# Test the classify_b function and calculate accuracy
test_values = data['values'].tolist()
true_classes = data['class'].tolist()
predicted_classes = [classify_b(value) for value in test_values]
accuracy = sum([1 for true_class, predicted_class in zip(true_classes, predicted_classes) if true_class == predicted_class]) / len(true_classes)

# Print the accuracy
print("Accuracy:", accuracy, "%")
```

The output cell shows the result:

```
Accuracy: 99.84992496248124 %
```

The Jupyter interface includes a sidebar with file explorer, outline, timeline, and zip explorer tabs. The bottom status bar shows system information like battery level, signal strength, and date/time.

The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The title bar reads "Untitled (Workspace)". The left sidebar displays the "EXPLORER" view, showing a folder structure under "UNTITLED (WORKSPACE)" and "C:\Users\apala\Downloads". The main editor area contains Python code:

```
# Print the percentage of incorrect predictions
print("Percentage of Incorrect Predictions:", incorrect_percentage, "%")
```

Output window:

```
[11] ✓ 0.0s
... Incorrect Predictions:
      Values True_Class Predicted_Class
94    0.509716      bn        bo
472   0.505442      bn        bo
1875  0.467431      bo        bn
```

Terminal output:

```
Percentage of Incorrect Predictions: 0.1500750375187594 %
```

Bottom status bar:

```
Spaces: 4 CRLF Cell 15 of 15
17°C Cloudy Search 2:09 AM 4/11/2024 ENG US
```

## Group 2

Group 2's B classifier has an accuracy of 99.9% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

The screenshot shows a Windows desktop environment with the Visual Studio Code (VS Code) application open. The title bar reads "Untitled (Workspace)". The left sidebar displays the "EXPLORER" view, showing a folder structure under "UNTITLED (WORKSPACE)" and "C:\Users\apala\Downloads". The main editor area contains Python code:

```
return original[0]
```

Output window:

```
[12] ✓ 0.0s
... classify_b(0.756699)
[13] ✓ 0.0s
... 'bo'
```

Terminal output:

```
test_values = df['Values'].tolist()
true_classes = df['Class'].tolist()
predicted_classes = [classify_b(value) for value in test_values]
accuracy = sum([1 for true_class, predicted_class in zip(true_classes, predicted_classes) if true_class == predicted_class]) / len(true_classes)
```

Output window:

```
# Print the accuracy
print("Accuracy:", accuracy, "%")
```

Output window:

```
[13] ✓ 2.8s
... Accuracy: 99.9 %
```

Bottom status bar:

```
Spaces: 4 CRLF Cell 15 of 15
17°C Cloudy Search 2:03 AM 4/11/2024 ENG US
```

The screenshot shows a Jupyter Notebook workspace in VS Code. The left sidebar displays a file tree with various files and folders. The main area shows a code cell with Python code and its output. The output includes a table titled "Incorrect Predictions" and a percentage of incorrect predictions.

```
# Print the percentage of incorrect predictions
print("Percentage of Incorrect Predictions:", incorrect_percentage, "%")
```

... Incorrect Predictions:

	Values	True_Class	Predicted_Class
95	0.509716	bn	bo
1876	0.467431	bo	bn

Percentage of Incorrect Predictions: 0.1 %

### Group 3

Group 3's B classifier has an accuracy of 100% for our data. We believe with perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

The screenshot shows a Jupyter Notebook cell with Python code. The code applies a classification function to a dataset and calculates the accuracy. The output shows the accuracy is 1.00.

```
[7] 1 # Applying the classify_b function to all values in the dataset to get predictions
2 predictions = data['value'].apply(classify_b)
3
4 # Calculating the accuracy as the proportion of correct predictions
5 accuracy = (predictions == data['class']).mean()
6 print(f'Accuracy: {accuracy:.2f}')
```

Accuracy: 1.00

0s

```
1 misclassified_rows = data[predictions != data['class']]
2 print("Misclassified Rows:")
3 print(misclassified_rows)
4
```

Misclassified Rows:

	class	value
95	bn	0.509716
473	bn	0.505442
1876	bo	0.467431

## Group 4

Group 4's B classifier has an accuracy of 80.6% for our data. We believe this accuracy can be improvised. The just took a few seconds to execute. We gave feedback accordingly to the team.

```
[2] 1 from sklearn.metrics import silhouette_score
2
3 # Calculate silhouette score
4 silhouette_avg = silhouette_score(X, kmeans.labels_)
5 print(f"Silhouette Score: {silhouette_avg}")

Silhouette Score: 0.806114888337743

[play] 1 misclassified_rows = data[data['Class'] != data['Cluster']].map({0: 'bn', 1: 'bo'})
2 print("Misclassified Rows:")
3 print(misclassified_rows)
4

Misclassified Rows:
   Class    Value  Cluster
0      1  1.078448      1
1      1  0.785672      1
2      0  0.237356      0
3      1  0.799500      1
4      0  0.324320      0
...
1994    ...     ...
1994      0  0.149535      0
1995    1  0.794233      1
1996    1  0.699805      1
1997    0  0.237857      0
1998    1  0.849937      1

[1999 rows x 3 columns]
```

## Group 5

Group 5's B classifier has an accuracy of 99.7% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

```
[ ] predictions = data['val'].apply(lambda x: classify_b(x, bn_range, bo_range))
correct_predictions = (predictions == data['bx']).sum()
total_predictions = len(data)
accuracy = correct_predictions / total_predictions

print("Accuracy of the classifier for class 'b':", accuracy)

Accuracy of the classifier for class 'b': 0.9974987493746873

▶ # Compare the predicted classes with the true classes from the dataset
incorrectly_classified = data[data['bx'] != predictions]

print("Incorrectly classified classes:")
print(incorrectly_classified)

→ Incorrectly classified classes:
      bx      val
94    bn  0.509716
472   bn  0.505442
973   bn  0.001092
1412  bn  0.001276
1875  bo  0.467431
```

## Group 6

Group 6's B classifier has an accuracy of 99.65% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

```
def calculate_accuracy(true_labels, predicted_labels):
    correct_predictions = sum(true_labels == predicted_labels)
    total_predictions = len(true_labels)
    accuracy = correct_predictions / total_predictions * 100
    return accuracy

# Step 1: Apply the classification function to the entire dataset
predicted_labels = data['value'].apply(classify_b)

# Step 2: If you have true labels available, compare with the predicted labels
# Assuming 'true_labels' contains the true labels for the entire dataset
true_labels = data['bn']

# Step 3: Calculate accuracy
accuracy = calculate_accuracy(true_labels, predicted_labels)
print(f"Accuracy: {accuracy:.2f}%")

Accuracy: 99.65%
```

## Group 7

Group 7's B classifier has an accuracy of 99.85% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

```
[9] # Apply the classify_b function to the 'Values' column of the DataFrame to get predictions
predictions = df['Values'].apply(classify_b)

# Compare the predicted labels with the actual labels from the DataFrame
accuracy = (predictions == df['Label']).mean()

print("Accuracy:", accuracy)
```

Accuracy: 0.9985

```
# Apply the classify_b function to the 'Values' column of the DataFrame to get predictions
predictions = df['Values'].apply(classify_b)

# Create a boolean mask to identify failure cases
failure_mask = predictions != df['Label']

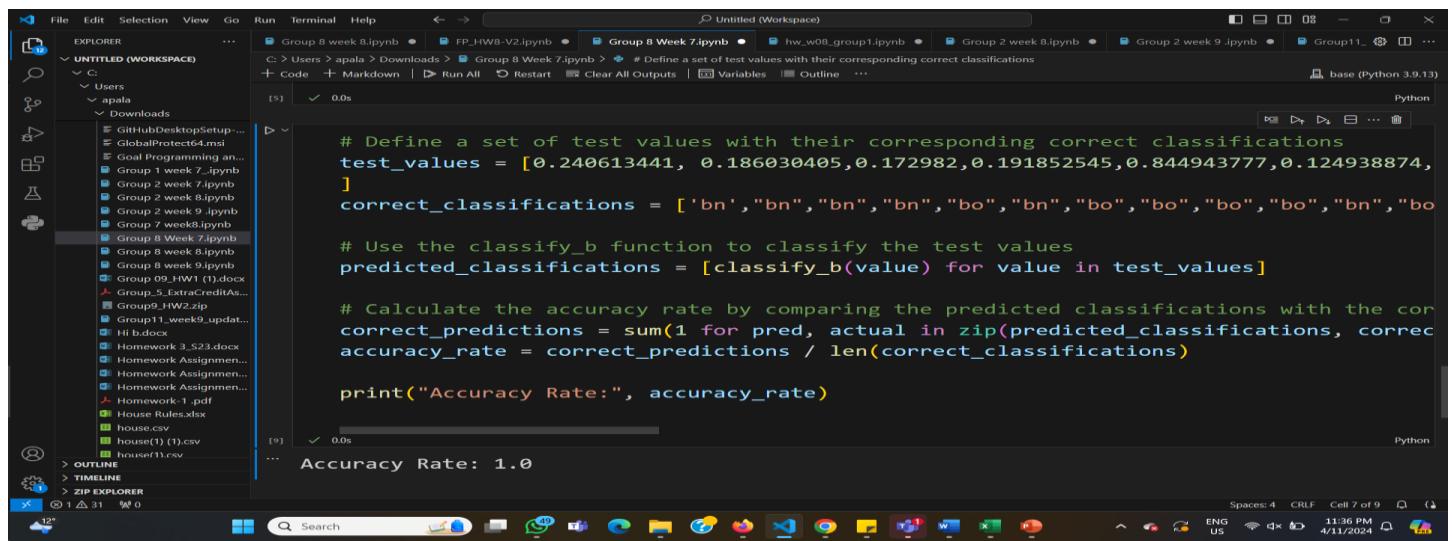
# Extract the failure cases
failure_cases = df[failure_mask]

# Print the failure cases
print("Failure Cases:")
print(failure_cases)
```

	Label	Values
95	bn	0.509716
473	bn	0.505442
1876	bo	0.467431

## Group 8

Group 8's B classifier has an accuracy of 100% for our data. We believe with perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team



```
# Define a set of test values with their corresponding correct classifications
test_values = [0.240613441, 0.186030405, 0.172982, 0.191852545, 0.844943777, 0.124938874,
]
correct_classifications = ['bn', 'bn', 'bn', 'bn', 'bo', 'bn', 'bo', 'bo', 'bo', 'bn', 'bo']

# Use the classify_b function to classify the test values
predicted_classifications = [classify_b(value) for value in test_values]

# Calculate the accuracy rate by comparing the predicted classifications with the correct ones
correct_predictions = sum(1 for pred, actual in zip(predicted_classifications, correct_classifications) if pred == actual)
accuracy_rate = correct_predictions / len(correct_classifications)

print("Accuracy Rate:", accuracy_rate)
```

## Group 9

Group 9's B classifier has an accuracy of 99.85% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

```

data = pd.read_csv(r"C:\Users\shagu\Downloads\School\Spr24\6390\Week 10\G11\dataset_b11.csv", header=None, names=header)

# Now you can use the 'data' DataFrame with the appropriate headers

# Initialize variables to store correct predictions
correct_predictions = 0
total_predictions = len(data)

# Iterate through the dataset and make predictions for each value
for index, row in data.iterrows():
    true_class = row['Class']
    value = row['Features']
    predicted_class = classify_b(value)
    if predicted_class == true_class:
        correct_predictions += 1

# Calculate accuracy
accuracy = correct_predictions / total_predictions

print("Accuracy:", accuracy)

```

[8]

... Accuracy: 0.9985

The screenshot shows a Jupyter Notebook interface with three tabs at the top: 'Group\_9\_week\_8.ipynb', 'Group\_9\_week\_7\_v2.ipynb' (selected), and 'Group\_9\_week\_9.ipynb'. The code cell contains Python code for calculating accuracy and incorrect predictions. The output cell shows the accuracy as 0.9985 and the percentage of incorrect predictions as 0.0s.

```

df_results = pd.DataFrame({'Values': test_values,
                           'True_Class': true_classes,
                           'Predicted_Class': predicted_classes})

incorrect_predictions = df_results[df_results['True_Class'] != df_results['Predicted_Class']]

incorrect_percentage = (len(incorrect_predictions) / len(df_results)) * 100

print("Incorrect Predictions:")
print(incorrect_predictions)

print("Percentage of Incorrect Predictions:", incorrect_percentage, "%")

```

[9] ✓ 0.0s

... Incorrect Predictions:

	Values	True_Class	Predicted_Class
95	0.509716	bn	bo
473	0.505442	bn	bo
1876	0.467431	bo	bn

Percentage of Incorrect Predictions: 0.15 %

## Group 10

Group 10's B classifier has an accuracy of 99.85% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took a few seconds to execute. We gave good feedback to the team

Group\_9\_week\_7\_v2.ipynb ● Group 10 week 7.ipynb X Market\_Basket.ipynb (1) (1).ipynb ● Group 11 week 7.ipynb

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 10 > Group 10 week 7.ipynb > Group 10 - Week 07:

+ Code + Markdown | ▶ Run All ⏪ Restart ⏹ Clear All Outputs | 📄 Variables 📌 Outline ...

```
▷ 
# Calculate mean and standard deviation for 'bn' class
bn_data = labeled_dataset[labeled_dataset.iloc[:, 0] == 'bn'].iloc[:, 1]
bn_mean = bn_data.mean()
bn_std = bn_data.std()

# Apply the classifier to each value in the dataset
predicted_labels = labeled_dataset.iloc[:, 1].apply(lambda x: classify_b(x, bn_mean, bn_std))

# Compare predicted labels with actual labels
actual_labels = labeled_dataset.iloc[:, 0]
accuracy = (predicted_labels == actual_labels).mean()

print(f"Accuracy: {accuracy * 100:.2f}%")
```

[31]

... Accuracy: 99.85%

Group\_9\_week\_8.ipynb ● Group\_9\_week\_7\_v2.ipynb ● Group 10 week 7.ipynb ● Group\_9\_week\_9.ipynb ●

up 10 - Week 07: FP-HW > c) Is there a way to distinguish between the values belonging to each class? Write a function called classify\_b that ta

```
+ Code + Markdown | ▶ Run All ⏪ Restart ⏹ Clear All Outputs | 📄 Variables 📌 Outline ...
▷ 
# Create a DataFrame with test values, true classes, and predicted classes
df_results = pd.DataFrame({'Values': labeled_dataset.iloc[:, 1], # Test values
                            'True_Class': actual_labels, # True classes
                            'Predicted_Class': predicted_labels}) # Predicted classes

# Filter out rows where the true class differs from the predicted class
incorrect_predictions = df_results[df_results['True_Class'] != df_results['Predicted_Class']]

# Calculate the percentage of incorrect predictions
incorrect_percentage = (len(incorrect_predictions) / len(df_results)) * 100

# Print the DataFrame of incorrect predictions
print("Incorrect Predictions:")
print(incorrect_predictions)

# Print the percentage of incorrect predictions
print("Percentage of Incorrect Predictions:", incorrect_percentage, "%")
```

[16] ✓ 0.0s

```
... Incorrect Predictions:
    Values True_Class Predicted_Class
94    0.509716      bn        bo
472    0.505442      bn        bo
1875   0.467431      bo        bn
Percentage of Incorrect Predictions: 0.1500750375187594 %
```

## C Classifier

### Group1

File Edit Selection View Go Run Terminal Help ← → ⌘ Untitled (Workspace)

EXPLORER UNTITLED (WORKSPACE) C: Users apala Downloads

- Hi.b.docx
- Homework\_3\_S23.docx
- Homework Assignment...
- Homework Assignment...
- Homework-1.pdf
- House Rules.xlsx
- house.csv
- house(1) (1).csv
- How to interpret Inter...
- How to interpret Inter...
- hw\_w08\_group1.ipynb
- HW2\_23 (1).docx
- HW2\_23.docx
- HWA4.docx
- HW5\_S22 (1).docx
- HW5\_S22.docx
- I-20.pdf
- Individual Peer Evaluati...
- Integer Linear Program...
- Interpretation of regre...
- Interpretation of regre...
- IRI\_POS\_Tablespreads...
- Kimball\_The-Data-War...

OUTLINE TIMELINE ZIP EXPLORER

```
#accuracy of the logistic reg:
from sklearn.metrics import accuracy_score

# Predict labels for test data
y_pred = clf_1.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

... Accuracy: 0.9669117647058824

Spaces: 4 Cell 16 of 16

17°C Cloudy 2:15 AM 4/11/2024 ENG US

File Edit Selection View Go Run Terminal Help ← → ⌘ Untitled (Workspace)

EXPLORER UNTITLED (WORKSPACE) C: Users apala Downloads

- Hi.b.docx
- Homework\_3\_S23.docx
- Homework Assignment...
- Homework Assignment...
- Homework-1.pdf
- House Rules.xlsx
- house.csv
- house(1) (1).csv
- How to interpret Inter...
- How to interpret Inter...
- hw\_w08\_group1.ipynb
- HW2\_23 (1).docx
- HW2\_23.docx
- HWA4.docx
- HW5\_S22 (1).docx
- HW5\_S22.docx
- I-20.pdf
- Individual Peer Evaluati...
- Integer Linear Program...
- Interpretation of regre...
- Interpretation of regre...
- IRI\_POS\_Tablespreads...
- Kimball\_The-Data-War...

OUTLINE TIMELINE ZIP EXPLORER

```
# Print the misclassified rows
print("Misclassified Rows:")
print(misclassified_rows)
```

... Misclassified Rows:

	X_test	y_test	y_pred
76	18.015998	cw	cb
396	112.083527	cw	cb
220	57.471713	cw	cb
294	80.526149	cw	cb
234	99.769320	cw	cb
526	103.753146	cw	cb
672	197.483872	cb	cw
0	185.997680	cb	cw
228	65.714664	cw	cb

Spaces: 4 Cell 16 of 16

17°C Cloudy 2:15 AM 4/11/2024 ENG US

## Group 2

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

C: > Users > apala > Downloads > Group 2 week 8.ipynb > M4 Part d) > def classify\_c(value):

+ Code + Markdown | ▶ Run All ⚡ Restart ⚡ Clear All Outputs Variables Outline ...

base (Python 3.9.13)

EXPLORER

UNTITLED (WORKSPACE)

↓ C:

↓ Users

↓ apala

↓ Downloads

Final\_Account\_Statement.pdf

Final\_Account\_Statement.xls

Firefox Installer.exe

formula\_sheet\_mid.docx

G11.zip

Git-2.43.0-32-bit.exe

Git-2.43.0-64-bit.exe

GitHubDesktopSetup-...

GlobalProtect4.msi

Goal Programming an...

Group 1 week 7.ipynb

Group 2 week 7.ipynb

Group 2 week 8.ipynb

Group 2 week 9.ipynb

Group 09\_HW1 (1).docx

Group\_5\_ExtraCreditAs...

Group1\_week9\_update...

Group9\_HW2.zip

Hi b.docx

Homework 3\_S23.docx

Homework Assignmen...

Homework Assignmen...

Homework Assignmen...

Homework-1.pdf

House Rules.xlsx

OUTLINE

TIMELINE

ZIP EXPLORER

15.8s

y\_pred\_train = best\_rf\_model\_c.predict(X\_train)

# Calculate accuracy

accuracy\_train = accuracy\_score(y\_train, y\_pred\_train)

print("Training Accuracy:", accuracy\_train)

# Evaluate the model on the testing set

y\_pred\_test = best\_rf\_model\_c.predict(X\_test)

# Calculate accuracy

accuracy\_test = accuracy\_score(y\_test, y\_pred\_test)

print("Testing Accuracy:", accuracy\_test)

Training Accuracy: 0.9390450389434474

Testing Accuracy: 0.925575101488498

def classify\_c(value):

pred = best\_rf\_model\_c.predict([[value]])

original = encoder\_c.inverse\_transform(pred)

Spaces: 4 CRLF Cell 12 of 13

2:37 AM 4/11/2024 ENG US

File Edit Selection View Go Run Terminal Help

Untitled (Workspace)

/nb ● 21 2888 59.547505 1 0

22 1663 152.086922 1 0

23 3667 233.355114 0 1

24 3672 165.639055 1 0

25 1405 160.919331 1 0

26 3336 184.541861 1 0

27 3526 196.731890 0 1

28 1785 33.054475 1 0

29 3539 103.753146 1 0

30 302 65.805878 1 0

31 79 127.185698 1 0

32 2359 206.333241 0 1

33 1572 186.273074 1 0

34 3378 202.356291 0 1

35 956 135.673469 0 1

36 2990 195.236470 1 0

37 1833 156.382363 0 1

38 1605 157.485811 1 0

39 3287 55.376777 0 1

40 1486 10.164194 1 0

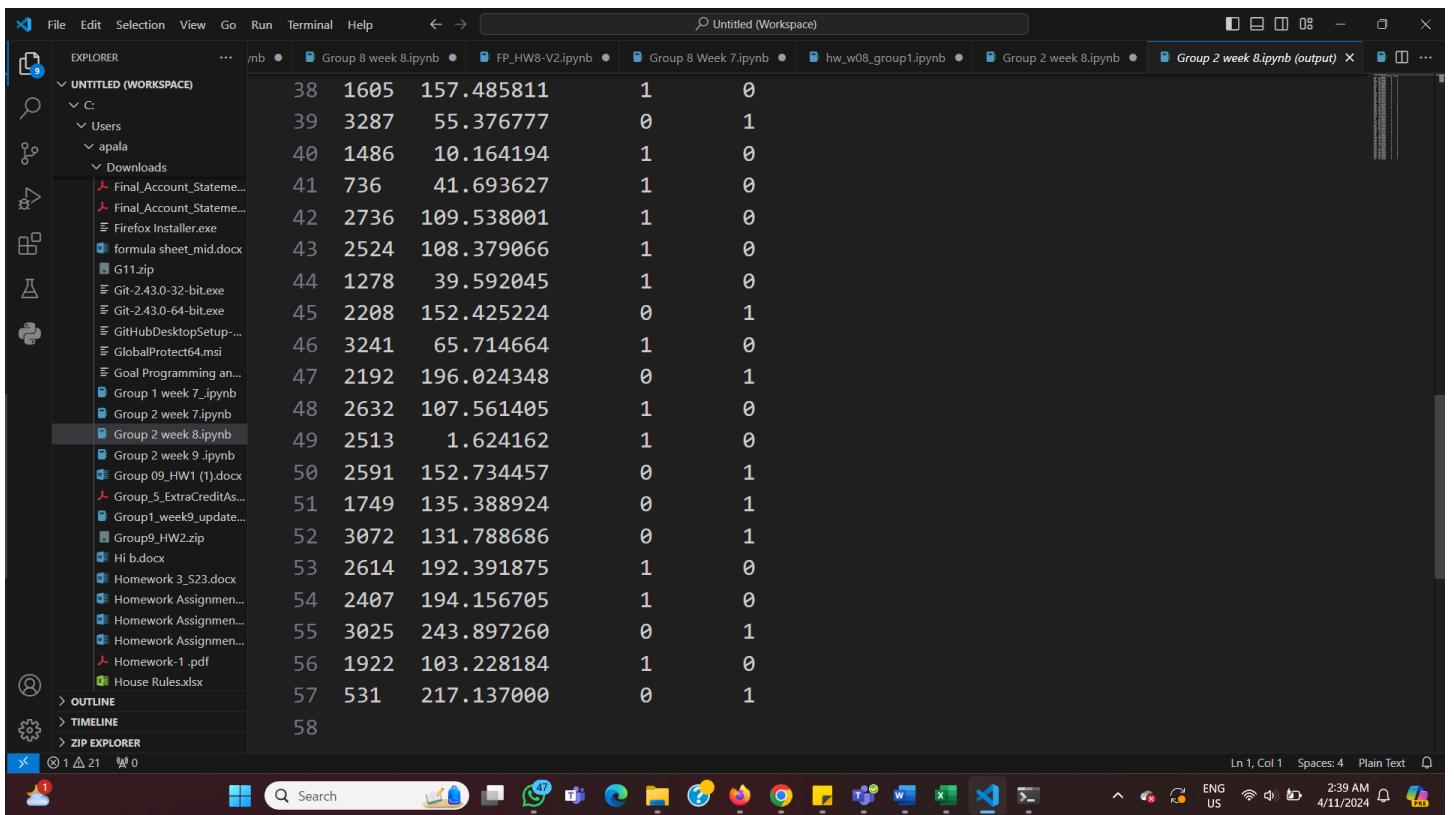
41 736 41.693627 1 0

42 2726 109.538001 1 0

Spaces: 4 Plain Text

Ln 1, Col 1

2:38 AM 4/11/2024 ENG US



## Group 3

```
[44] 1 prediction = n_cdf['value'].tolist()
2 predictions = classify_c(prediction)
3 actual = n_cdf['category'].tolist()
4
5 matching_count = 0
6
7 # Iterate through the elements of both lists simultaneously
8 for val1, val2 in zip(predictions, actual):
9     if val1 == val2:
10         matching_count += 1
11
12 total = len(n_cdf)
13 accuracy = matching_count/total
14 print('accuracy:',accuracy)

accuracy: 0.7587152369761065
```

```

✓ 0s 1 # Create a DataFrame containing y_test and y_pred
2 results_df = pd.DataFrame({'true_category': y_test.values, 'predicted_category': y_pred})
3
4 # Reset index of n_cdf
5 n_cdf_reset = n_cdf.reset_index(drop=True)
6
7 # Merge n_cdf_reset with results_df on their index
8 merged_df = pd.concat([n_cdf_reset, results_df], axis=1)
9
10 # Find misclassified rows where true_category is not equal to predicted_category
11 misclassified_rows_1 = merged_df[merged_df['true_category'] != merged_df['predicted_category']]
12 print("Misclassified Rows for Classifier 1 (c10):")
13 print(misclassified_rows_1)
14

```

Misclassified Rows for Classifier 1 (c10):

	category	value	true_category	predicted_category
7	cb	107.138563	cb	cw
8	cw	229.755125	cb	cw
9	cw	224.779218	cb	cw
11	cw	228.923615	cw	cb
23	cb	170.384366	cw	cb
...	...	...	...	...
2548	cb	179.653660	NaN	NaN
2549	cw	225.434341	NaN	NaN
2550	cb	190.148235	NaN	NaN
2551	cb	179.599881	NaN	NaN
2552	cb	180.115127	NaN	NaN

## Group 4

```

✓ 0s [22] 1 print("Accuracy:", round(100*(data[data['Cluster']] == data['Prediction']).shape[0]/data.shape[0]), 2), "%")
Accuracy: 4.56 %

```

```

✓ 0s [24] 1
2
3 # Find misclassified rows where the predicted cluster labels do not match the true cluster labels
4 misclassified_rows = data[data['Prediction'] != data['Cluster']]
5
6 # Print or do further analysis with misclassified_rows
7 print("Misclassified Rows:")
8 print(misclassified_rows)
9

```

Misclassified Rows:

	Cluster	Feature	Prediction
0	cb	105.279305	cw
1	cb	91.244381	cw
2	cw	230.445359	cb
3	cb	102.225524	cw
4	cw	226.920342	cb
...	...	...	...
762	cw	226.340258	cb
763	cw	223.494378	cb
764	cw	231.765622	cb
765	cb	103.423415	cw
766	cw	221.386075	cb

[732 rows x 3 columns]

## Group 5

```
[11] def classify_c(value):
    if value <= 200:
        return 'cb'
    else:
        return 'cw'

[12] predictions = testCdataset['value'].apply(classify_c)

[13] accuracy = (predictions == testCdataset['cb']).mean()
print("Accuracy:", accuracy)

Accuracy: 0.8815551537070524
```

▶ failure\_mask = predictions != testCdataset['cb']

# Extract the failure cases
failure\_cases = testCdataset[failure\_mask]

# Print the failure cases
print("Failure Cases:")
print(failure\_cases)

☞ Failure Cases:

	cb	value
45	cw	71.733092
50	cw	182.833195
58	cb	203.622196
140	cw	127.185698
146	cw	183.215341
...	...	...
969	cw	103.753146
977	cw	99.769320
1004	cw	150.728038
1019	cb	246.398694
1025	cw	88.743998

[131 rows x 2 columns]

```

▶ # Step 1: Prepare the data
X = testCdataset['value'].values.reshape(-1, 1)
y = testCdataset['cb']

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Step 3: Train a classifier
classifier = LogisticRegression()
classifier.fit(X_train, y_train)

# Step 4: Predict the target variable for the testing data
y_pred = classifier.predict(X_test)

# Step 5: Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the classifier:", accuracy)

```

→ Accuracy of the classifier: 0.7900677200902935

## Group 7

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Untitled (Workspace).
- Toolbar:** Includes icons for file operations like New, Open, Save, and Run All, along with Restart, Clear All Outputs, Variables, and Outline.
- Status Bar:** Shows the base Python version (3.9.13) and the current cell number (Cell 13 of 13).
- Left Sidebar (EXPLORER):** Shows the file system structure under 'UNTITLED (WORKSPACE)'.
- Code Cell:** Contains Python code for preparing data, splitting it into training and testing sets, training a logistic regression classifier, predicting the target variable for the testing data, and evaluating the accuracy of the classifier using the `accuracy\_score` function from `sklearn.metrics`.
- Output Cell:** Displays the accuracy rate: 0.7861975642760487.

The screenshot shows a Jupyter Notebook workspace in Visual Studio Code. The left sidebar displays a file tree under 'EXPLORER' titled 'UNTITLED (WORKSPACE)'. The tree includes various files and folders such as 'C:\Users\apala\Downloads', 'G11.zip', 'Git-2.43.0-32-bit.exe', 'GitHubDesktopSetup...', 'GlobalProtect4.msi', 'Goal Programming an...', 'Group 1 week 7.ipynb', 'Group 2 week 7.ipynb', 'Group 2 week 8.ipynb', 'Group 2 week 9.ipynb', 'Group 7 week8.ipynb', 'Group 09\_HW1 (1).docx', 'Group\_5\_ExtraCreditAs...', 'Group9\_HW2.zip', 'Group11\_week9\_update...', 'Hi.docx', 'Homework 3\_S23.docx', 'Homework Assignmen...', 'Homework Assignmen...', 'Homework-1.pdf', 'House Rules.xlsx', and 'house.csv'. Below the file tree are buttons for 'OUTLINE', 'TIMELINE', and 'ZIP EXPLORER'. The main area shows a Python script with code for finding misclassified indices in a logistic regression model. The output cell shows the result: 'Misclassified Indices: [8, 12, 26, 34, 45, 46, 54, 58, 65, 67, 68, 70, 77, 83, 85, 91, 95, 99, 101, 114, 117, 119, 126, 128, 130, 131, 137, 145, 148, 160, 161, 164, 168, 169, 171, 180, 181, 182, 191, 199, 201, 204, 226, 232, 233, 235, 236, 237, 247, 249, 251, 259, 264, 268, 278, 280, 282, 283, 286, 289, 293, 299, 300, 302, 305, 314, 315, 320, 323, 324, 328, 333, 336, 337, 339, 341, 344, 350, 352, 358, 359, 361, 363, 372, 374, 375, 376, 383, 395, 397, 414, 418, 420, 424, 427, 428, 435, 441, 442, 443, 451, 463, 464, 467, 473, 476, 478, 480, 485, 486, 487, 488, 490, 495, 507, 508, 515, 522, 524, 528, 531, 535, 539, 548, 549, 553, 554, 555, 556, 561, 566, 571, 595, 611, 618, 620, 625, 634, 640, 645, 648, 653, 655, 658, 664, 678, 683, 686, 689, 691, 692, 695, 698, 714, 716, 722, 731, 734]'

```
def misclassified_indices(input_vector, true_labels, log_model):
    # Reshape the input vector to a 2D array
    input_vector_2d = np.array(input_vector).reshape(-1, 1)
    # Predict using the logistic regression model
    predictions = log_model.predict(input_vector_2d)
    # Find misclassified indices
    misclassified_indices = [i for i, (pred, true) in enumerate(zip(predictions, true_labels)) if pred != true]
    return misclassified_indices

# Example usage
misclassified_idx = misclassified_indices(X_test_reshaped, y_test, log_model)
print("Misclassified Indices:", misclassified_idx)
```

All rows that were misclassified:

[8, 12, 26, 34, 45, 46, 54, 58, 65, 67, 68, 70, 77, 83, 85, 91, 95, 99, 101, 114, 117, 119, 126, 128, 130, 131, 137, 145, 148, 160, 161, 164, 168, 169, 171, 180, 181, 182, 191, 199, 201, 204, 226, 232, 233, 235, 236, 237, 247, 249, 251, 259, 264, 268, 278, 280, 282, 283, 286, 289, 293, 299, 300, 302, 305, 314, 315, 320, 323, 324, 328, 333, 336, 337, 339, 341, 344, 350, 352, 358, 359, 361, 363, 372, 374, 375, 376, 383, 395, 397, 414, 418, 420, 424, 427, 428, 435, 441, 442, 443, 451, 463, 464, 467, 473, 476, 478, 480, 485, 486, 487, 488, 490, 495, 507, 508, 515, 522, 524, 528, 531, 535, 539, 548, 549, 553, 554, 555, 556, 561, 566, 571, 595, 611, 618, 620, 625, 634, 640, 645, 648, 653, 655, 658, 664, 678, 683, 686, 689, 691, 692, 695, 698, 714, 716, 722, 731, 734]

## Group 8

The screenshot shows a Jupyter Notebook interface with the title bar "Untitled (Workspace)". The left sidebar displays a file tree under "C:\Users\apala\Downloads". A code cell in the main area contains Python code to calculate accuracy from a trained model. The output cell shows the calculated accuracy rate.

```
return output_vector
```

```
from sklearn.metrics import accuracy_score

# Use the trained model to predict the test set
y_pred = log_model.predict(X_test)

# Calculate the accuracy rate
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy Rate:", accuracy)
```

... Accuracy Rate: 0.7821380243572396

The screenshot shows a Jupyter Notebook interface with the title bar "Untitled (Workspace)". The left sidebar displays a file tree under "C:\Users\apala\Downloads". A code cell in the main area prints the number of misclassified rows and then displays a table of misclassified rows with columns for Actual and Predicted labels.

```
print("Misclassified Rows:")
print(misclassified_rows)
```

... Misclassified Rows:

	Actual	Predicted
2642	cw	cb
2374	cb	cw
1422	cb	cw
1402	cw	cb
1578	cw	cb
...	...	...
2454	cb	cw
1710	cw	cb
1433	cw	cb
1507	cw	cb
2991	cw	cb

[161 rows x 2 columns]

## Group 9

Group\_9\_week\_8.ipynb ● Group\_9\_week\_9.ipynb ●

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 9 > Group\_9\_week\_8.ipynb > #1 (a)

+ Code + Markdown | ▶ Run All ⏪ Restart ⏷ Clear All Outputs ⏷ Go To | Variables Outline ... Python 3.12.2

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.20, random_state=42)

# Initialize Logistic Regression Model with default parameters
simple_log_model = LogisticRegression()
simple_log_model.fit(X_train, y_train)

# Predicting and Evaluating the model
y_pred = simple_log_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))

# Simplified prediction function
def simplified_categorize(measurement_vector):
    measurement_vector_scaled = scaler.transform(np.array(measurement_vector).reshape(-1, 1))
    predictions = simple_log_model.predict(measurement_vector_scaled)
    return predictions.tolist()

# Testing the simplified approach
measurement_values = [50, 150, 200, 80, 120]
print("Categorized Labels:", simplified_categorize(measurement_values))
```

[ ] Python

... Accuracy: 0.881203007518797  
Categorized Labels: ['cb', 'cw', 'cw', 'cb', 'cw']  
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MinMaxScaler was fitted with feature warnings.warn(

Group\_9\_week\_8.ipynb ● Group\_9\_week\_7\_v2.ipynb ● Group 10 week 7.ipynb ● Group 10 week 8.ipynb ●

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 9 > Group\_9\_week\_8.ipynb > empty cell

+ Code + Markdown | ▶ Run All ⏪ Restart ⏷ Clear All Outputs | Variables Outline ...

```
# Print the DataFrame of incorrect predictions
print("Incorrect Predictions:")
print(incorrect_predictions)

# Print the percentage of incorrect predictions
print("Percentage of Incorrect Predictions:", incorrect_percentage, "%")
```

[12] ✓ 0.0s

... Incorrect Predictions:

Measurements	True_Category	Predicted_Category
1565	0.543374	cw
2271	0.616143	cb
1642	0.529491	cw
1940	0.589895	cb
1553	0.533628	cw
...	...	...
3004	0.646437	cb
2657	0.304327	cw
2487	0.613929	cb
2080	0.653532	cb
2034	0.646255	cb

[160 rows x 3 columns]

Percentage of Incorrect Predictions: 21.650879566982407 %

## Group 10

Group 10 week 9.ipynb • Group 10 week 8.ipynb X

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 10 > Group 10 week 8.ipynb

+ Code + Markdown | ▶ Run All ⚡ Restart ⌂ Clear All Outputs | ⌂ Variables ⌂ Outline ⋮

```
▶ from sklearn.metrics import accuracy_score

# Using the classify_c function to predict classes for the test set
predicted_classes = classify_c(X_test)

# Calculating accuracy
accuracy = accuracy_score(y_test, predicted_classes)
print("Accuracy:", accuracy)
```

[19]

... Accuracy: 0.7886178861788617

Group\_9\_week\_8.ipynb • Group\_9\_week\_7\_v2.ipynb • Group 10 week 7.ipynb • Group 10 week 8.ipynb • Group\_9\_week\_9.ipynb •

Up 10 - Week 08: FP-HW > m d) Create a dynamic criteria that is able to distinguish between the values belonging to each class based on the file. Write a function called cla

+ Code + Markdown | ▶ Run All ⚡ Restart ⌂ Clear All Outputs | ⌂ Variables ⌂ Outline ⋮

```
# Print the datarame of incorrect predictions
print("Incorrect Predictions:")
print(incorrect_predictions)

# Print the percentage of incorrect predictions
print("Percentage of Incorrect Predictions:", incorrect_percentage, "%")
```

[18] ✓ 0.0s

... Incorrect Predictions:

	Values	True_Class	Predicted_Class
724	181.691275	cb	cw
502	163.912628	cb	cw
510	198.420924	cb	cw
294	80.526149	cw	cb
418	148.921252	cw	cb
..	...	...	...
300	191.726892	cb	cw
107	148.993852	cw	cb
162	164.194963	cb	cw
324	11.464411	cw	cb
36	29.171277	cw	cb

[156 rows x 3 columns]

Percentage of Incorrect Predictions: 21.138211382113823 %

## O Classifier

Group1

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Back, Forward, Search, Untitled (Workspace).
- Left Sidebar (EXPLORER):** UNTITLED (WORKSPACE) folder containing C:, Users, apala, Downloads, and various Excel files (Fig11-1.xlsm to Fig14-41.xlsm), Final\_Account\_Statement, Firefox Installer.exe, and formula sheet\_mid.docx.
- Code Cell:** Contains Python code for a classifier. The output shows the accuracy is 0.9975.
- Output Cell:** Shows the values\_vector\_1 and classification variables.
- Bottom Status Bar:** Spaces: 4, Cell 13 of 15, 2:45 AM, 4/11/2024.

## Group 2

### Group 2 has no misclassification in o classifier

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Back, Forward, Search, Untitled (Workspace).
- Left Sidebar (EXPLORER):** UNTITLED (WORKSPACE) folder containing C:, Users, apala, Downloads, and various files including Git, Goal Programming, and Jupyter notebooks (Group 1 week 7.ipynb to Group 2 week 9.ipynb).
- Code Cell:** Contains Python code for a classifier. The output shows the accuracy is 1.0.
- Bottom Status Bar:** Spaces: 4, Cell 11 of 12, 2:41 AM, 4/11/2024.

### Group 3

```
0s 2 from sklearn import svm
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score
5
6 # Assuming X_train, X_test, y_train, y_test are your feature and target variables
7 X = df[['a','b','c', 'd', 'e', 'f', 'g', 'h']].values
8 Y = df['class']
9 # Split the data into training and testing sets
10 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
11
12 # Create SVM classifier with a linear kernel
13 svm_classifier = svm.SVC(kernel='linear')
14
15 # Train the SVM classifier
16 svm_classifier.fit(X_train, y_train)
17
18 # Predict the labels for the test set
19 y_pred = svm_classifier.predict(X_test)
20
21 # Calculate accuracy
22 accuracy = accuracy_score(y_test, y_pred)
23 print('Accuracy: ', accuracy)
24 print("SVM Parameters:")
25 print(svm_classifier.get_params())
26
```

```
Accuracy: 0.995
SVM Parameters:
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': 42, 'tol': 0.001, 'verbose': False}
```

```
0s 1 # Find misclassified indices
2 misclassified_indices = np.where(y_test != y_pred)[0]
3
4 # Extract misclassified rows using the misclassified indices
5 misclassified_rows = df.iloc[misclassified_indices]
6
7 # Display the misclassified rows
8 print("Misclassified Rows:")
9 print(misclassified_rows)
10
```

```
Misclassified Rows:
   class      a      b      c      d      e      f \
116     1  1.981851  0.00327  0.008875  0.011671  0.030388  0.002271

      g      h
116  0.016106  0.024919
```

### Group 4

```

1s 0 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
1 o1 2.125414 0.015460 0.008512 0.013443 0.052705 0.011392
2 o1 1.829091 0.004966 0.009709 0.002849 0.098940 0.086140
3 o1 2.106705 0.000282 0.010213 0.003058 0.082583 0.039043
4 o5 1.178369 0.004911 0.013323 0.001259 0.041105 1.140624
.. ...
994 o1 2.103797 0.013355 0.001890 0.016420 0.105692 0.049285
995 o1 2.002819 0.000202 0.001644 0.018230 0.015025 0.058542
996 o6 0.989879 0.904669 0.003790 0.765534 0.039305 0.884575
997 o6 0.979193 1.150142 0.004176 0.740266 0.030199 0.865416
998 o8 0.935741 0.929695 0.148255 0.733941 0.008351 0.039278

    Feature7  Feature8 Prediction
0  1.107407  0.089122      o3
1  0.039739  0.003396      o1
2  0.032137  0.051206      o1
3  0.027789  0.028874      o1
4  0.752167  0.003102      o5
.. ...
994 0.038040  0.037962      o1
995 0.038257  0.086832      o1
996 0.062832  0.073284      o6
997 0.009247  0.097826      o6
998 0.021517  0.037740      o8

[999 rows x 10 columns]
Accuracy: 100.0 %

=====
Actual Class: o1
Predicted output from Random Forest: o1
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was
warnings.warn(

```

```

0s 1 # Filter the DataFrame for misclassified rows
2 misclassified_rows = data[data['cluster'] != data['Prediction']]
3
4 # Print the misclassified rows
5 print("Misclassified Rows:")
6 print(misclassified_rows)
7

Misclassified Rows:
Empty DataFrame
Columns: [Cluster, Feature1, Feature2, Feature3, Feature4, Feature5, Feature6, Feature7, Feature8, Prediction]
Index: []

```

0

**Group 5**

```
#logistic Regression
import pandas as pd
from sklearn.model_selection import train_test_split

# Split the dataset into features and target variable
X = data.drop(columns=['Class']) # Features
y = data['Class'] # 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)

# Initialize the logistic regression model
log_reg = LogisticRegression(max_iter=1000, random_state=42)

# Train the model
log_reg.fit(X_train, y_train)

# Predict on the test set
y_pred = log_reg.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Accuracy: 0.995

▶ # Print misclassified labels  
print("Misclassified Labels:")  
for index, row in misclassified\_samples.iterrows():  
 print(f"True Label: {row['True\_Label']}, Predicted Label: {row['Predicted\_Label']}")

Misclassified Labels:  
True Label: o4, Predicted Label: o9

```
▶ #using logistic regression

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Prepare the data for all feature columns (col2 to col9)
X = data_df.iloc[:, 1: ].values # Extract feature values
y = data_df['col1'].values # Extract class labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Train a logistic regression model
clf = LogisticRegression(random_state=56)
clf.fit(X_train, y_train)

# Classification function for all features
def classify_c(values_vector):
    class_label = clf.predict(np.array(values_vector).reshape(1, -1))[0]
    return class_label

# Predict the class labels for the test set
y_pred = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

→ Accuracy: 0.9975

```
[10] # Find misclassified files
misclassified_indices = np.where(y_pred != y_test)[0]

# Print misclassified files
print("Misclassified Files:")
for index in misclassified_indices:
    print(data_df.iloc[index])
```

Misclassified Files:

col1	o7
col2	1.977721
col3	0.000187
col4	0.014306
col5	0.006106
col6	1.126774
col7	0.032714
col8	1.001159
col9	0.048462

Name: 118, dtype: object

```
[ ] # Apply the classify_o function to the features to get predictions
predictions = df[['Feature1', 'Feature2', 'Feature3', 'Feature4', 'Feature5', 'Feature6', 'Feature7', 'Feature8']].apply(classify_o, axis=1)

# Compare the predicted labels with the actual labels from the dataset
accuracy = (predictions == df['Class']).mean()

print("Accuracy:", accuracy)

Accuracy: 0.997
```

## Group 8

```
✓ [6] 1 # Split the data into training and testing sets
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
3
4 # Train an SVM model
5 clf = SVC(random_state=42) # Use SVC instead of LogisticRegression
6 clf.fit(X_train, y_train)
7
8 # Classification function for all features
9 def classify_o(values_vector):
10    class_label = clf.predict(np.array(values_vector).reshape(1, -1))[0]
11    return class_label
12
13
14 # Predict the class labels for the test set
15 y_pred = clf.predict(X_test)
16
17 # Calculate the accuracy of the model
18 accuracy = accuracy_score(y_test, y_pred)
19 print("Accuracy:", accuracy)
```

Accuracy: 0.995

```
✓ [ ] 1 # Find misclassified rows
2 misclassified_indices = np.where(y_pred != y_test)[0]
3
4 # Extract misclassified rows using the misclassified indices
5 misclassified_rows = df.iloc[misclassified_indices]
6
7 # Print the misclassified rows
8 print("Misclassified Rows:")
9 print(misclassified_rows)
10
```

Misclassified Rows:

	Class	Feature1	Feature2	Feature3	Feature4	Feature5	Feature6	Feature7	Feature8
118	07	1.977721	0.000187	0.014306	0.006106	1.126774	0.032714		
118		1.001159	0.048462						

## Group 9

The screenshot shows a Jupyter Notebook interface with the title bar "Group\_9\_week\_9.ipynb". The code cell contains three examples of feature vectors and their predicted classes:

```
new_vector1 = [0.5, 0.4, 0.6, 0.7, 0.3, 0.2, 0.1, 0.8] # Example feature vector
predicted_class1 = classify_c(new_vector1)
print(f"The predicted class for the new vector is: {predicted_class1}")

new_vector2 = [0.9, 0.1, 0.2, 0.8, 0.7, 0.3, 0.6, 0.4] # A new example feature vector
predicted_class2 = classify_c(new_vector2)
print(f"The predicted class for the new vector is: {predicted_class2}")

new_vector3 = [2.10790574080421, 0.00507871131822994, 0.4, 0.00925471115228404, 0.822137483440471, 0.0229208351721836, 0.924457327487032, 0.088936
               # A new example feature vector
predicted_class3 = classify_c(new_vector3)
print(f"The predicted class for the new vector is: {predicted_class3}")
```

The output cell [9] shows the results:

```
[9] ✓ 0.0s
...
The predicted class for the new vector is: o2
The predicted class for the new vector is: o8
The predicted class for the new vector is: o7
c:\Users\shagu\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but Gr
    warnings.warn(
c:\Users\shagu\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but Gr
    warnings.warn(
c:\Users\shagu\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but Gr
    warnings.warn(
```

The screenshot shows a Jupyter Notebook interface with the title bar "Group\_9\_week\_9.ipynb". The code cell contains two examples of feature vectors and their predicted classes:

```
Predicted_Class
116      o1
165      o1
Percentage of Incorrect Predictions: 1.0 %
```

The output cell [12] shows the results:

```
[12] ✓ 0.0s
...
Incorrect Predictions:
   Features True_Class \
116  [1.0926703805801403, 0.7664855662915072, 0.214...      o4
165  [1.2686880978561563, 0.7722584932167964, 0.295...      o9

Predicted_Class
116      o1
165      o1
```

## Group 10

Group 10 week 9.ipynb

Group 10 week 8.ipynb

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 10 > Group 10 week 9.ipynb > from collections import Counter  
+ Code + Markdown | Run All ⚡ Restart ⌛ Clear All Outputs | Variables ⌗ Outline ...  
The input vector doesn't belong to any class - Returning most frequent class  
The input vector belongs to class o4  
The input vector belongs to class o2  
The input vector belongs to class o2  
The input vector belongs to class o8  
The input vector belongs to class o4  
The input vector belongs to class o7  
The input vector doesn't belong to any class - Returning most frequent class  
The input vector belongs to class o7  
The input vector belongs to class o4  
The input vector belongs to class o4  
The input vector belongs to class o8  
The input vector belongs to class o1  
The input vector belongs to class o8  
The input vector belongs to class o6  
The input vector belongs to class o3  
The input vector belongs to class o1  
The input vector belongs to class o1  
The input vector belongs to class o1  
The input vector doesn't belong to any class - Returning most frequent class  
The input vector belongs to class o0  
...  
The input vector belongs to class o6  
The input vector belongs to class o6  
The input vector belongs to class o8  
Accuracy: 0.8758758758758759

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

Group\_9\_week\_8.ipynb

Group\_9\_week\_7\_v2.ipynb

Group 10 week 7.ipynb

Group 10 week 8.ipynb

Group\_9\_week\_9.ipynb

Group 10 week 9.ipynb

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 10 > Group 10 week 9.ipynb > empty cell

+ Code + Markdown | Run All ⚡ Restart ⌛ Clear All Outputs | Variables ⌗ Outline ...  
The input vector doesn't belong to any class - Returning most frequent class  
The input vector belongs to class o4  
The input vector belongs to class o2  
The input vector belongs to class o2  
The input vector belongs to class o8  
The input vector belongs to class o4  
The input vector belongs to class o7  
The input vector doesn't belong to any class - Returning most frequent class  
The input vector belongs to class o7  
The input vector belongs to class o4  
The input vector belongs to class o4  
The input vector belongs to class o8  
The input vector belongs to class o1  
The input vector belongs to class o8  
The input vector belongs to class o6  
The input vector belongs to class o3  
The input vector belongs to class o1  
The input vector belongs to class o1  
The input vector belongs to class o1  
The input vector doesn't belong to any class - Returning most frequent class  
The input vector belongs to class o0  
...  
940 | o5 | o1  
960 | o4 | o1  
968 | o4 | o1  
975 | o5 | o1

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...