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.

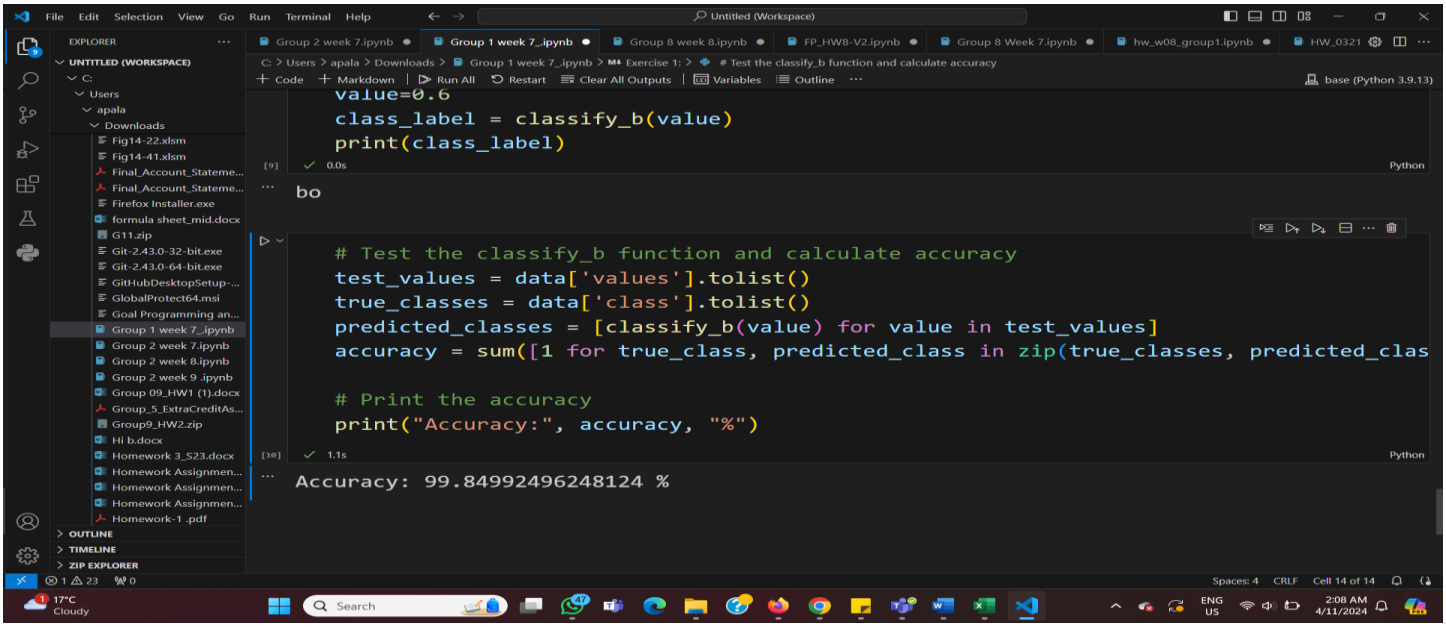
Defining Test Objectives: The test objective is that each classifier of all 10 groups should perform with an accuracy greater than 80 percent. We intend to share feedback with the groups according to the accuracies we obtained for their respective functions.

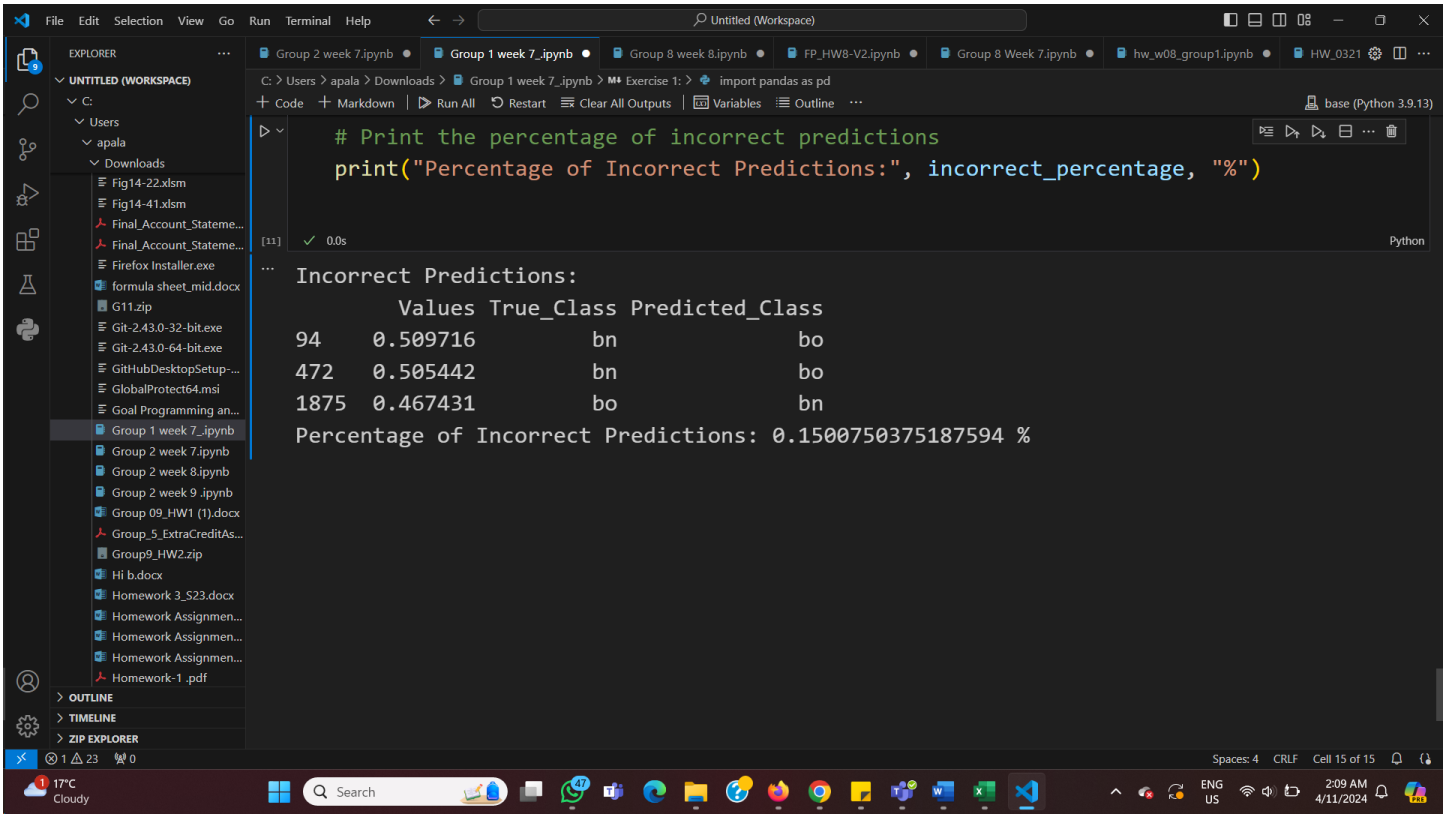
Test Cases

B Classifier

Group1

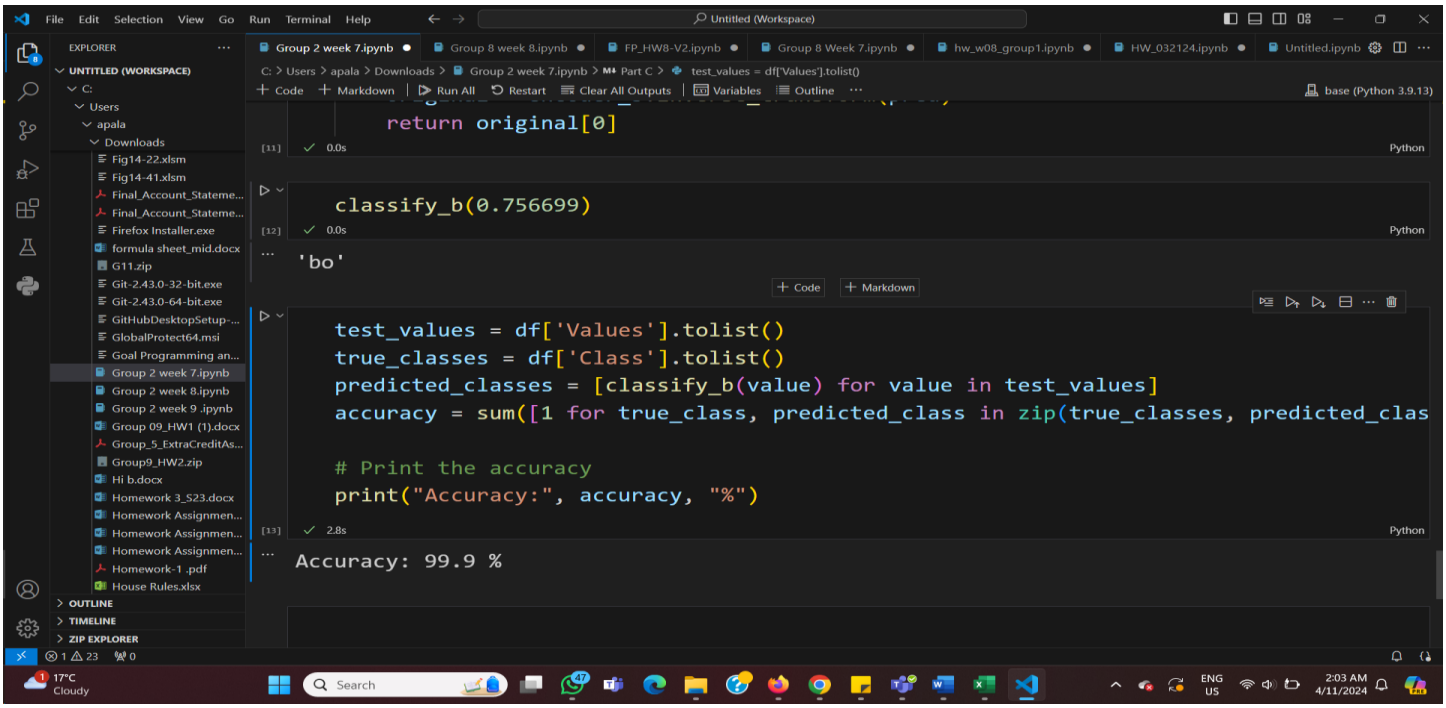
Group 1's B classifier has an accuracy of 99.84% for our data. We believe with almost perfect accuracy there is no need to improvise this function. The just took 2 seconds to execute.

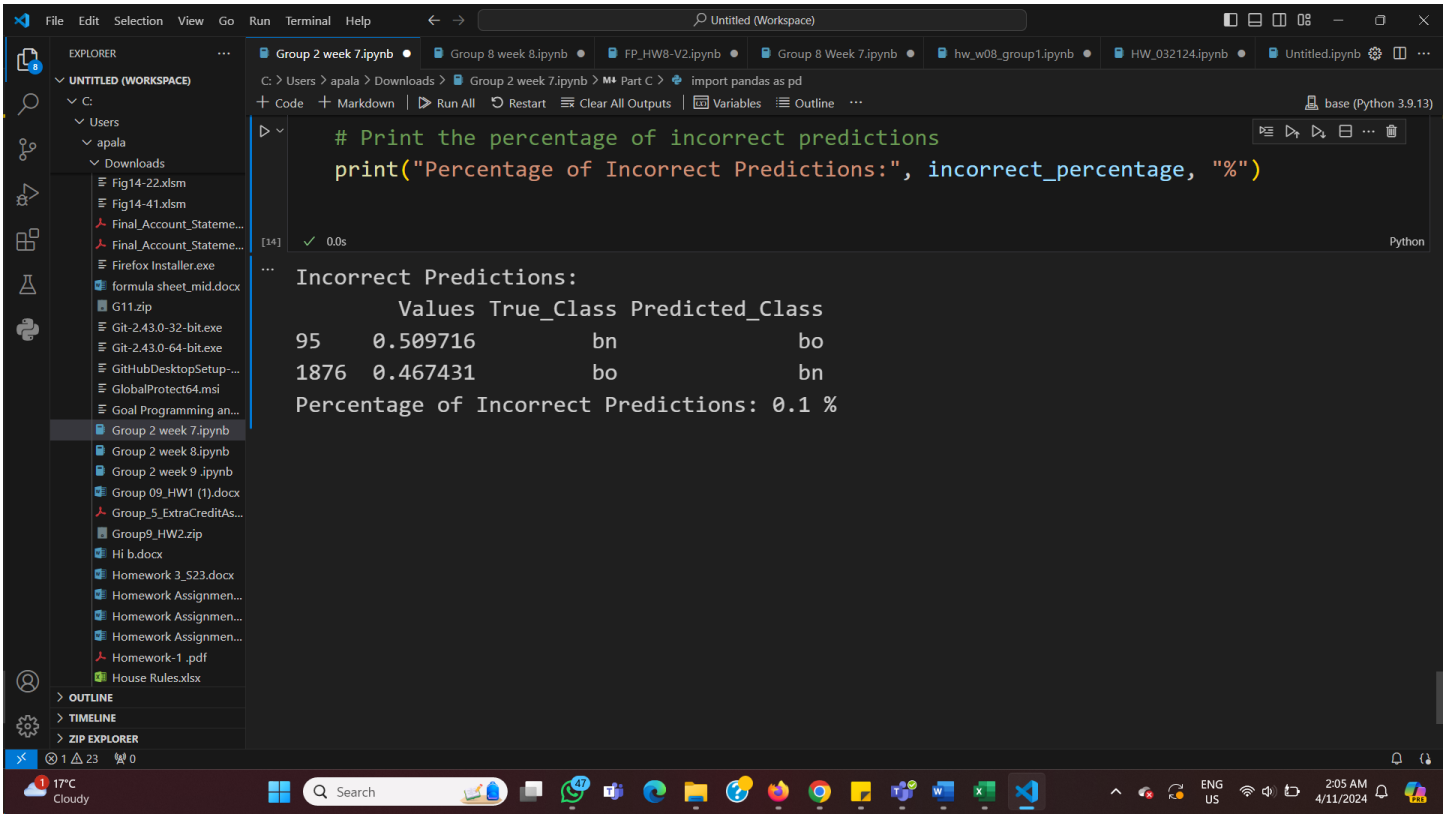




Group 2

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





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

```
[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
```

```
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
```

```
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      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)
```

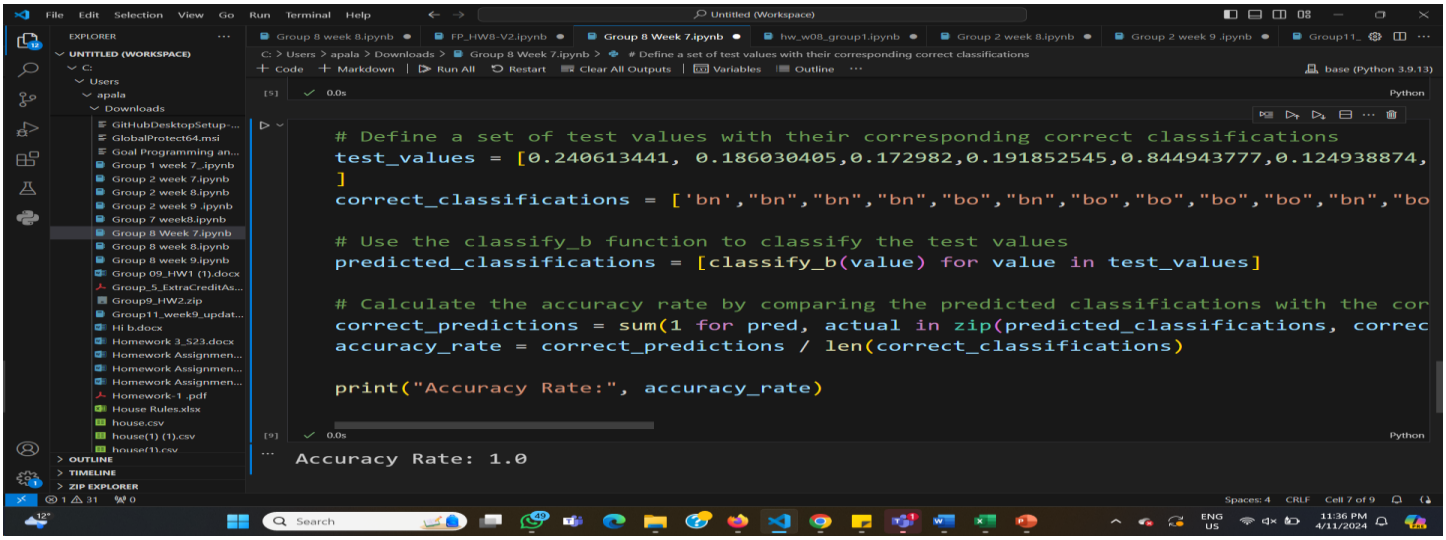
```
Accuracy: 0.9985
```

```
# 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)
```

```
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



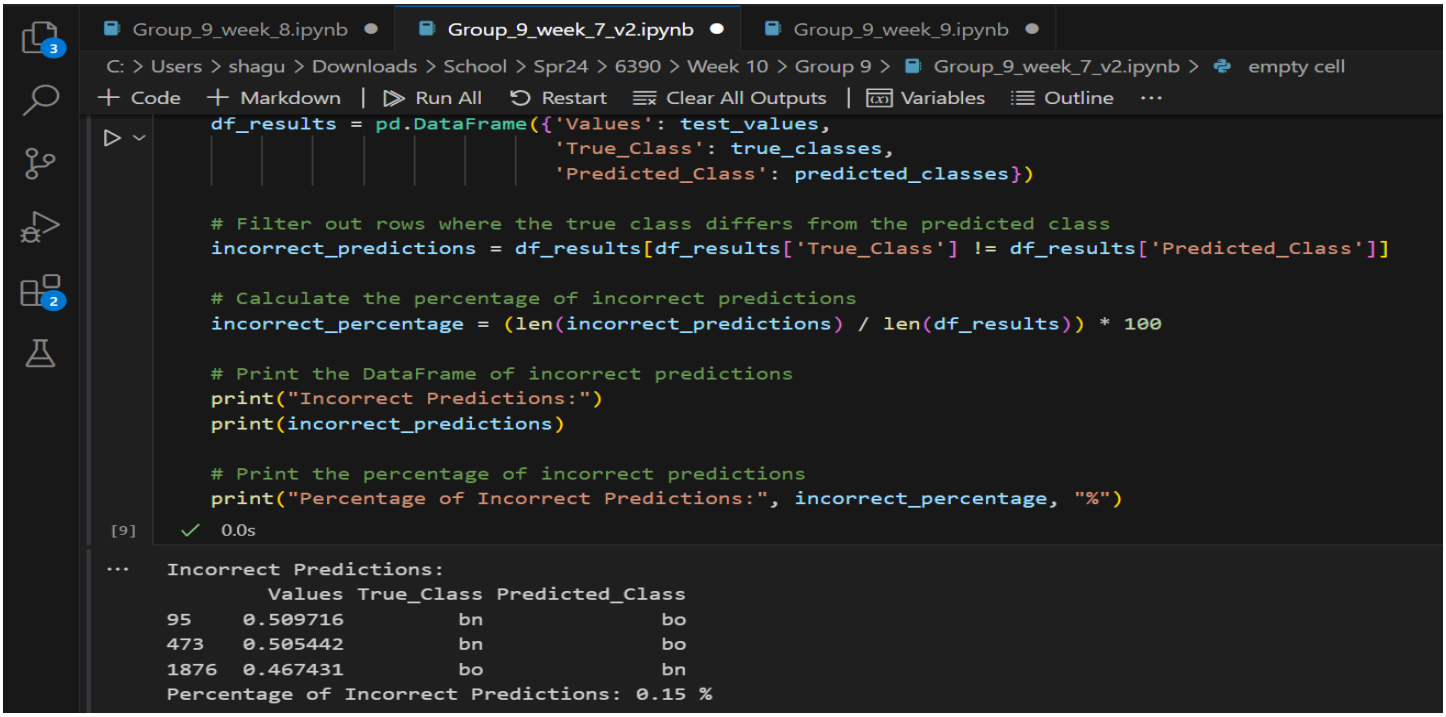
Group 9

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

```python
data = pd.read_csv(r"C:\Users\shagu\Downloads\School\Spr24\6390\Week 10\G11\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
```



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 × | Market_Basketipynb (1) (1).ipynb ● | Group 11 week 7.ipynb

C: > Users > shagu > Downloads > School > Spr24 > 6390 > Week 10 > Group 10 > Group 10 week 7.ipynb > M↓ 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 > M↓ 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 ···

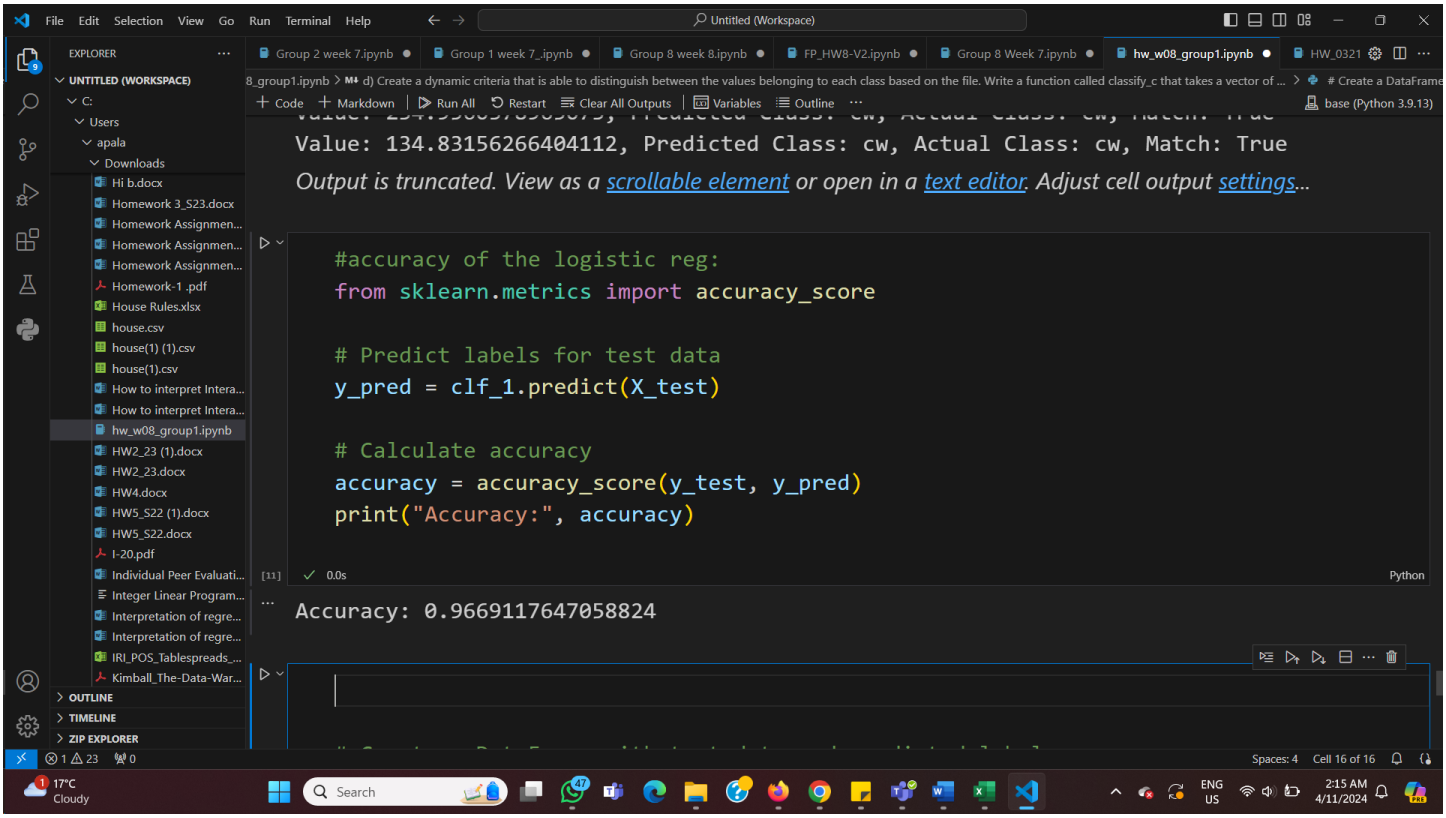
```
# Create a DataFrame with test values, true classes, and predicted classes
df_results = pd.DataFrame({'Values': labeled_dataset.iloc[:, 1],  # Test values
                           'True_Class': actual_labels,            # True classes
                           'Predicted_Class': predicted_labels})  # 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, "%")
```

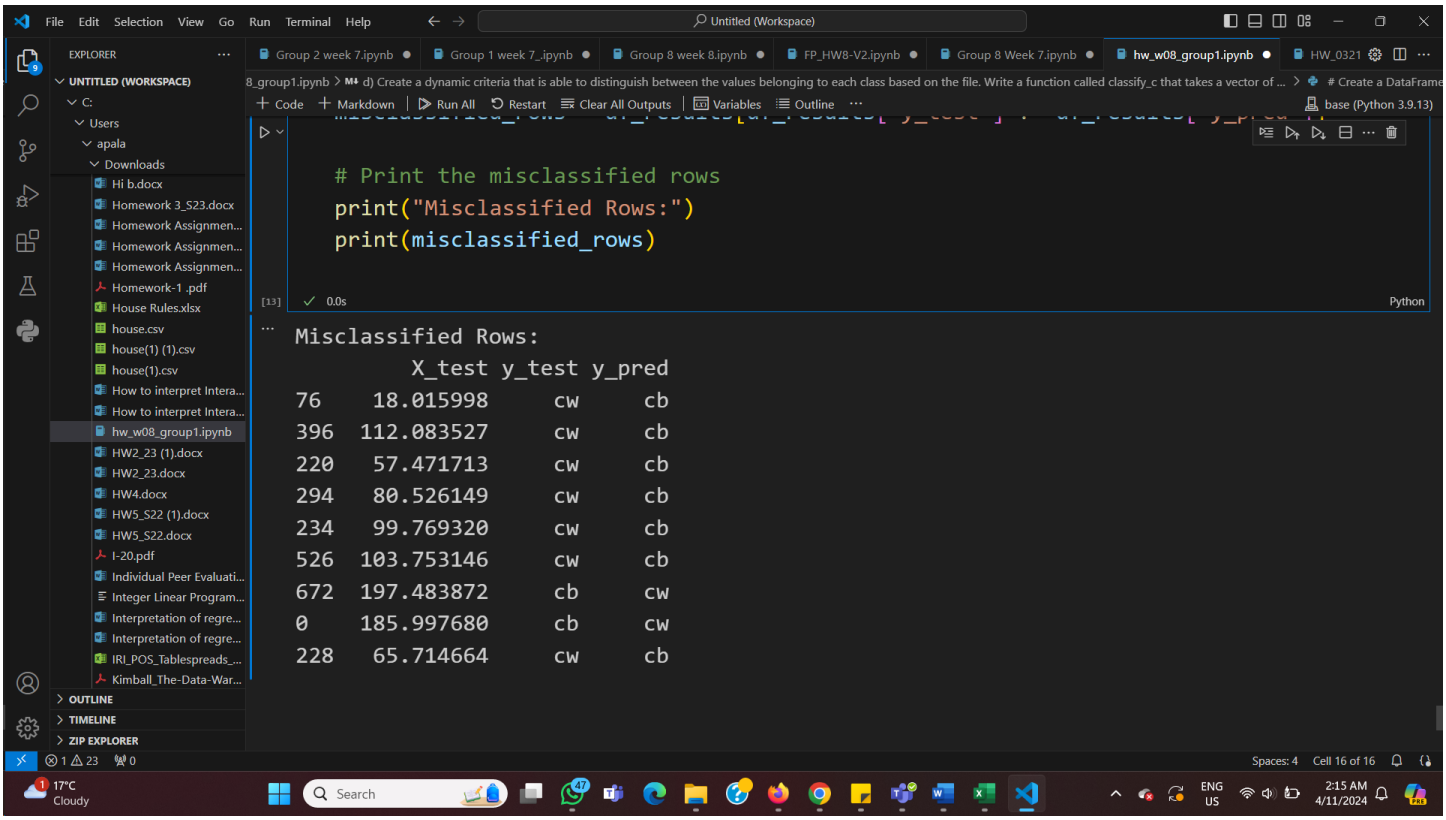
[16] ✓ 0.0s

```
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 %
```

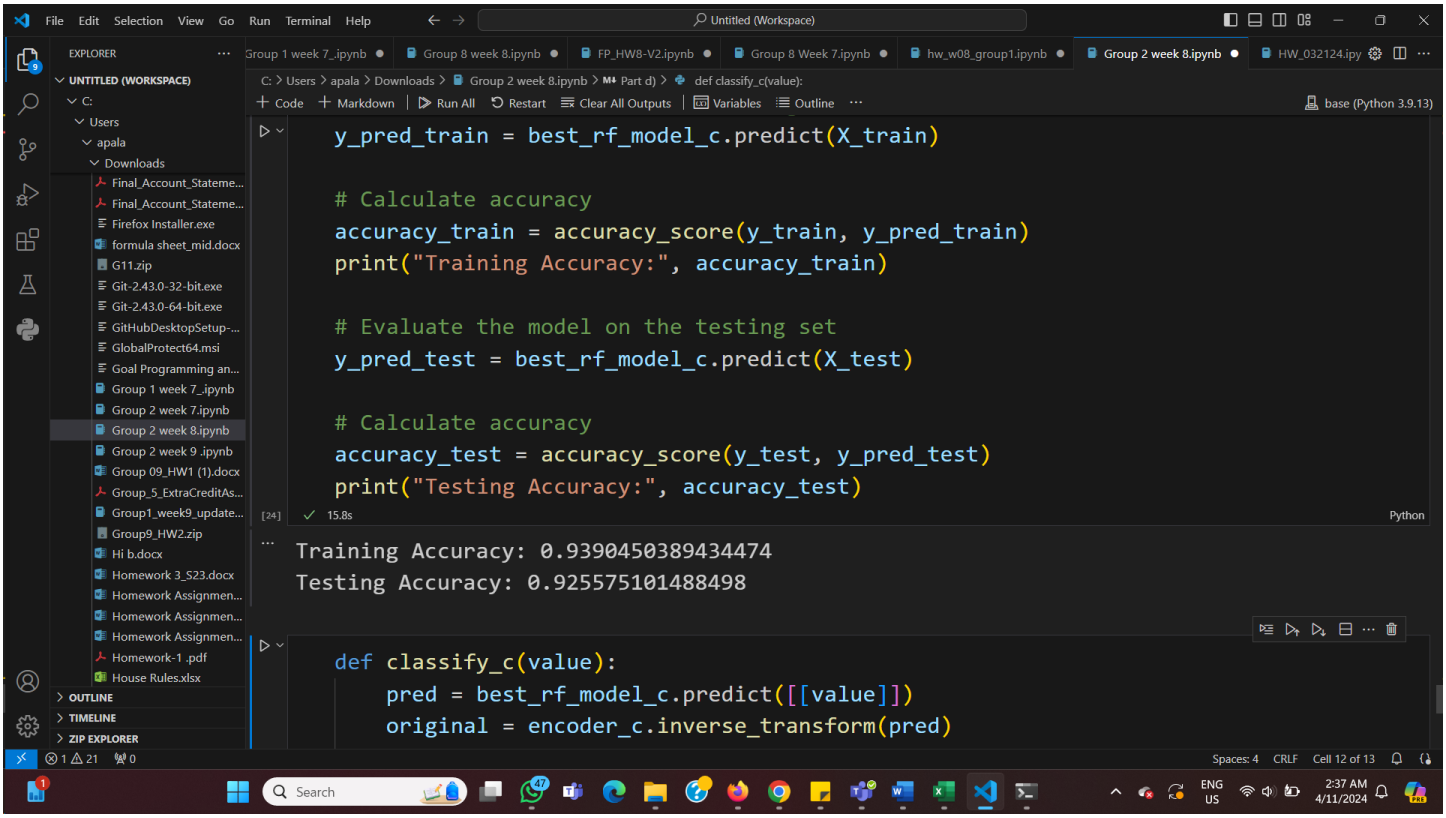
C Classifier

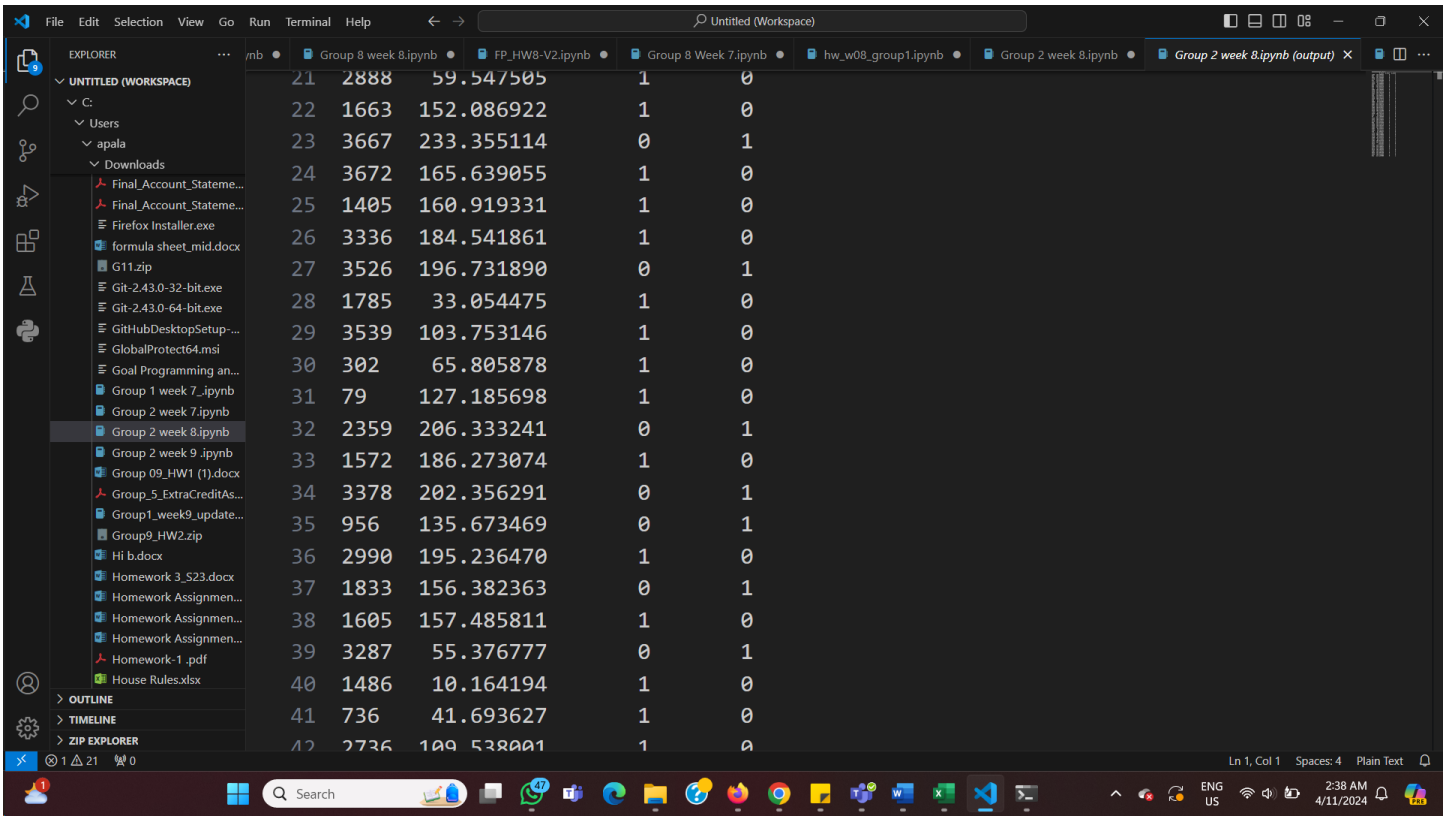
Group1

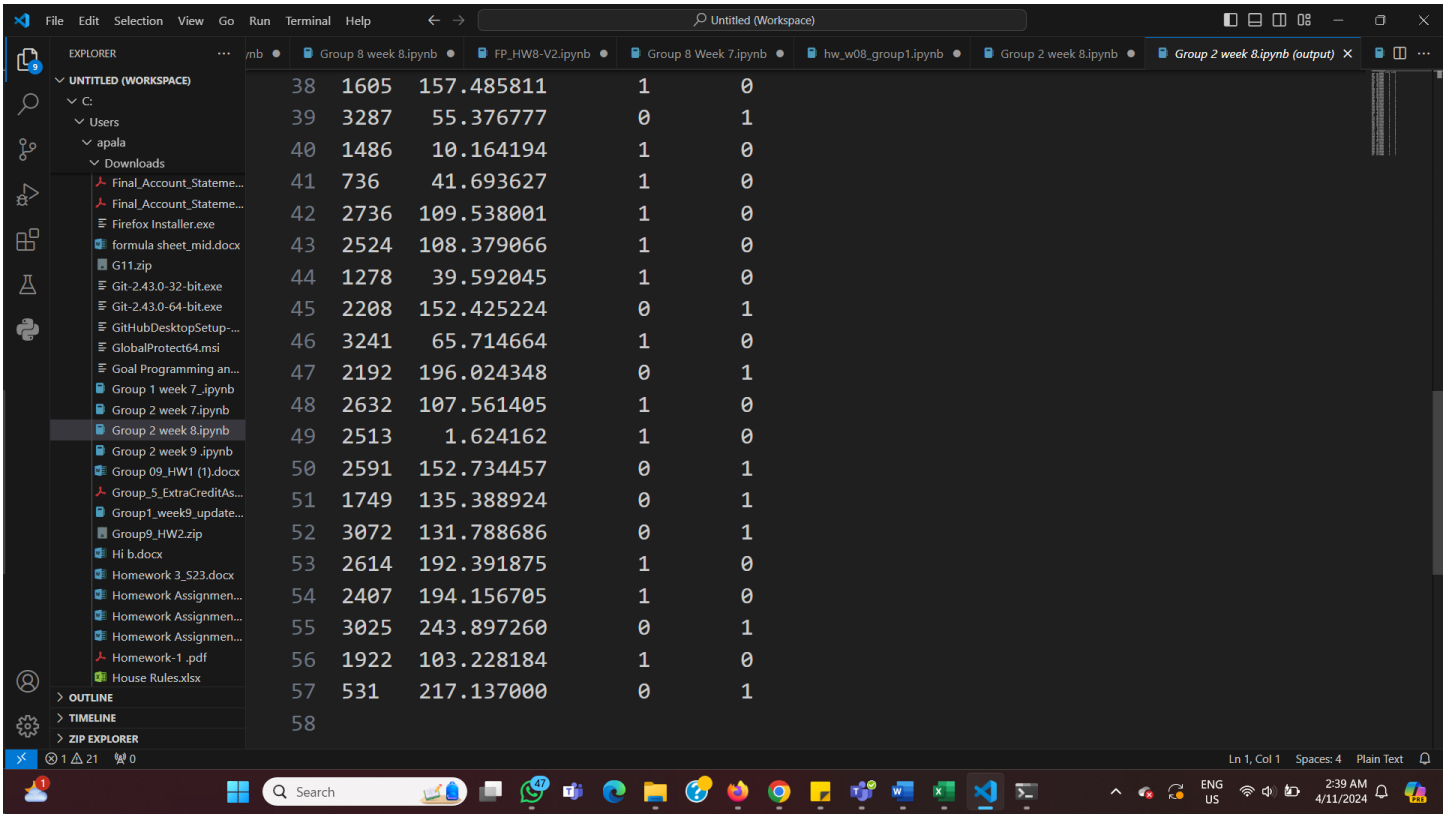

Value: 134.83156266404112, Predicted Class: cw, Actual Class: cw, Match: True
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
#accuracy of the logistic reg:
from sklearn.metrics import accuracy_score
# Predict labels for test data
y_pred = clf_1.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
Accuracy: 0.9669117647058824


# 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

Group 2


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)


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



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
```

```
# Create a DataFrame containing y_test and y_pred
results_df = pd.DataFrame({'true_category': y_test.values, 'predicted_category': y_pred})

# Reset index of n_cdf
n_cdf_reset = n_cdf.reset_index(drop=True)

# Merge n_cdf_reset with results_df on their index
merged_df = pd.concat([n_cdf_reset, results_df], axis=1)

# Find misclassified rows where true_category is not equal to predicted_category
misclassified_rows_1 = merged_df[merged_df['true_category'] != merged_df['predicted_category']]
print("Misclassified Rows for Classifier 1 (c10):")
print(misclassified_rows_1)

```

```
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

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

```
Accuracy: 4.56 %
```

```
[24]
# Find misclassified rows where the predicted cluster labels do not match the true cluster labels
misclassified_rows = data[data['Prediction'] != data['Cluster']]

# Print or do further analysis with misclassified_rows
print("Misclassified Rows:")
print(misclassified_rows)

```

```
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
```

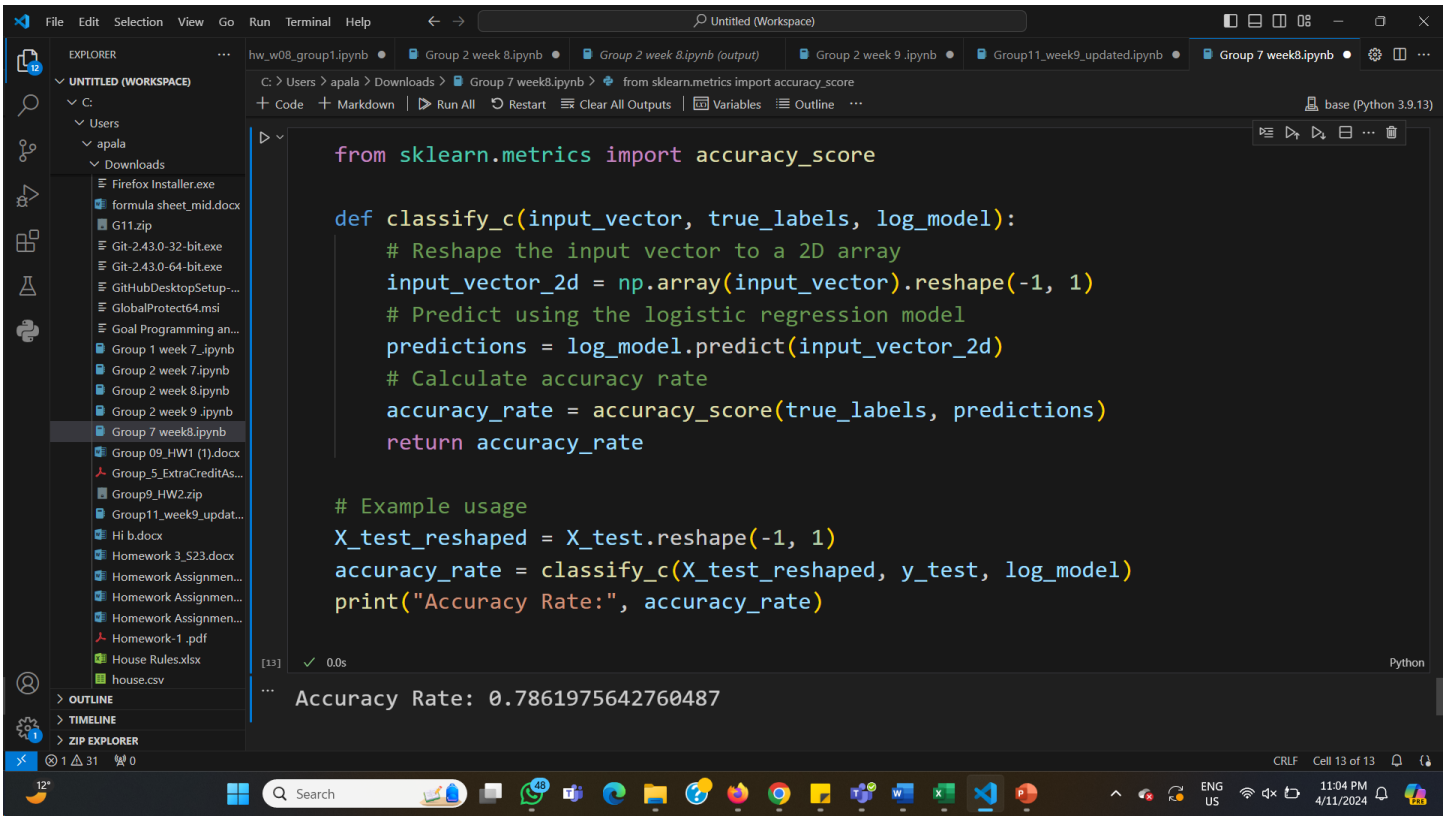
```
failure_mask = predictions != testCdataset['cb']

# Extract the failure cases
failure_cases = testCdataset[failure_mask]

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

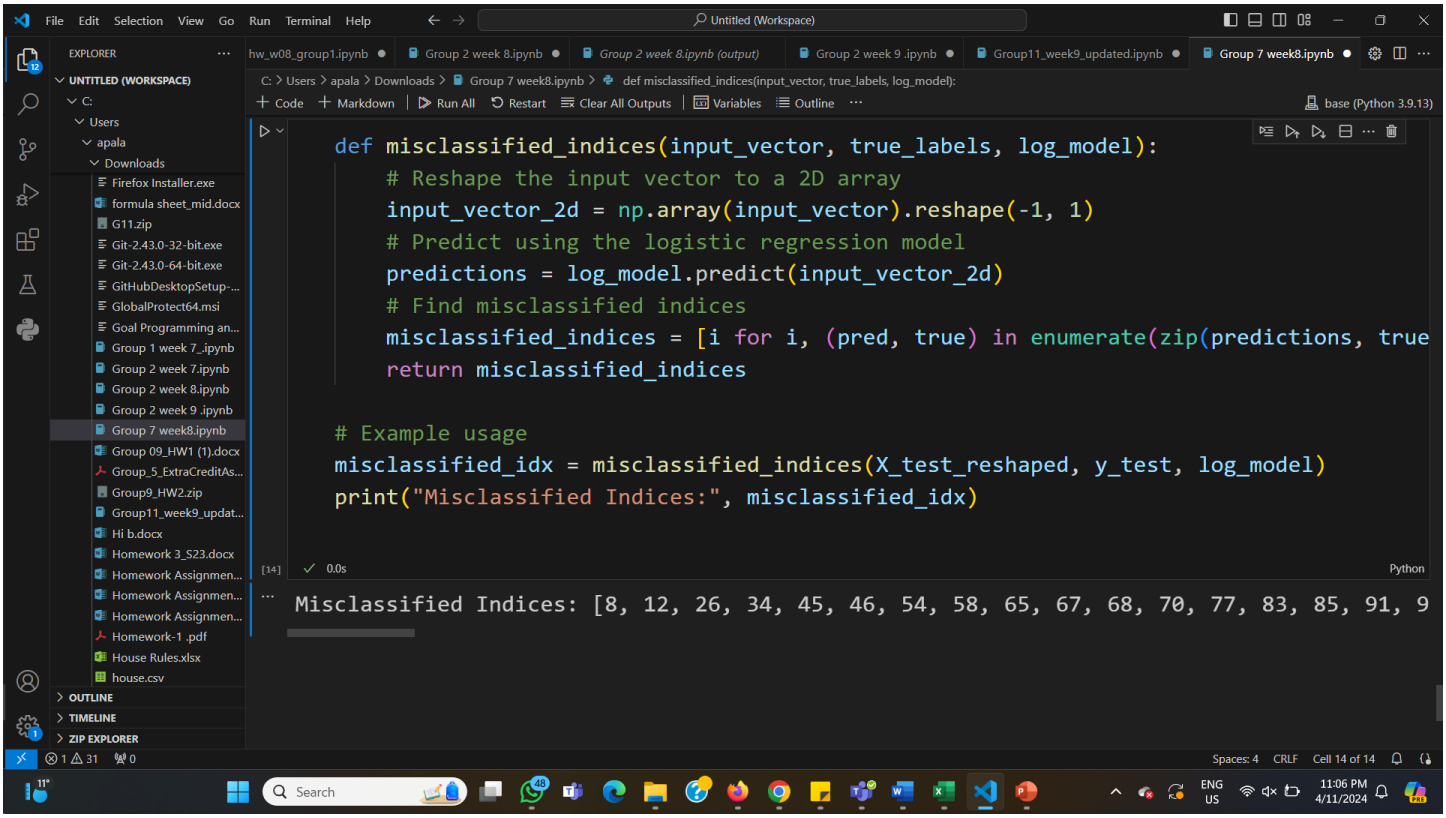
```
Failure Cases:
      cb       value
45    cw   71.733092
50    cw  182.833195
58    cb  203.622196
140   cw  127.185698
146   cw  183.215341
...   ..         ...
969   cw  103.753146
977   cw   99.769320
1004  cw  150.728038
1019  cb  246.398694
1025  cw   88.743998

[131 rows x 2 columns]
```

```python
# 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

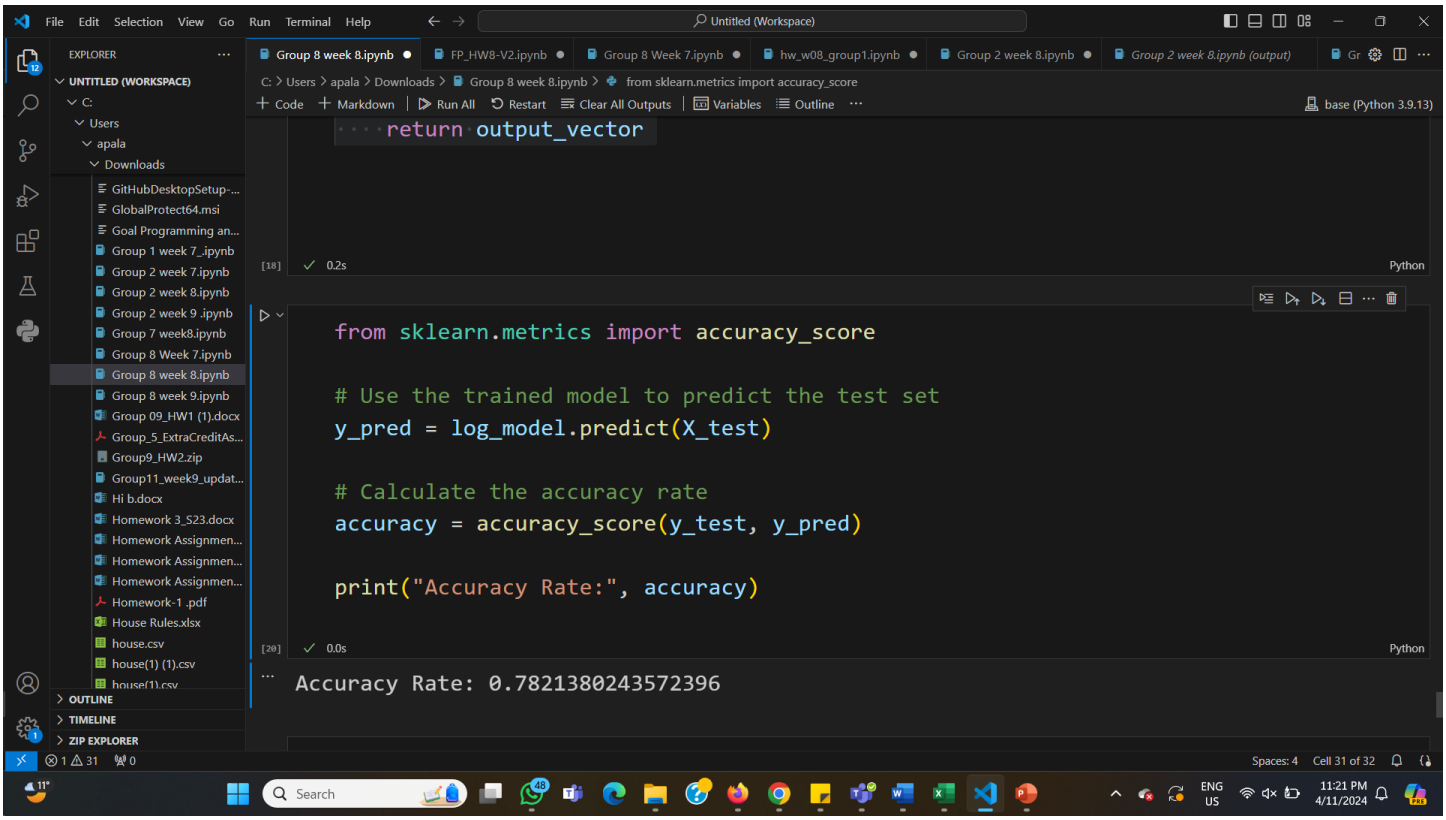


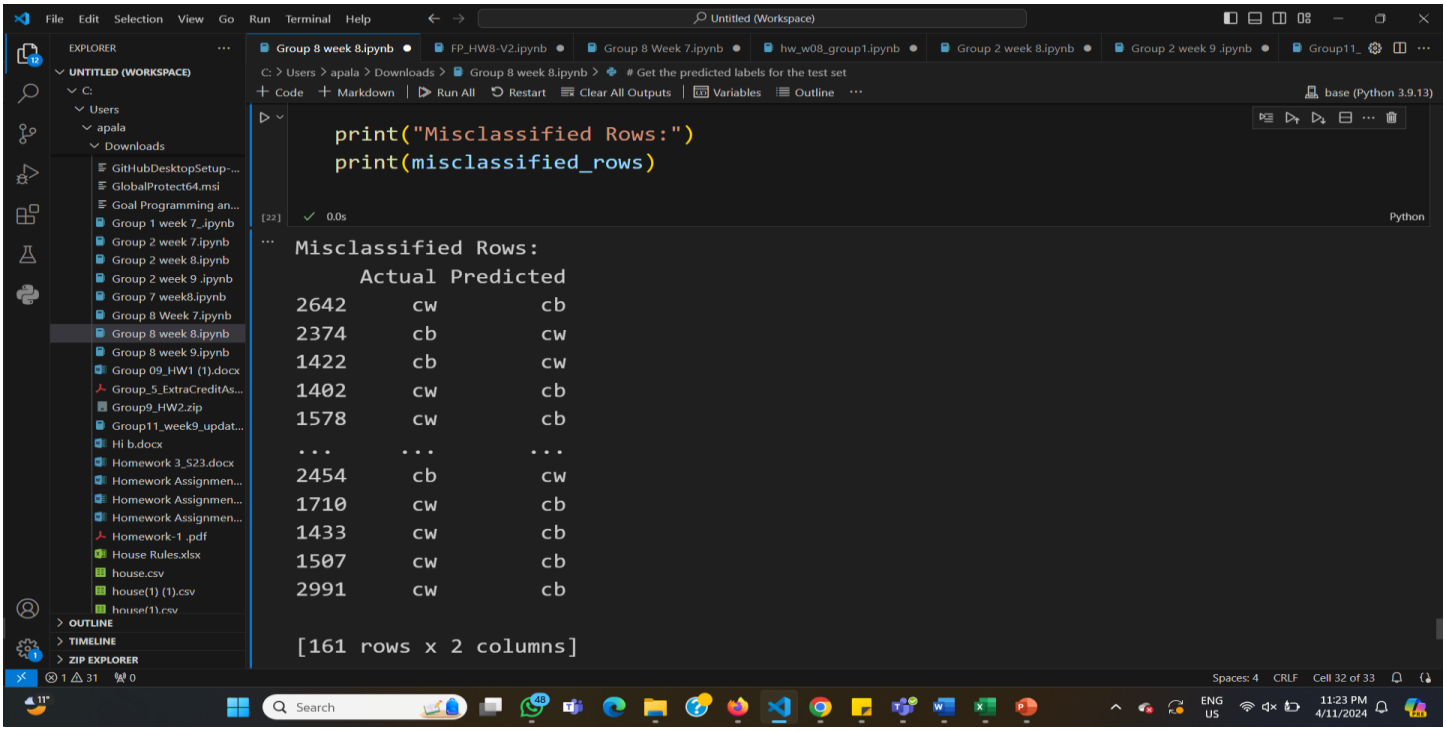


All rows that were misclassified:

[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





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

[160 rows x 3 columns]
Percentage of Incorrect Predictions: 21.650879566982407 %
```

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 8.ipyn

+ 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 ···

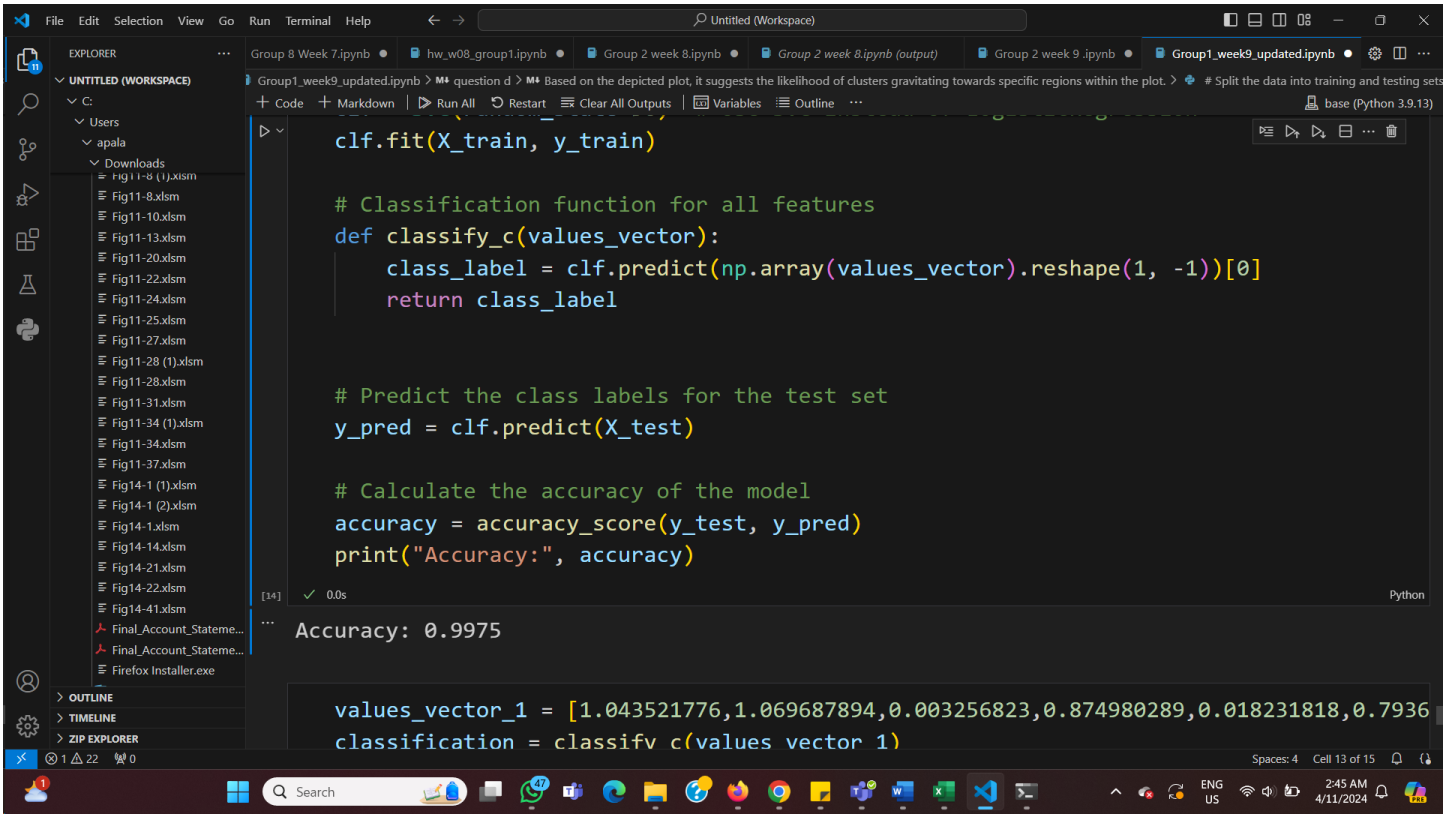
```
# Print the DataFrame of incorrect predictions
print("Incorrect Predictions:")
print(incorrect_predictions)

# 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 %
```

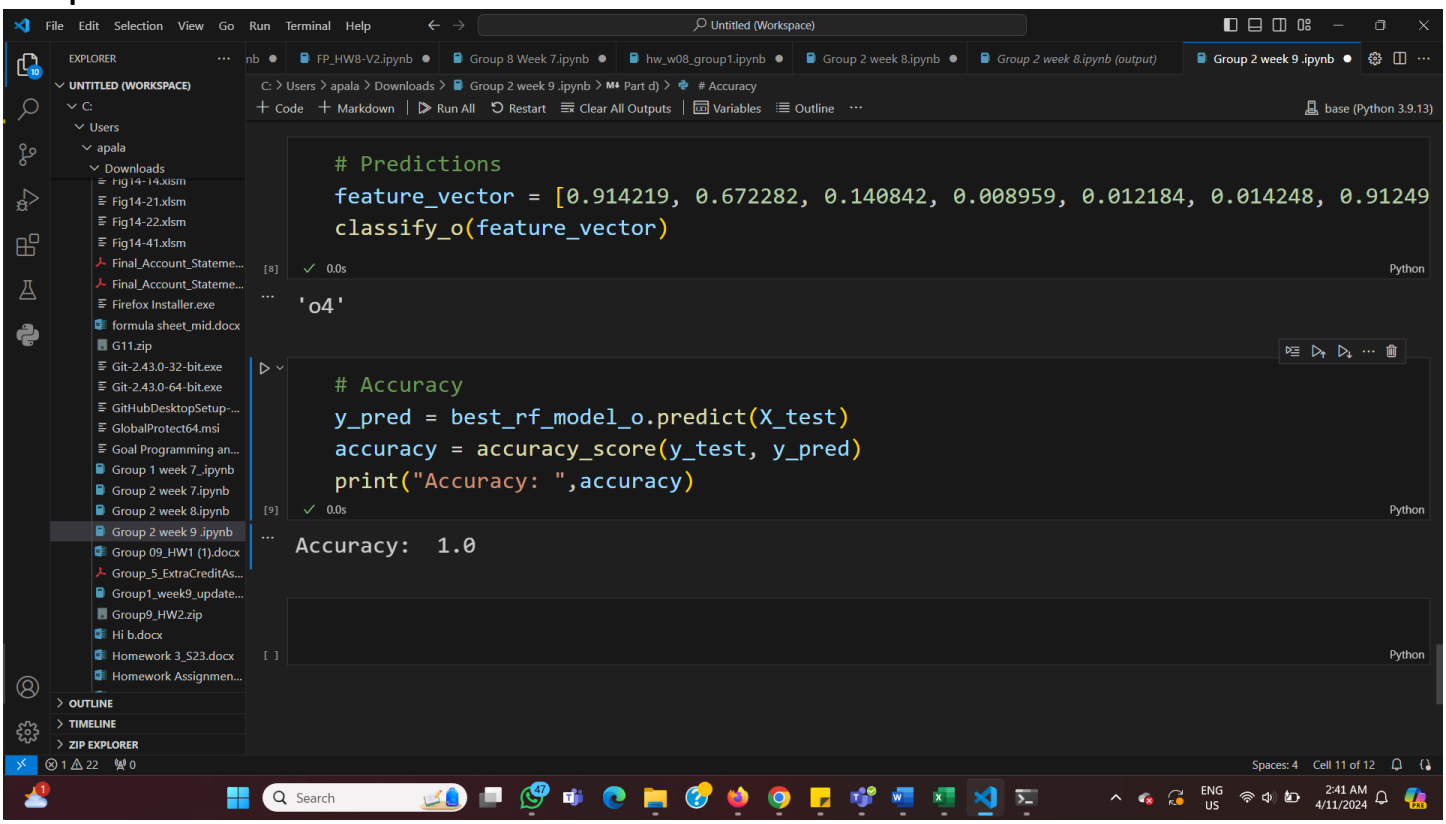
O Classifier

Group1



Group 2

Group 2 has no misclassification in o classifier



Group 3

```python
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming X_train, X_test, y_train, y_test are your feature and target variables
X = df[['a','b','c', 'd', 'e', 'f', 'g', 'h']].values
Y = df['class']
# Split the data 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 SVM classifier with a linear kernel
svm_classifier = svm.SVC(kernel='linear')

# Train the SVM classifier
svm_classifier.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print("SVM Parameters:")
print(svm_classifier.get_params())

```

```
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': '
```

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

# Extract misclassified rows using the misclassified indices
misclassified_rows = df.iloc[misclassified_indices]

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

```

```
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

```
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 wa
  warnings.warn(
```

```python
# Filter the DataFrame for misclassified rows
misclassified_rows = data[data['Cluster'] != data['Prediction']]

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

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

0

Group 5

```python
#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
```

```python
# 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
```

Group 7

```
[ ] # 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

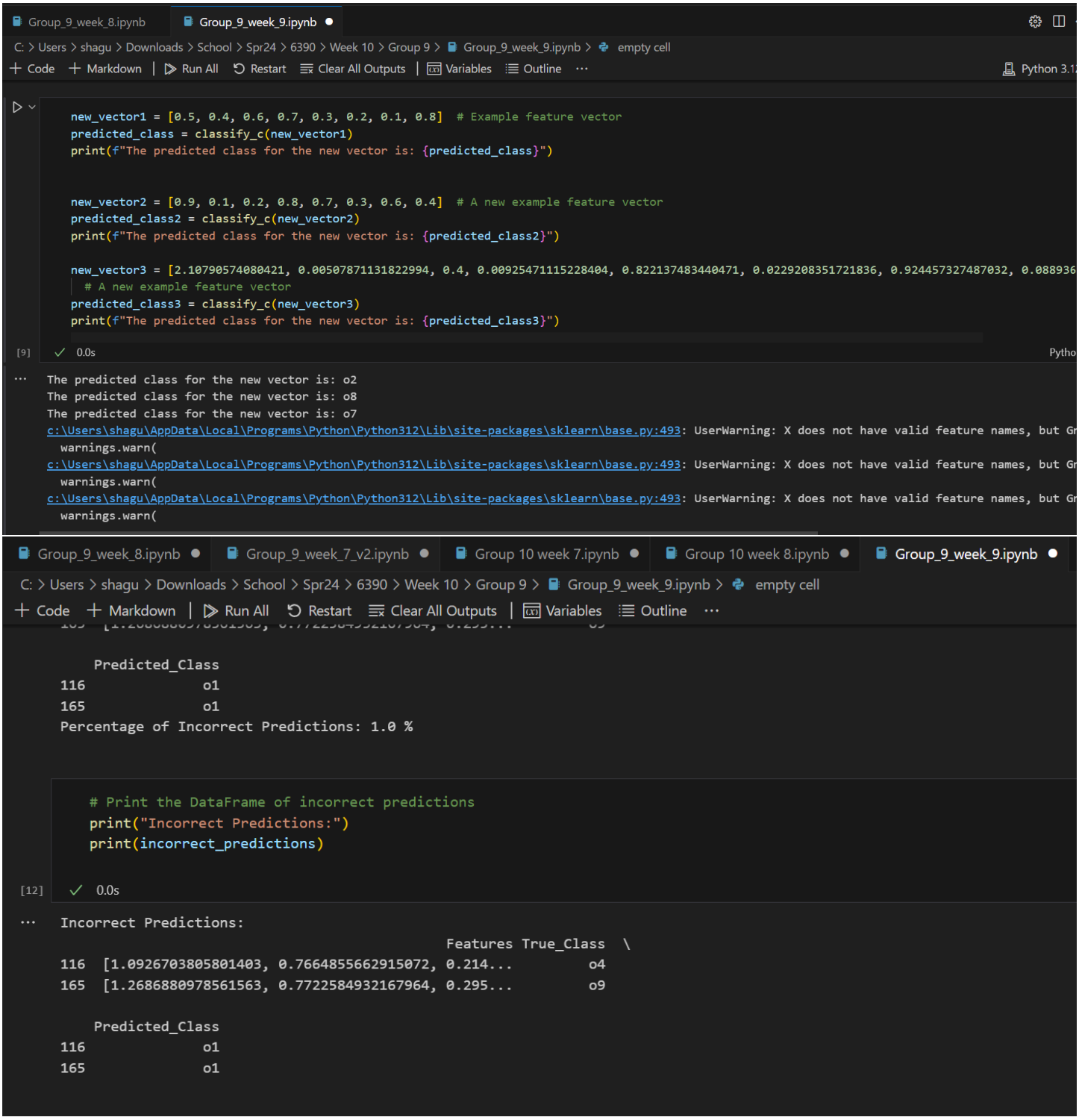
```
[6] # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Train an SVM model
    clf = SVC(random_state=42)  # Use SVC instead of LogisticRegression
    clf.fit(X_train, y_train)

    # Classification function for all features
    def classify_o(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: 0.995
```

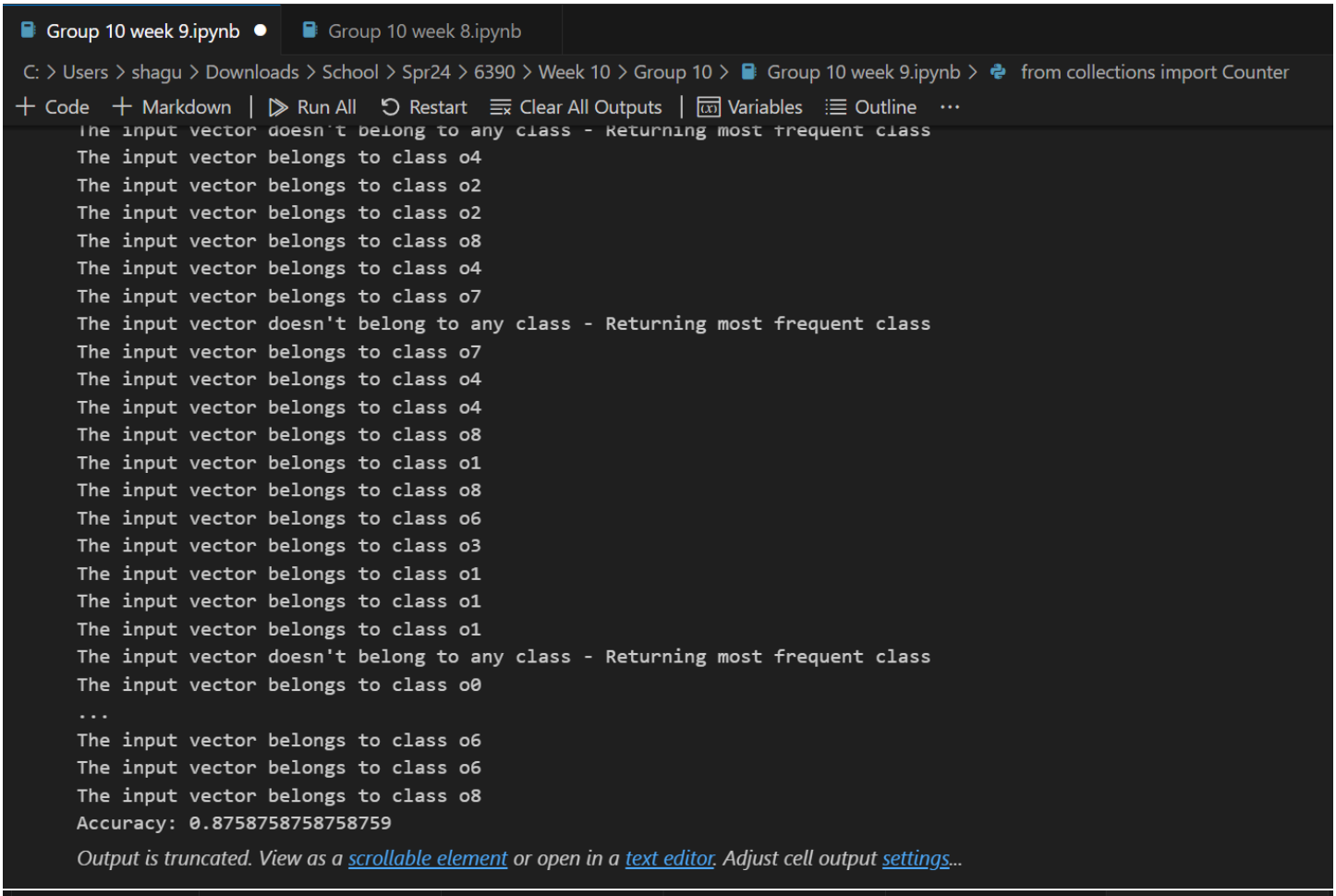
```
# Find misclassified rows
misclassified_indices = np.where(y_pred != y_test)[0]

# Extract misclassified rows using the misclassified indices
misclassified_rows = df.iloc[misclassified_indices]

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

```

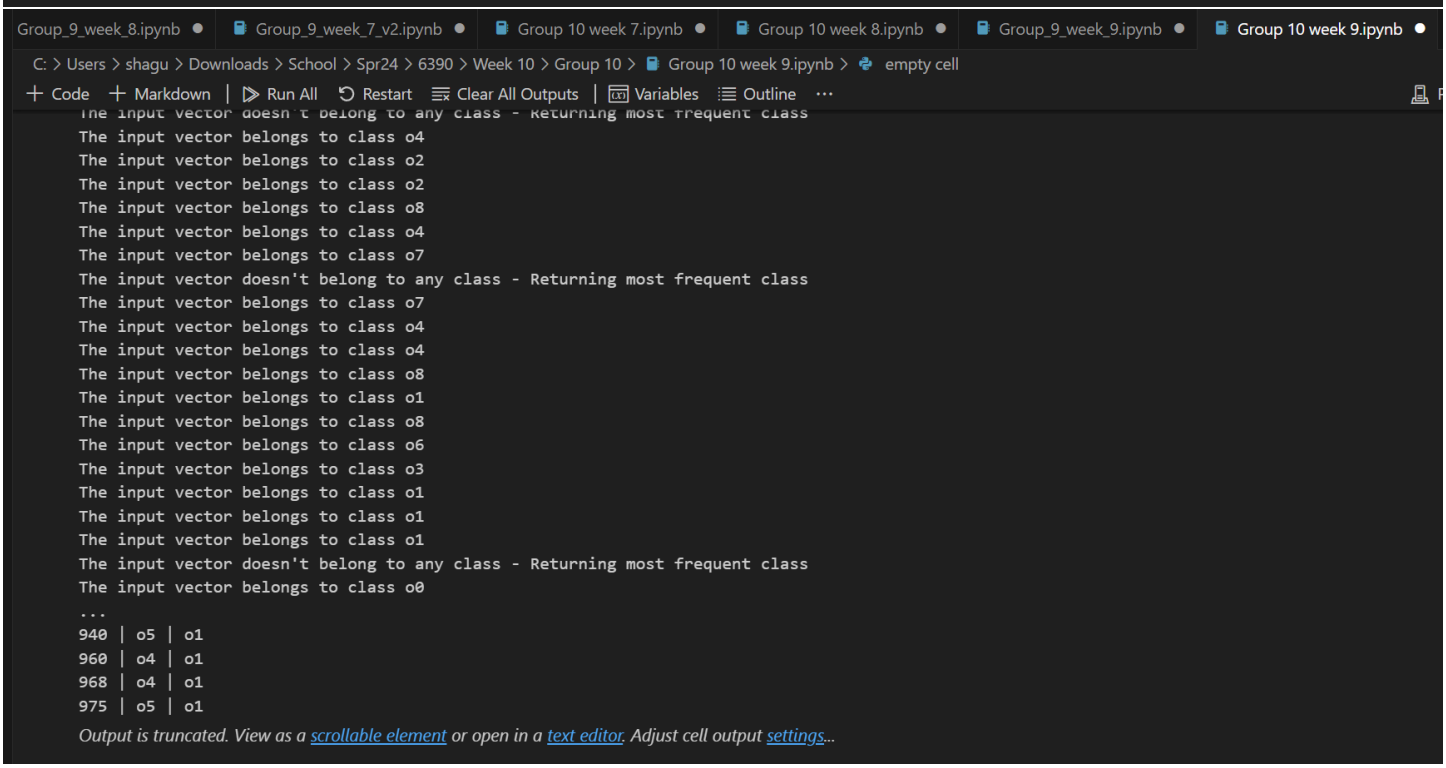
```
Misclassified Rows:
     Class  Feature1  Feature2  Feature3  Feature4  Feature5  Feature6  \
118     o7  1.977721  0.000187  0.014306  0.006106  1.126774  0.032714

     Feature7  Feature8
118  1.001159  0.048462
```

Group 9



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 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...