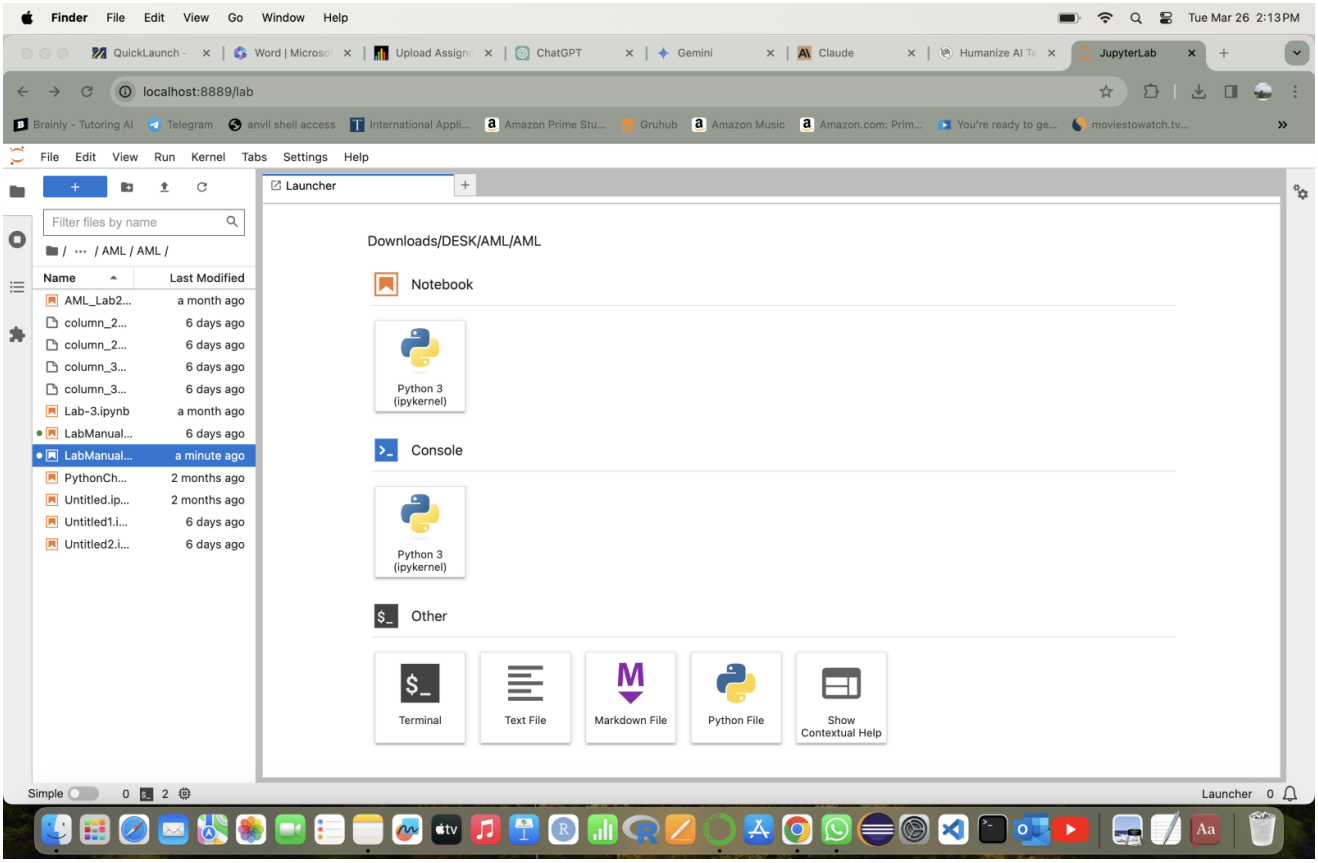
Jeevan Kumar Banoth - 02105145

Advanced machine learning

Homework – 6:

➢ Creating Jupyter Notebook with Anaconda navigator:



→ By using Anaconda navigator, first I opened Jupyter lab, and redirected it to my working folder.

→ In the left side of the notebook, we can see that there are files on which we have previously worked.

→ Created a notebook for this task by selecting File > New, Notebook, and then conda_python3 in the kernel dialog window.

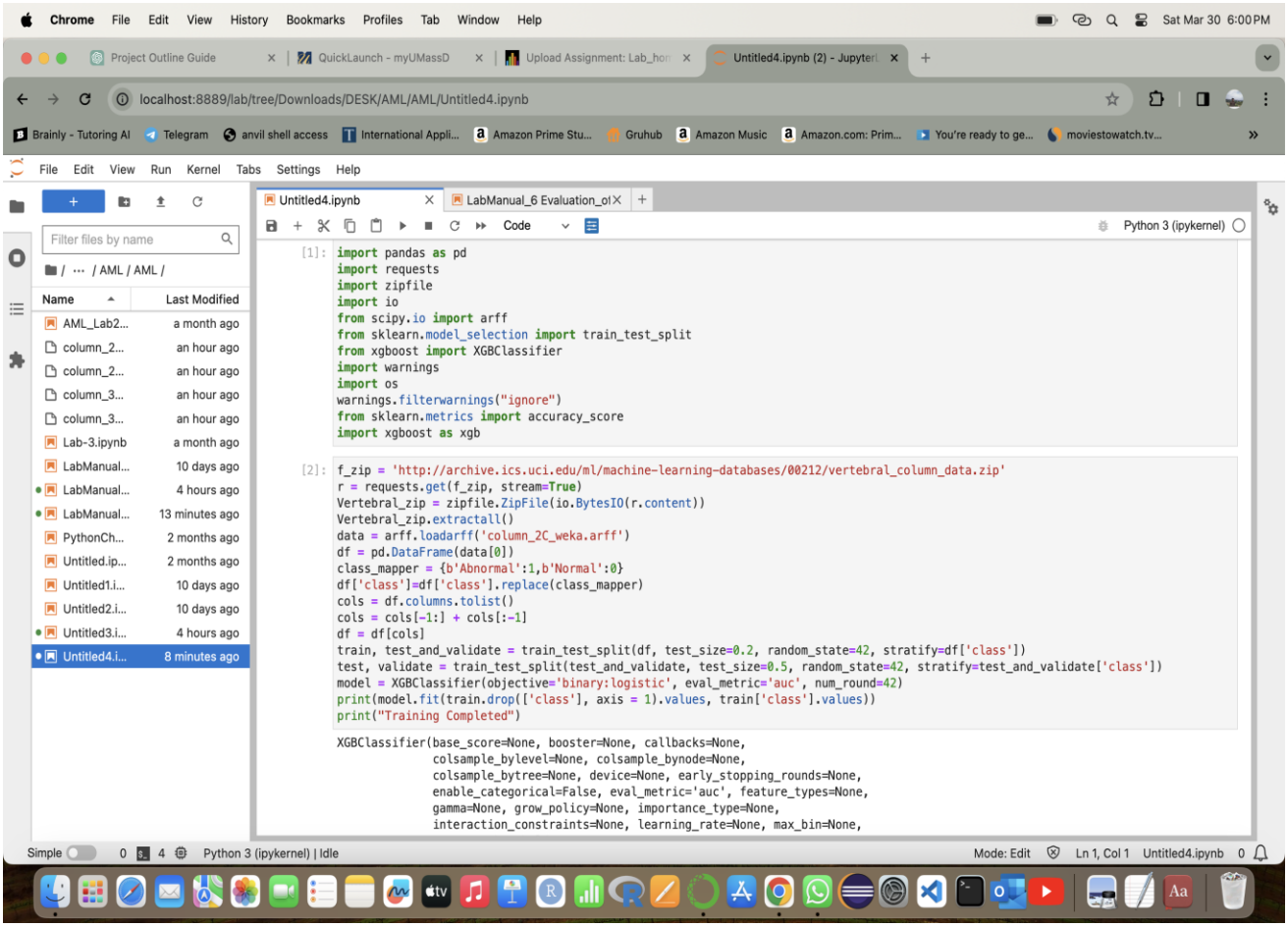
→ But, for this lab there is no need to work on the new notebook so we will start it by uploading.

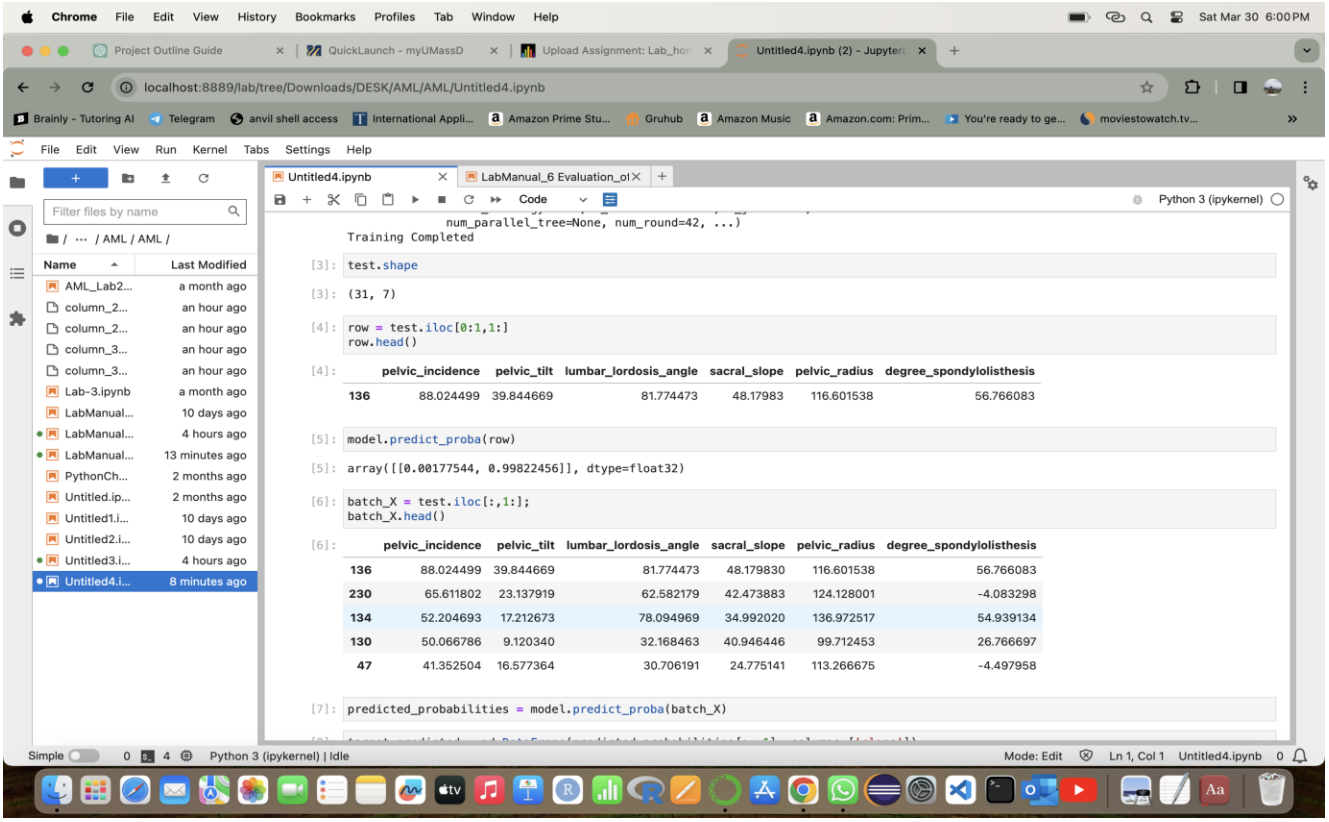
→ After opening the document, I created a new notebook by using python3 kernel and started doing the lab.

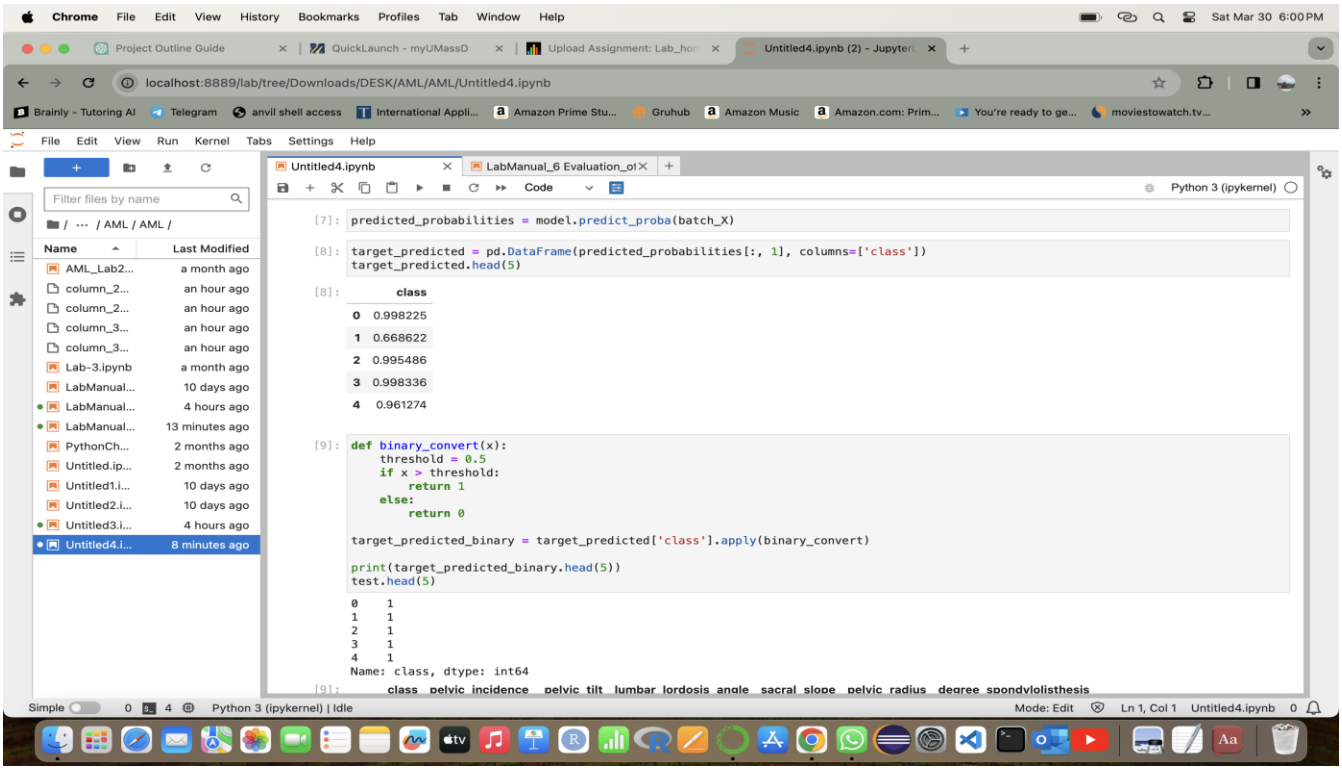
→ So, the first part of the lab is all same as the previous one, this lab is the continuation of that lab so we must perform the previous part and continue with this one.

→ Below are the Images of the code which has previous lab part.

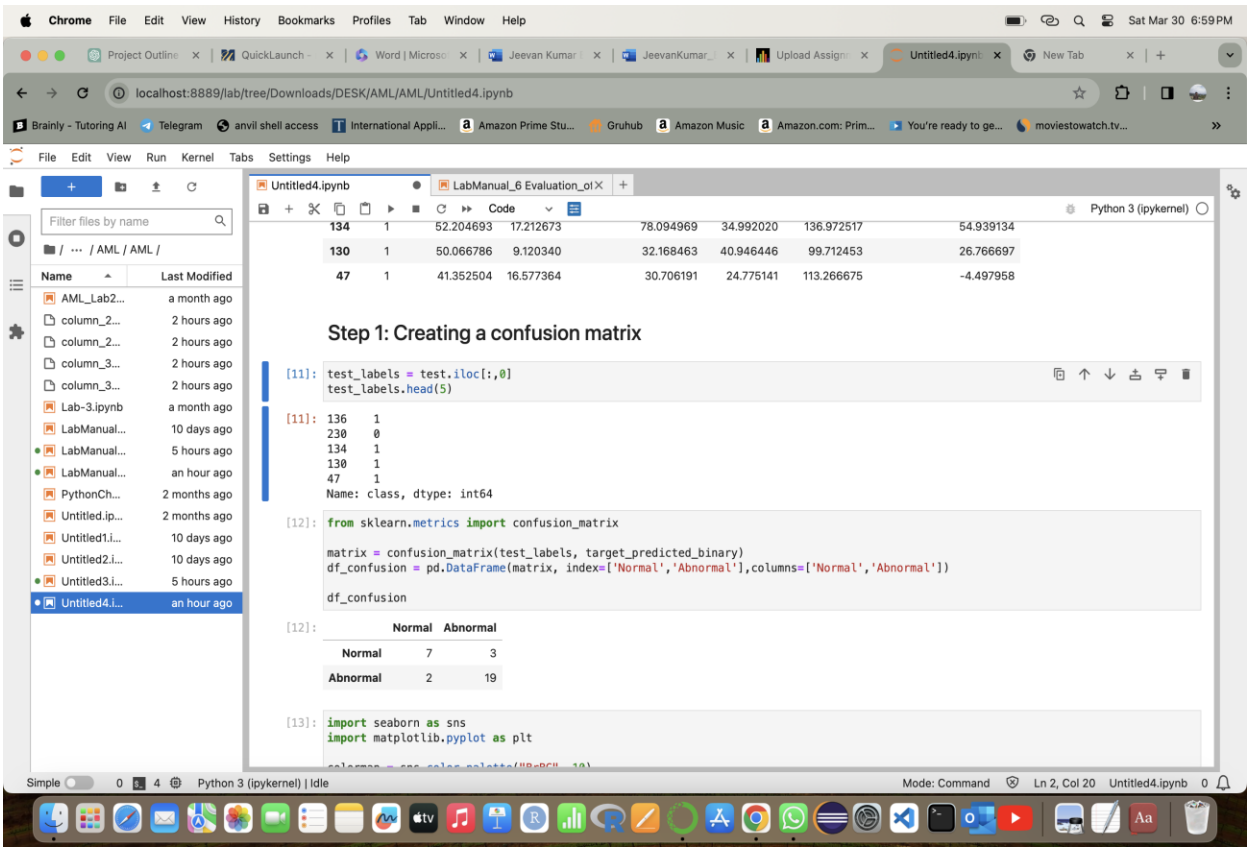
Note: As I have already explained the code in the previous lab, I am not specially explaining it now. I am just attaching those pictures for the proof that I performed the lab.







Step 1: Creating confusion matrix:



→ **test_labels = test.iloc[:, 0]:** Extracting Actual Labels: - The first line retrieves the true labels from the test data, assuming they're in column 0 of the test DataFrame.

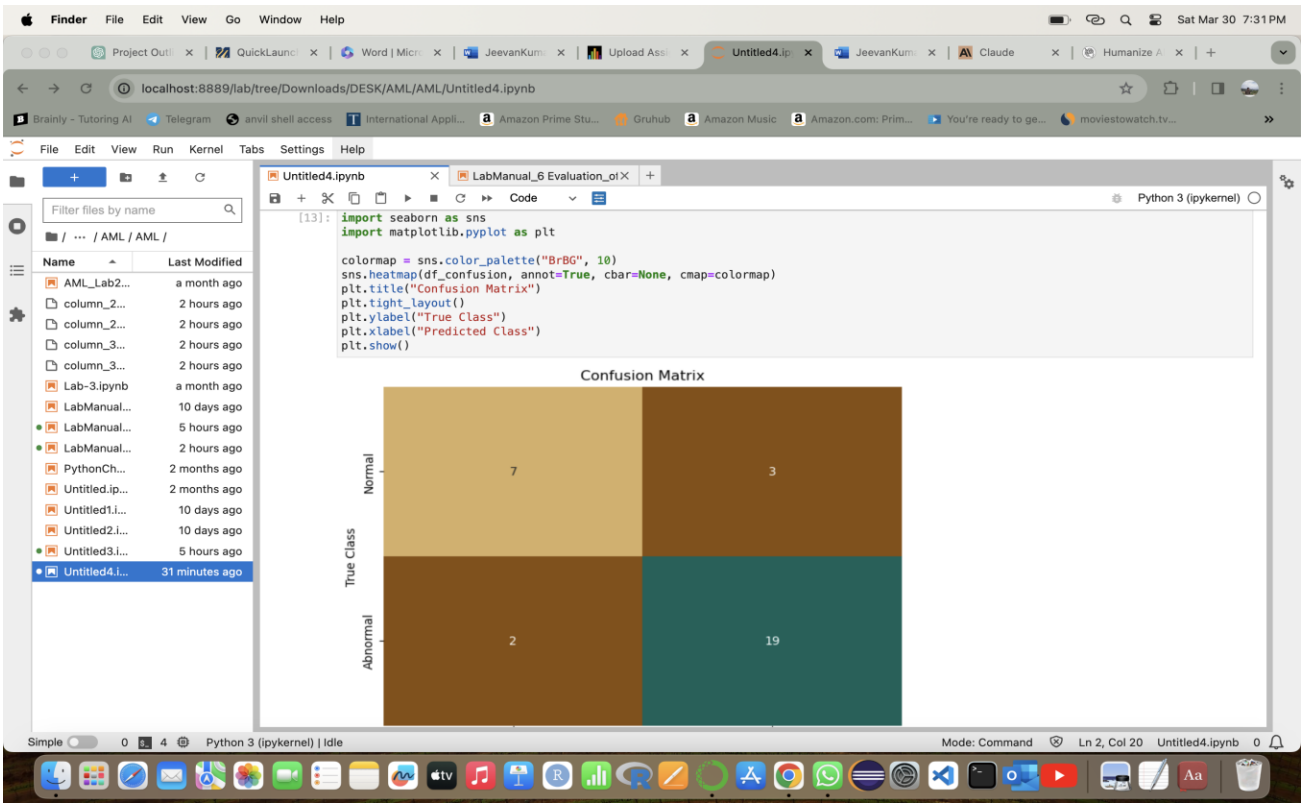
→ **test_labels.head(5):** Printing First Five Labels: - The second line displays the initial five entries of the label series for easy reference.

→ **from sklearn.metrics import confusion_matrix:** Importing Confusion Matrix Function: - The third line imports the function that evaluates the confusion matrix from the Python library.

→ **Calculating Confusion Matrix:** - The final line employs the imported function to create a confusion matrix, evaluating the results against the real labels and projected binary labels.

→ A NumPy array called "matrix" is used to make a Pandas DataFrame called "df_confusion."

→ The rows and columns of the DataFrame have labels like "Normal" and "Abnormal," which stand for the classes in your classification task. The "df_confusion" DataFrame is then printed to the console, showing a table of the confusion matrix.



→ **import seaborn as sns and import matplotlib.pyplot as plt:** To display data visually, this code uses Seaborn and Matplotlib libraries.

→ **colormap = sns.color_palette("BrBG", 10):** A color palette called "colormap" is created using the "BrBG" colormap from Seaborn, with 10 different colors.

→ Finally, a heatmap is generated using Seaborn's "heatmap" function, with the confusion matrix (df_confusion) as the data.

→ The values within the confusion matrix are displayed on the cells (annot=True), the color bar is hidden (cbar=None), and the "colormap" palette is used for coloring (cmap=colormap).

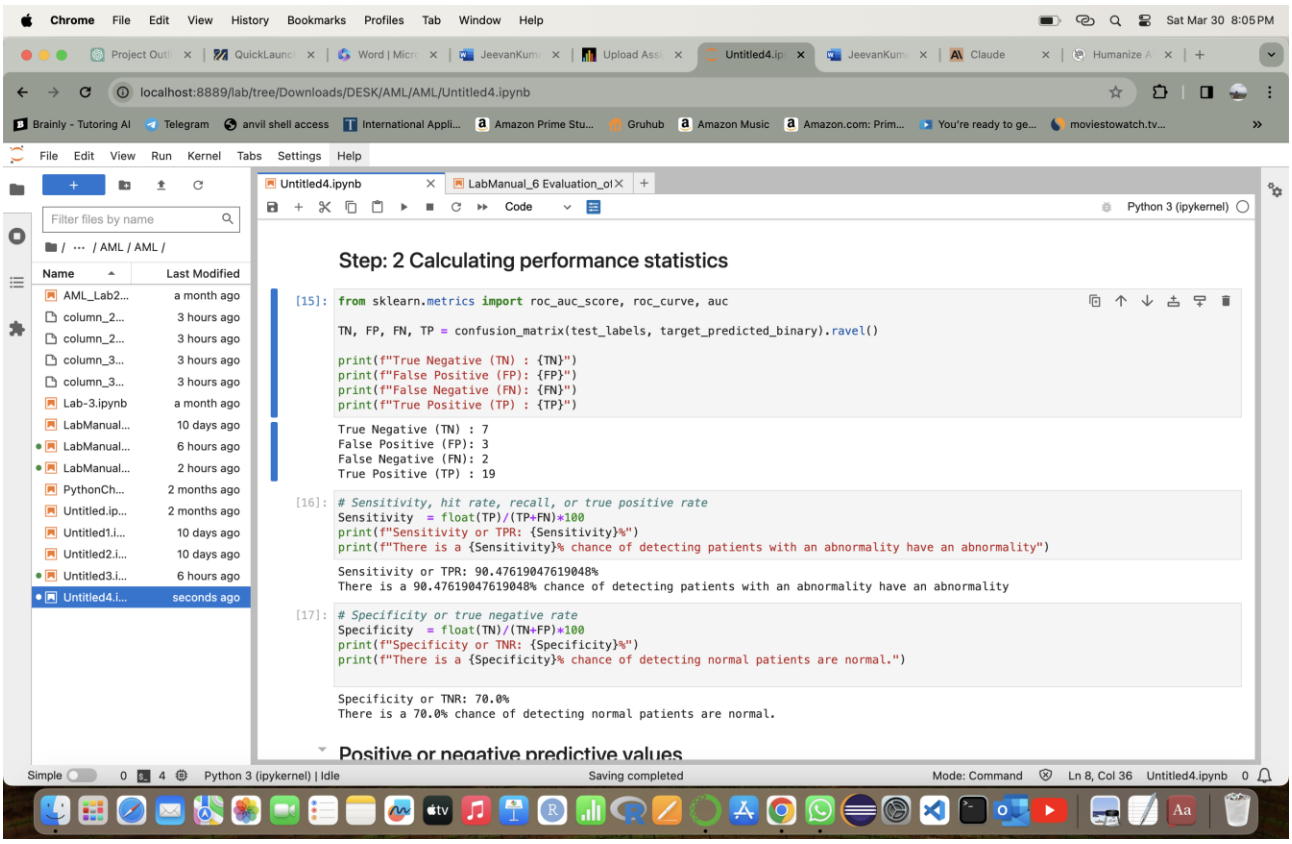
→ **plt.title("Confusion Matrix")":** Set the plot's title to "Confusion Matrix."

→ **"plt.tight_layout()":** Optimize the space around plot elements to avoid clipping labels or titles.

→ **"plt.ylabel("True Class")":** Label the y-axis with "True Class."

→ **"plt.xlabel("Predicted Class")"**: Label the x-axis with "Predicted Class."

→ **plt.show()** - This line displays the plot in a new window.



→ **from sklearn.metrics import roc_auc_score, roc_curve, auc:** The line imports three functions: - `roc_auc_score`: Calculates the Area Under the Receiver Operating Characteristic (AUC-ROC) curve. - `roc_curve`: Generates the ROC curve. - `auc`: Computes the AUC-ROC metric.

→ **TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel():** This line unpacks the values of TN, FP, FN, and TP from the confusion matrix, which compares actual and predicted binary class labels. The matrix is flattened into a one-dimensional array using `ravel()`.

→ The subsequent four lines display the values of TN, FP, FN, and TP.

→ Sensitivity = float (TP)/(TP+FN) *100 - This line calculates the sensitivity (also known as recall or true positive rate) by dividing the number of true positives by the sum of true positives and false negatives, and then multiplying by 100 to get the percentage.

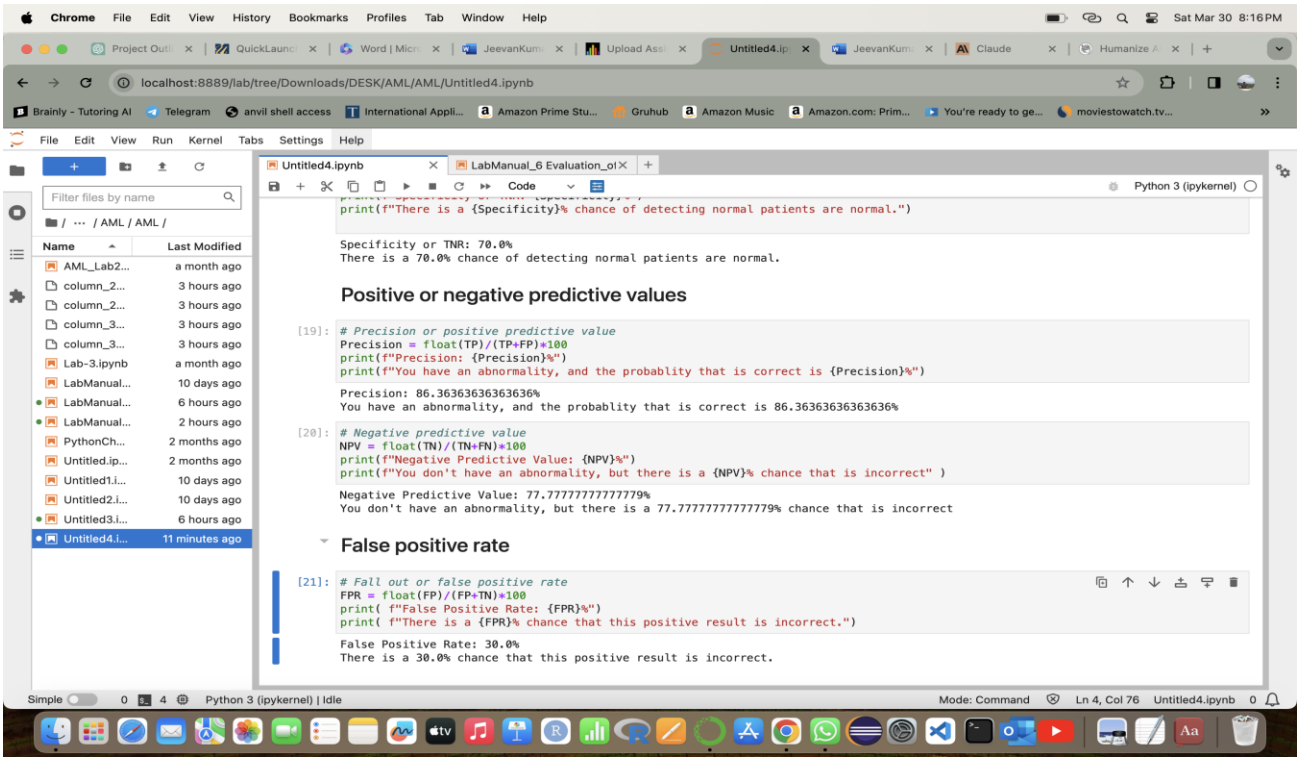
→ The next two lines print the value of sensitivity and provide an interpretation of what it means.

→ The code is calculating and printing important metrics related to the performance of a binary classification model. These metrics are derived from the confusion matrix, which is a table that summarizes the true and false predictions of the model.

→ Sensitivity measures the proportion of positives correctly identified by the model.

→ Specificity measures the proportion of negatives correctly identified by the model.

Positive or negative predictive values:



- → This code calculates more performance indicators for a classification model that distinguishes between two classes.
- → It uses data from a confusion matrix, which keeps track of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- → Precision, also called Positive Predictive Value (PPV), is one of these metrics. It shows how many of the predictions that said something was positive were right.
- → The formula is: Precision = TP / (TP + FP) * 100.
- → The code shows the precision value and says: "You have an abnormality, and the probability that is correct is {Precision}%".

NPV (Negative Predictive Value):

NPV measures how many predictions for "no abnormality" are correct.

Formula: NPV = (True Negatives / (True Negatives + False Negatives)) x 100

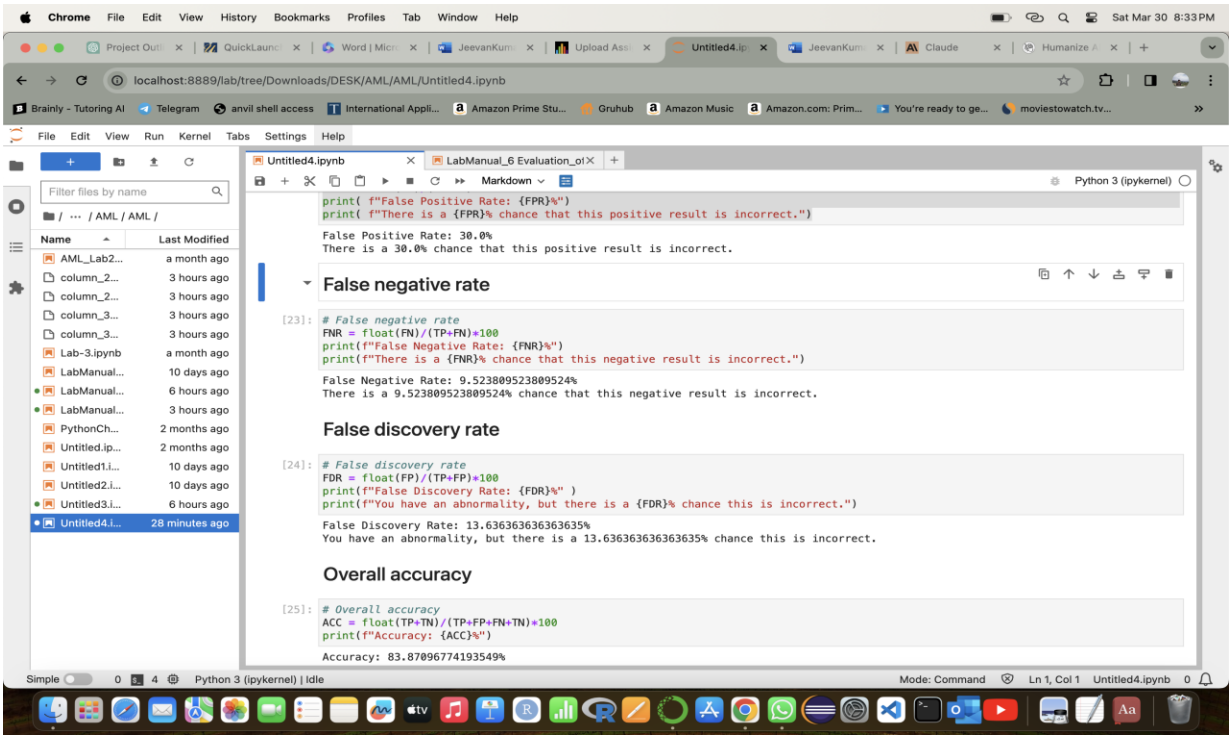
The code shows the NPV percentage and says, "You don't have an abnormality, but there's a {NPV}% chance that's wrong."

FPR (False Positive Rate):

FPR measures how many actual negative cases are accidentally labeled as positive.

Formula: FPR = (False Positives / (False Positives + True Negatives)) x 100

The code shows the FPR percentage and says, "There's a {FPR}% chance this positive result is wrong."

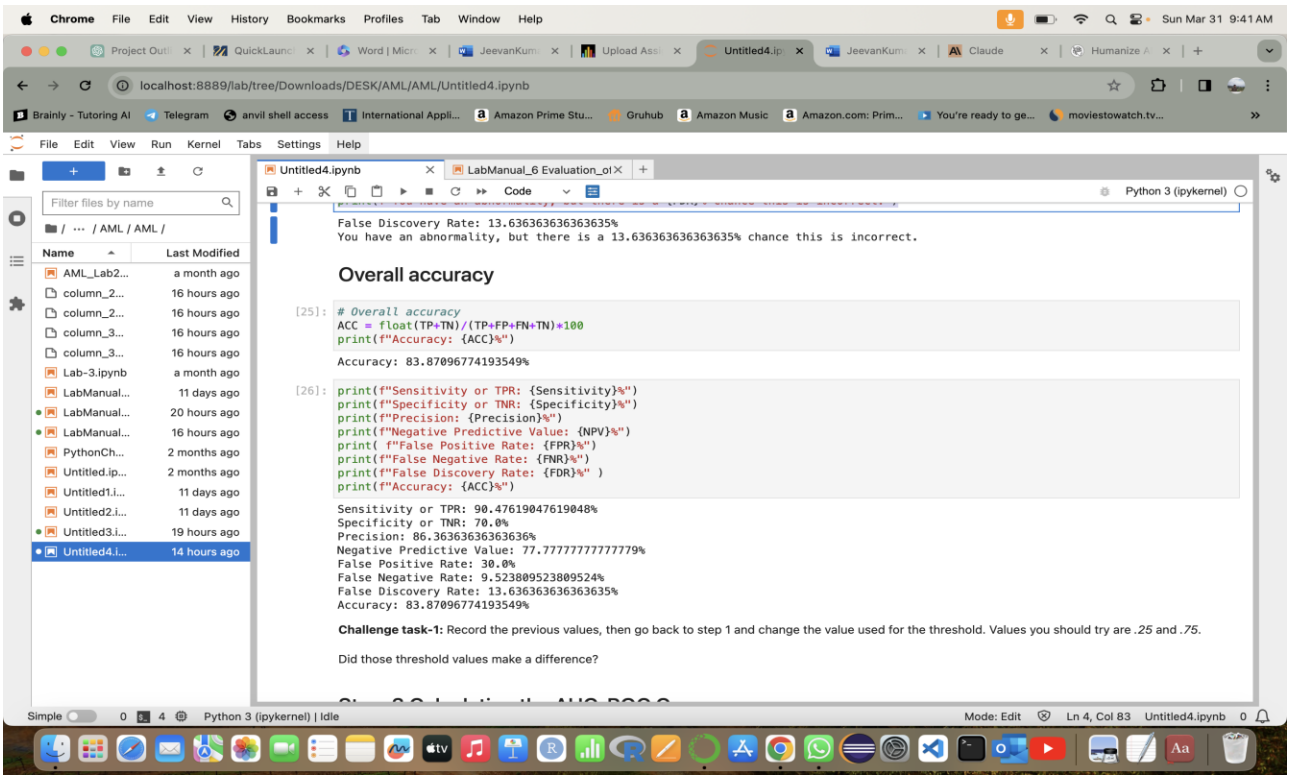


→ False Negative Rate quantifies the proportion of actual positives that are incorrectly classified as negatives, which is related to the model's sensitivity (or recall). A high FNR indicates the model is missing many positive instances.

→ False Discovery Rate quantifies the proportion of positive predictions that are false positives. It is related to the model's precision (or positive predictive value). A high FDR suggests that many of the positive predictions made by the model are incorrect.

→ **For example,** if the FNR is high, it may indicate that the model is struggling to identify positive instances accurately, and adjustments or additional training may be required to improve its sensitivity.

→ Conversely, if the FDR is high, it suggests that the model is producing too many false positive predictions, and steps may be needed to improve its precision.



Overall Accuracy:

→ The accuracy measure represents the proportion of correct predictions (both positive and negative) made by the model.

→ It is calculated as: ACC = (TP + TN) / (TP + FP + FN + TN) * 100

→ The code prints the accuracy value as a percentage.

→ These measures comprehensively assess a binary classification model's effectiveness, enabling you to identify its strengths and limitations. While accuracy provides a broad view of the model's performance, it may not fully capture its behavior, especially in imbalanced datasets.

→ To fully evaluate the model, it is essential to consider other metrics alongside accuracy. These metrics provide insights into the model's ability to correctly classify positive and negative samples and the trustworthiness of its predictions.

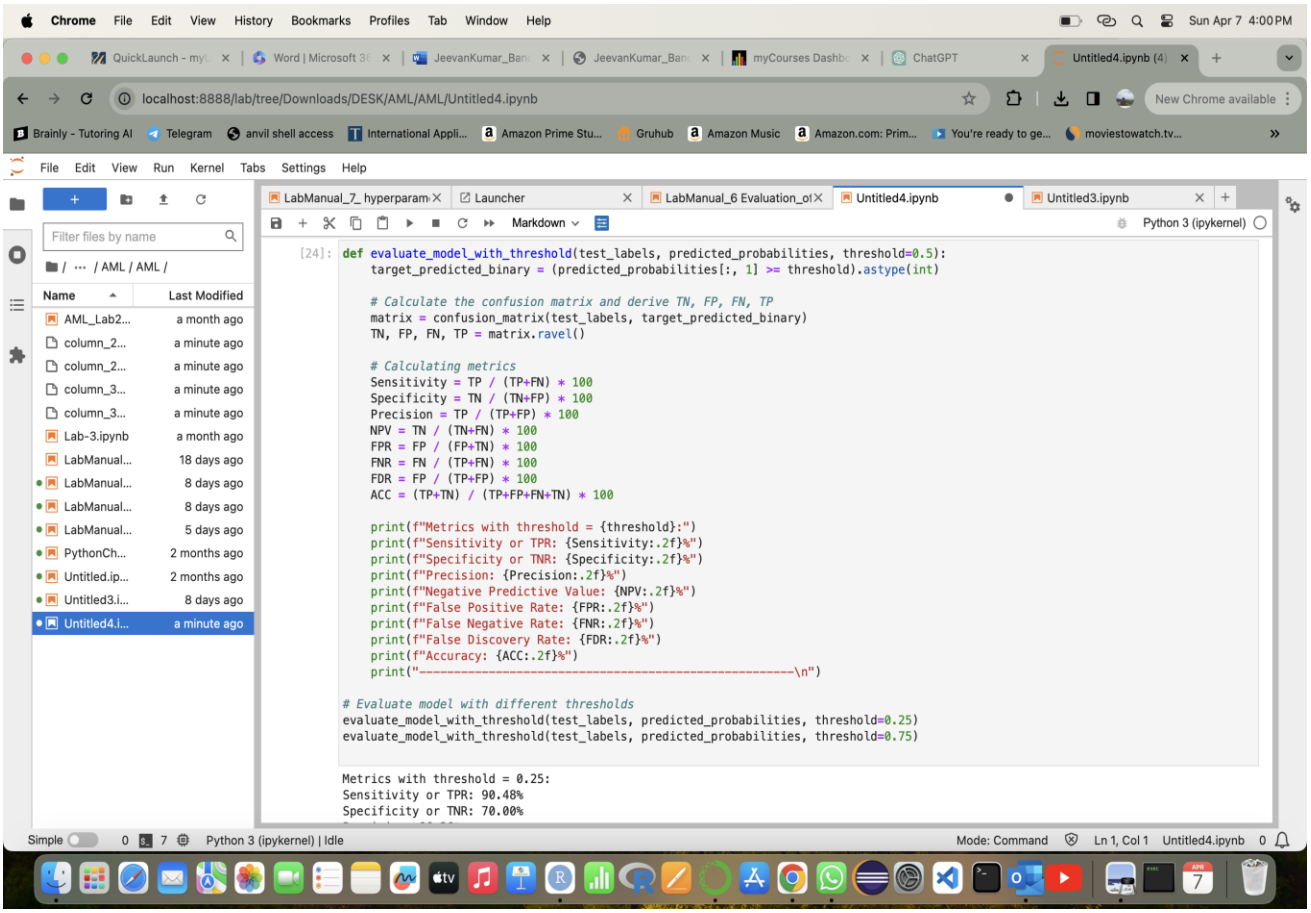
→ For example, a high sensitivity coupled with a low specificity may indicate that the model is good at detecting positive instances but also tends to misclassify many negative instances as positive (high false positive rate).

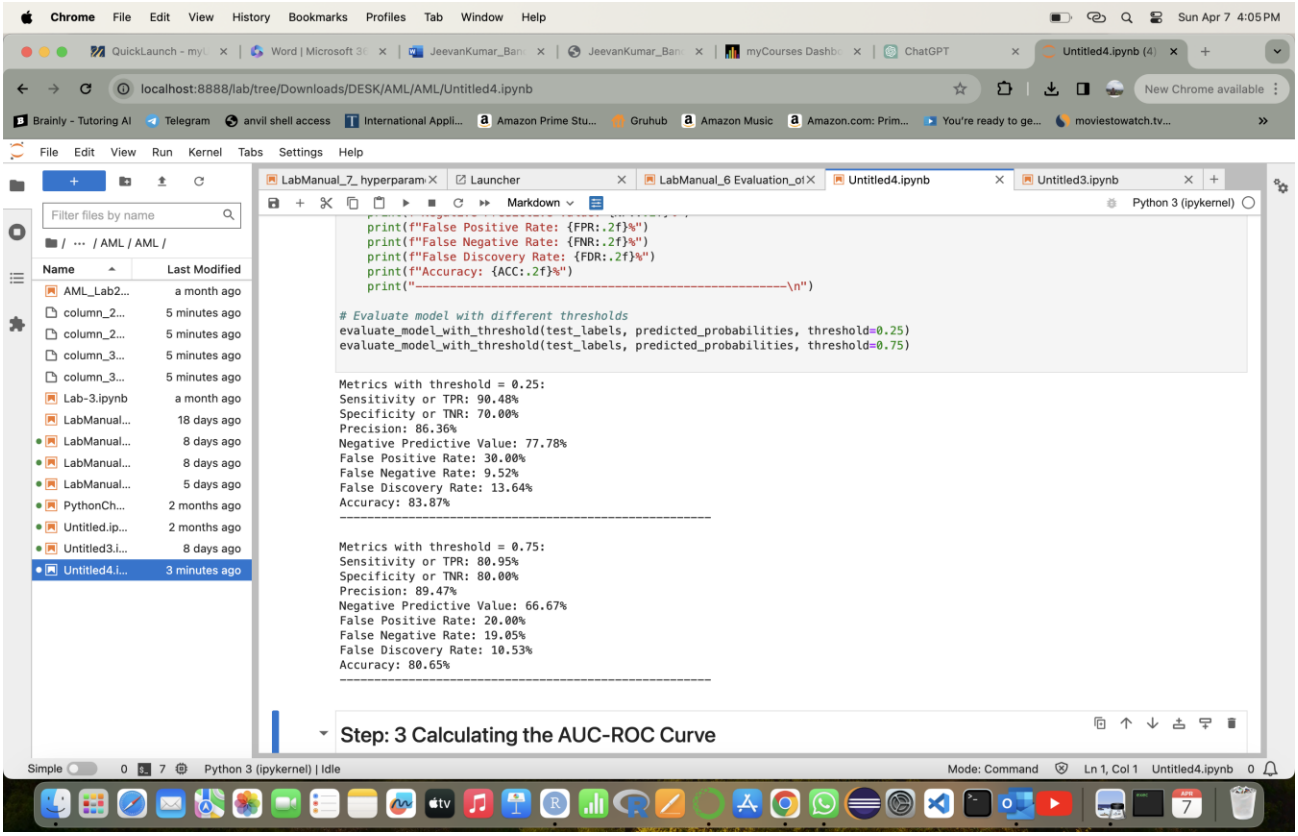
→ By analyzing this collection of metrics, you can identify potential areas for improvement, such as adjusting the model's decision threshold, gathering more training data, or exploring different modeling techniques or feature engineering approaches.

<u>Challenge task 1:</u> Record the previous values, then go back to step 1 and change the value used for the threshold. The values you should try are *.25* and *.75*.

Did those threshold values make a difference?

Ans:





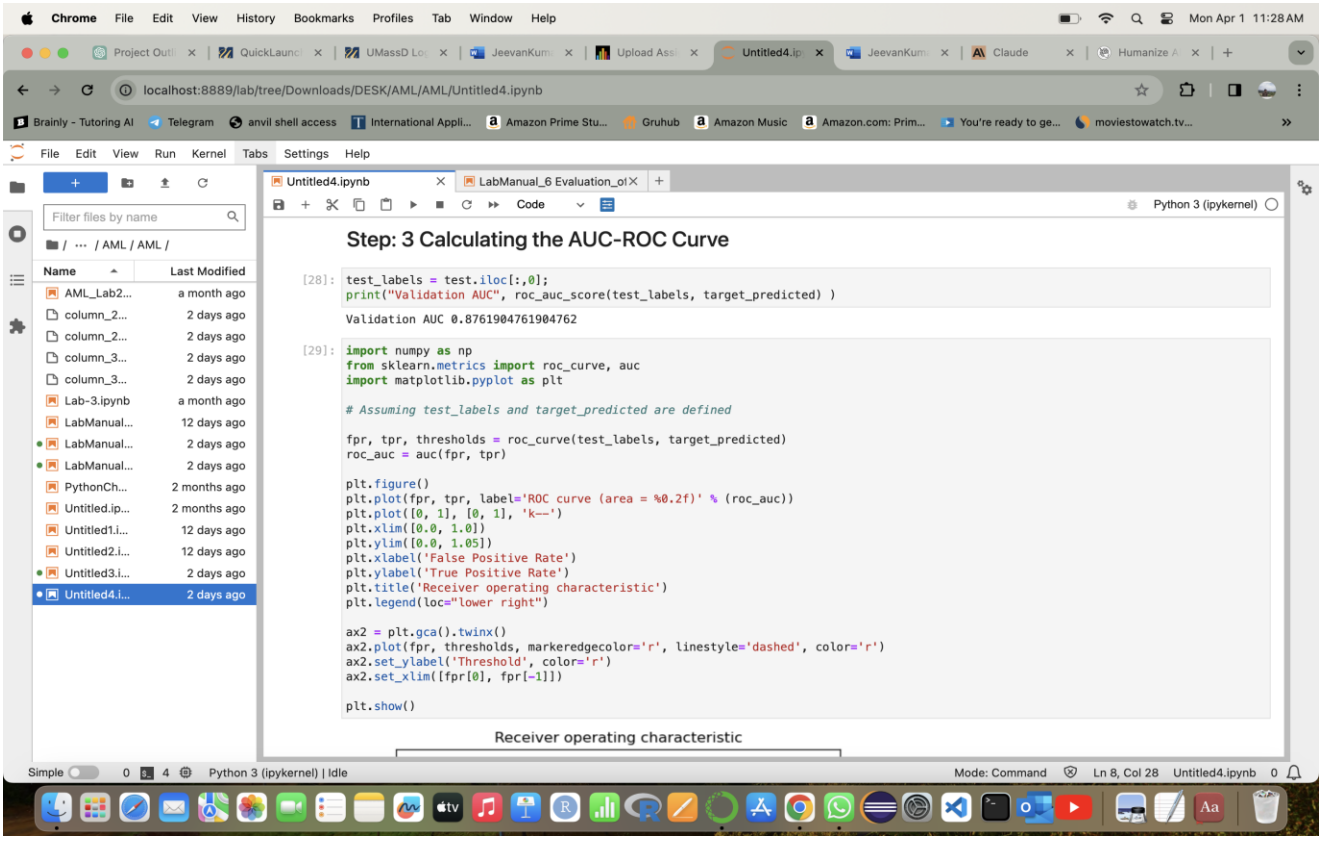
If the threshold is changed to 0.25 (lower threshold):

- The True Positive Rate (TPR) or Sensitivity would likely increase, as more instances would be classified as positive (abnormal).
- The False Positive Rate (FPR) would also increase, as more negative instances (normal) would be incorrectly classified as positive (abnormal).
- The Precision or Positive Predictive Value (PPV) would likely decrease, as the number of False Positives increases.
- The overall Accuracy might decrease or increase, depending on the distribution of classes and the impact of increased False Positives and True Positives.

If the threshold is changed to 0.75 (higher threshold):

- The True Positive Rate (TPR) or Sensitivity would likely decrease, as fewer instances would be classified as positive (abnormal).
- The False Positive Rate (FPR) would decrease, as fewer negative instances (normal) would be incorrectly classified as positive (abnormal).
- The Precision or Positive Predictive Value (PPV) would likely increase, as the number of False Positives decreases.
- The overall Accuracy might decrease or increase, depending on the distribution of classes and the impact of decreased False Positives and True Positives.

In general, adjusting the threshold value allows you to tradeoff between different performance metrics. A lower threshold favors higher recall (or TPR) at the cost of lower precision, while a higher threshold favors higher precision at the cost of lower recall.



This code is used to visualize the Receiver Operating Characteristic (ROC) curve and calculate the Area Under the ROC Curve (AUC-ROC) for a binary classification model.

- → print ("Validation AUC", roc_auc_score(test_labels, target_predicted)) - This line calculates and prints the AUC-ROC score using the roc_auc_score function from sklearn.metrics. The AUC-ROC score is a single value that summarizes the model's performance across all possible classification thresholds.
- → fpr, tpr, thresholds = roc_curve(test_labels, target_predicted) - This line computes the false positive rate (fpr), true positive rate (tpr), and classification thresholds (thresholds) for the model using the roc_curve function from sklearn.metrics.
- → roc_auc = auc(fpr, tpr) - This line calculates the AUC-ROC value using the auc function from sklearn.metrics and the previously computed fpr and tpr values.
- → The following lines create a new figure and plot the ROC curve using matplotlib.pyplot. The ROC curve is plotted by plotting the tpr (y-axis) against the fpr (x-axis). The AUC-ROC value is also displayed in the plot legend.

→ plt.plot([0, 1], [0, 1], 'k--') - This line plots a diagonal line from (0, 0) to (1, 1), which represents the performance of a random classifier.

→ The code sets the x and y limits of the plot, adds labels to the x and y axes, and sets a title for the plot.

→ ax2 = plt.gca().twinx() - This line creates a secondary y-axis on the same plot.

→ ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r') - This line plots the classification thresholds on the secondary y-axis using a red dashed line.

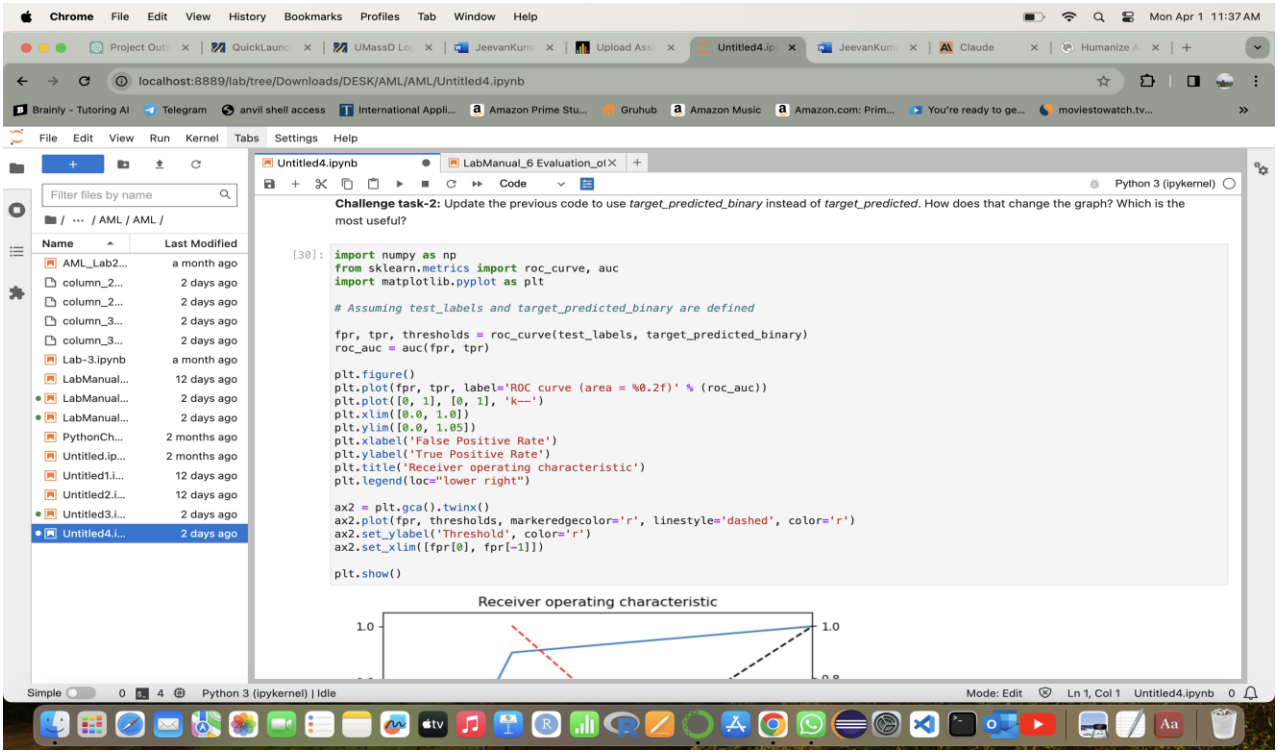
→ The code sets the y-axis label for the secondary y-axis as "Threshold" and adjusts the x-axis limits for the secondary y-axis.

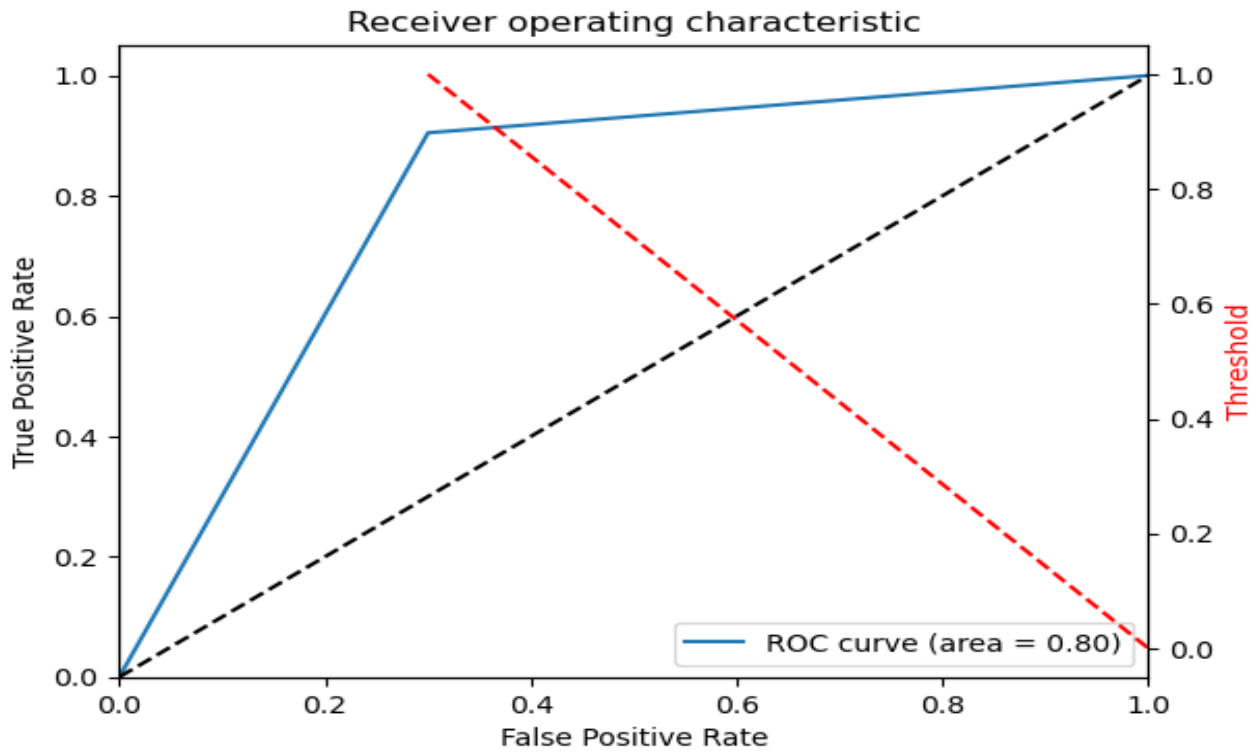
→ plt.show() - This line displays the plot.

Challenge task 2: Update the previous code to use *target_predicted_binary* instead of *target_predicted*. How does that change the graph? Which is the most useful?

Ans:

Below is the updated code as instructed in the challenging task. And you can see the output of it as well.





The main difference between using target_predicted and target_predicted_binary is that target_predicted likely contains the predicted probabilities or scores for each instance, while target_predicted_binary contains the binary predictions (0 or 1) based on a specific threshold.

When using target_predicted_binary, the ROC curve will be a step function, with each step representing a change in the classification threshold. This is because there are a limited number of unique thresholds that can be used to convert the predicted probabilities or scores into binary predictions.

On the other hand, when using target_predicted (the predicted probabilities or scores), the ROC curve will be a smooth curve, as there are many possible thresholds that can be used to convert the probabilities or scores into binary predictions.

→ The ROC curve generated using target_predicted is more useful for analyzing the model's performance across all possible thresholds and for comparing the performance of different models or algorithms. It provides a more detailed view of the trade-off between true positive rate and false positive rate at different thresholds.

→ The ROC curve generated using target_predicted_binary is more useful for understanding the performance of the model at the specific threshold used to generate the binary predictions. It can help identify the operating point of the model (the specific true positive rate and false positive rate) at that threshold.

Finally, the choice of which ROC curve to use depends on the specific goals and requirements of the analysis, as well as the availability of the predicted probabilities or scores (target_predicted) or the binary predictions (target_predicted_binary).

Conclusion:

- This code thoroughly assesses and graphically displays the performance of a binary classification model.
- It calculates numerous performance metrics, such as sensitivity, specificity, and overall accuracy, using confusion matrix values.
- The code also emphasizes the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC-ROC) metric.
- It computes the AUC-ROC score using sklearn.metrics' roc_auc_score function, which provides a summary measure of the model's performance across all possible classification thresholds.
- The code creates a graph known as an ROC curve. This graph helps visualize how well the model performs in classifying data.
- It shows the true positive rate (TPR) and false positive rate (FPR) at different classification thresholds.
- Two versions of the curve are generated: one based on predicted probabilities, and one based on binary predictions.
- The first curve is smooth, while the second is a step function. The visualization includes a second y-axis that shows the classification threshold for each point on the curve.
- This information is helpful for choosing an appropriate threshold, which balances the TPR and FPR based on your specific needs.
- The code then focuses on the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC-ROC).
- It demonstrates the generation of two ROC curves: one using the predicted probabilities and another using the binary predictions. The former provides a smooth curve, while the latter is a step function.
- This approach enables a thorough understanding of the model's strengths and weaknesses, facilitating informed decision-making and potential model optimization strategies based on the specific requirements and priorities of the problem, such as prioritizing sensitivity over specificity or vice versa, or optimizing precision or negative predictive value based on the consequences of false positives or false negatives.
- A wide range of performance metrics are calculated, including sensitivity (recall or true positive rate), specificity (true negative rate), precision (positive predictive value), negative predictive value, false positive rate, false negative rate, false discovery rate, and overall accuracy.

- These metrics quantify different aspects of the model's performance and help identify potential areas for improvement or areas where the model excels.
- Both curves visualize the trade-off between true positive rate and false positive rate at different classification thresholds.
- Overall, the code offers a comprehensive toolkit for evaluating and visualizing the performance of a binary classification model.
- It enables a thorough understanding of the model's strengths and weaknesses, facilitating informed decision-making and potential model optimization strategies.
- By combining the confusion matrix, performance metrics, and ROC curve analysis, the code offers a comprehensive toolkit for evaluating and visualizing the performance of a binary classification model.
- This approach enables a thorough understanding of the model's behavior, facilitating informed decision-making and potential model optimization strategies.

-------X------