**TASK 3 REPORT**

**1. Dataset Preparation**

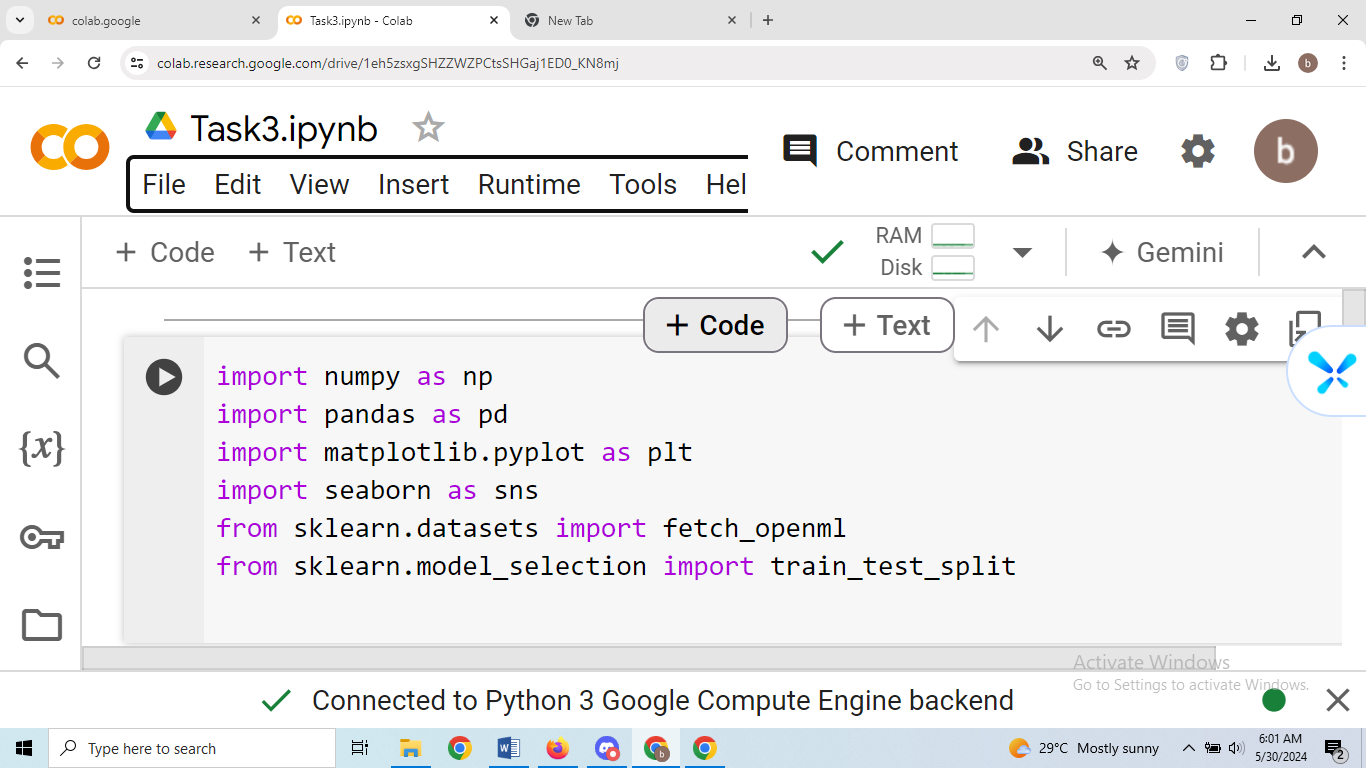
a. Split the MNIST dataset into training and testing sets. Use scikit-learn functions for splitting or write your own function. Ensure your code is clearly commented to explain your actions.

Condition 1: If your name is john doe, use john\_train and doe\_test as your variable names.

Condition 2: if you are using any random\_state in you code, the value for the random\_state = [Last two digits of your student ID**]**

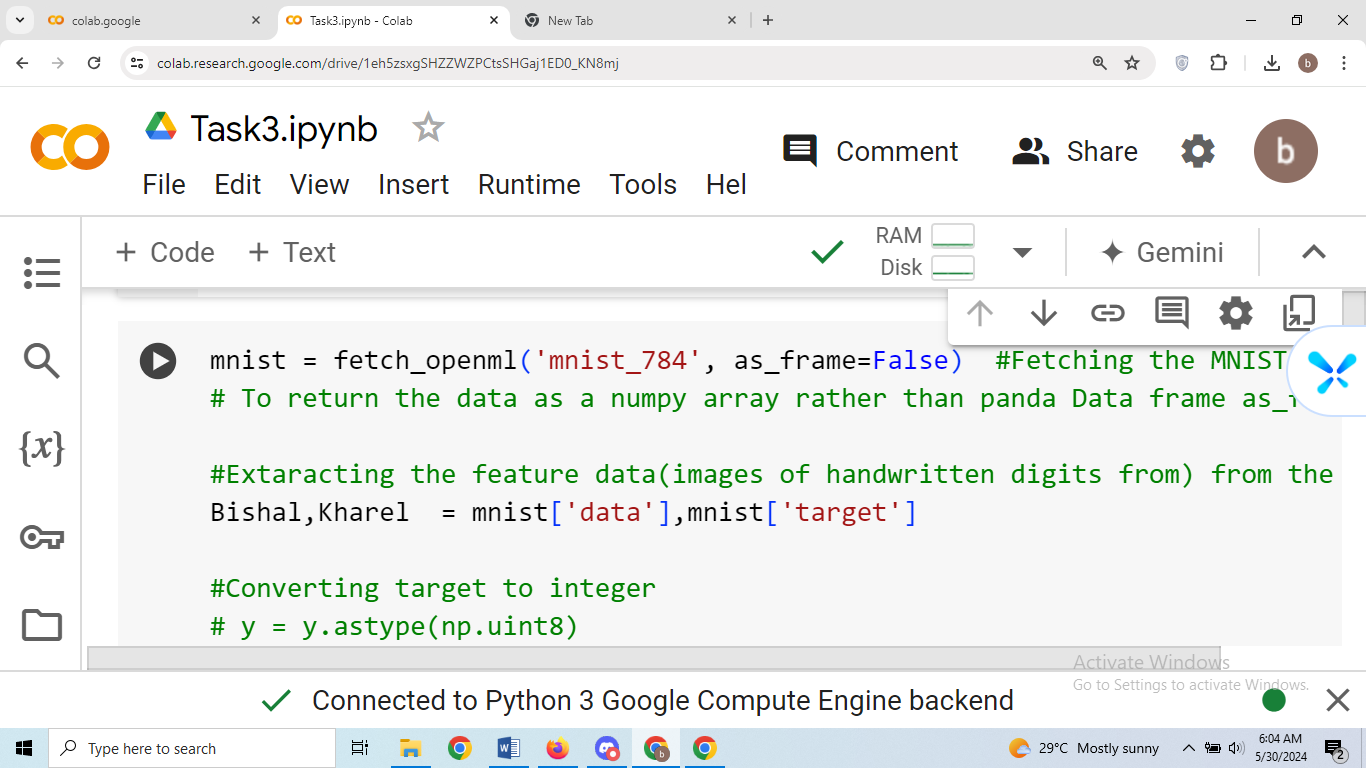
**Step 1.1:** Loading the required python libraries:

**Input:**

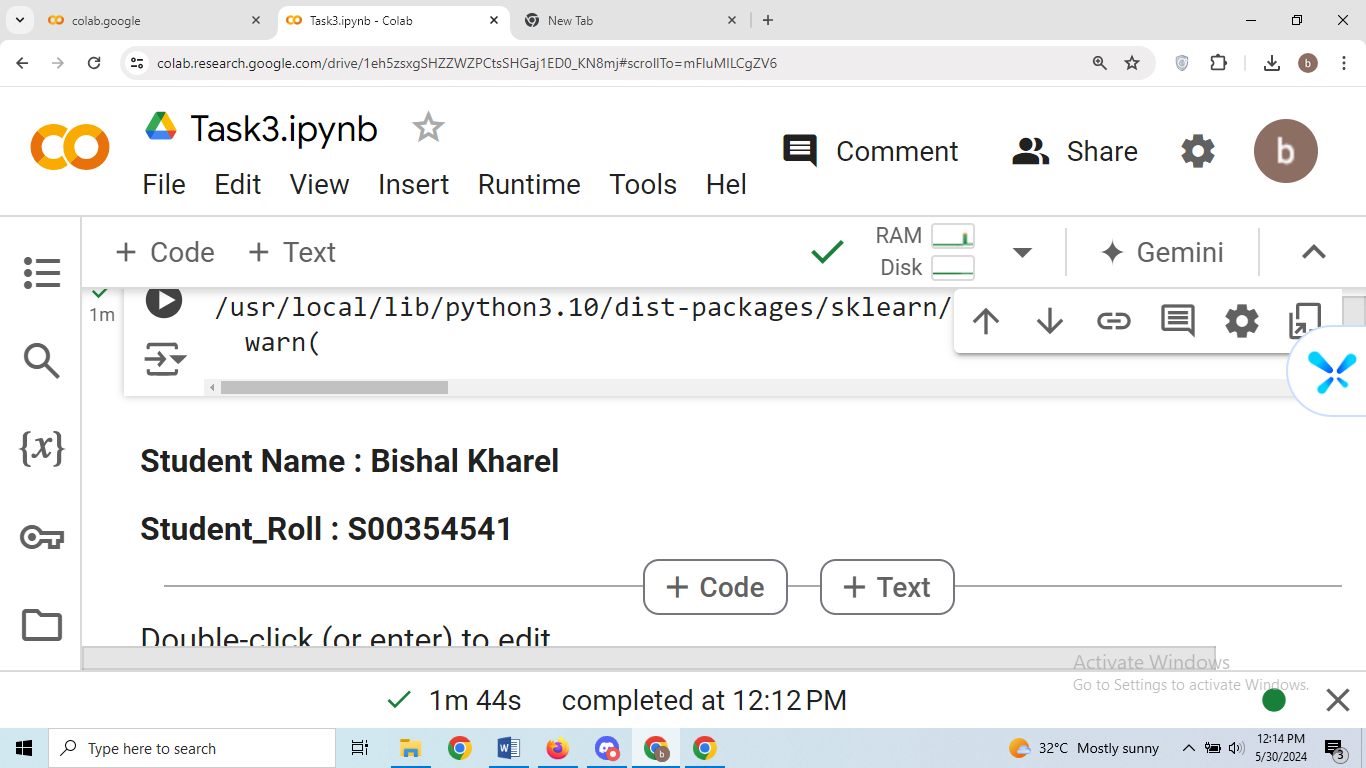
****

**Step 1.2:** Fetching 'mnist\_784' MNIST Hand digits dataset from sklearn.datasets.fetch\_openml library. Extracting the feature data (images of handwritten digits from) from the fetched dataset storing into Bishal variable, similarly extracting 'target' feature into the Kharel variable

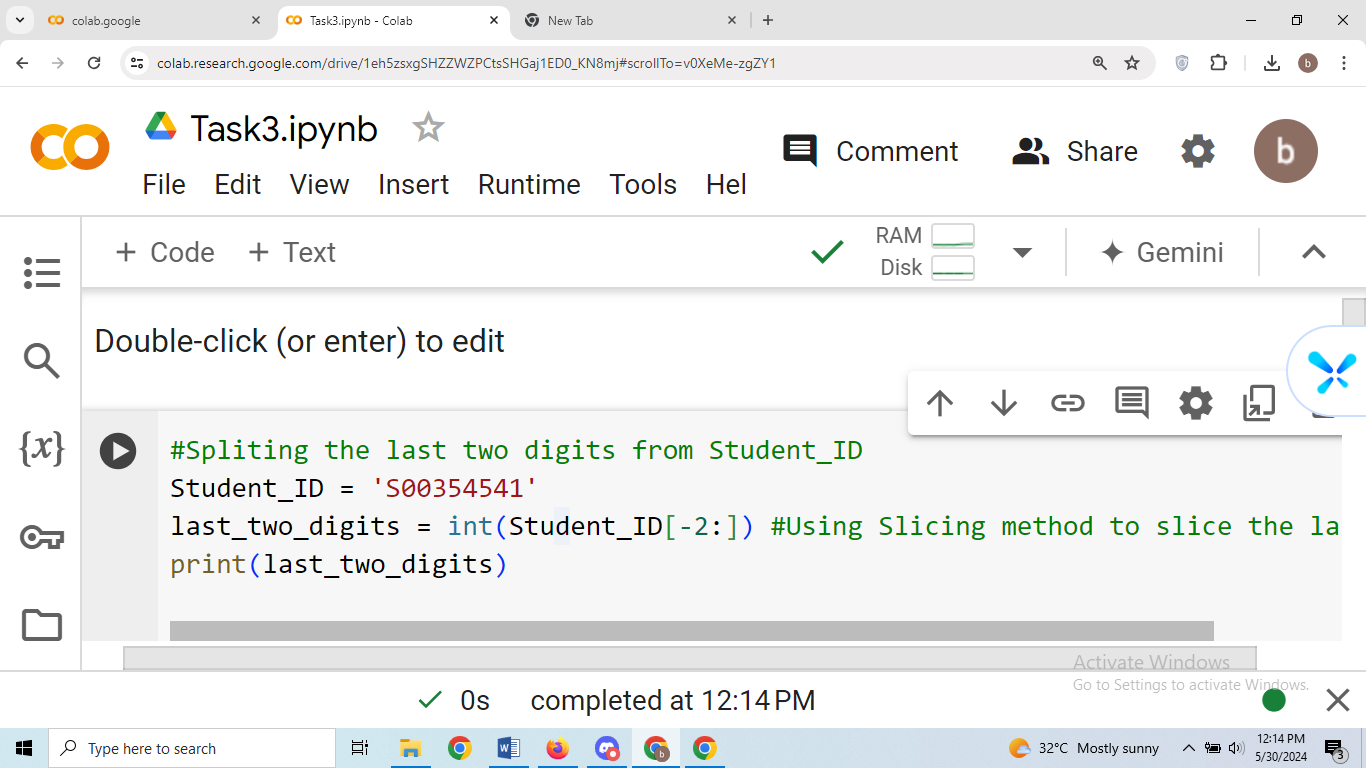
**Input:**

****

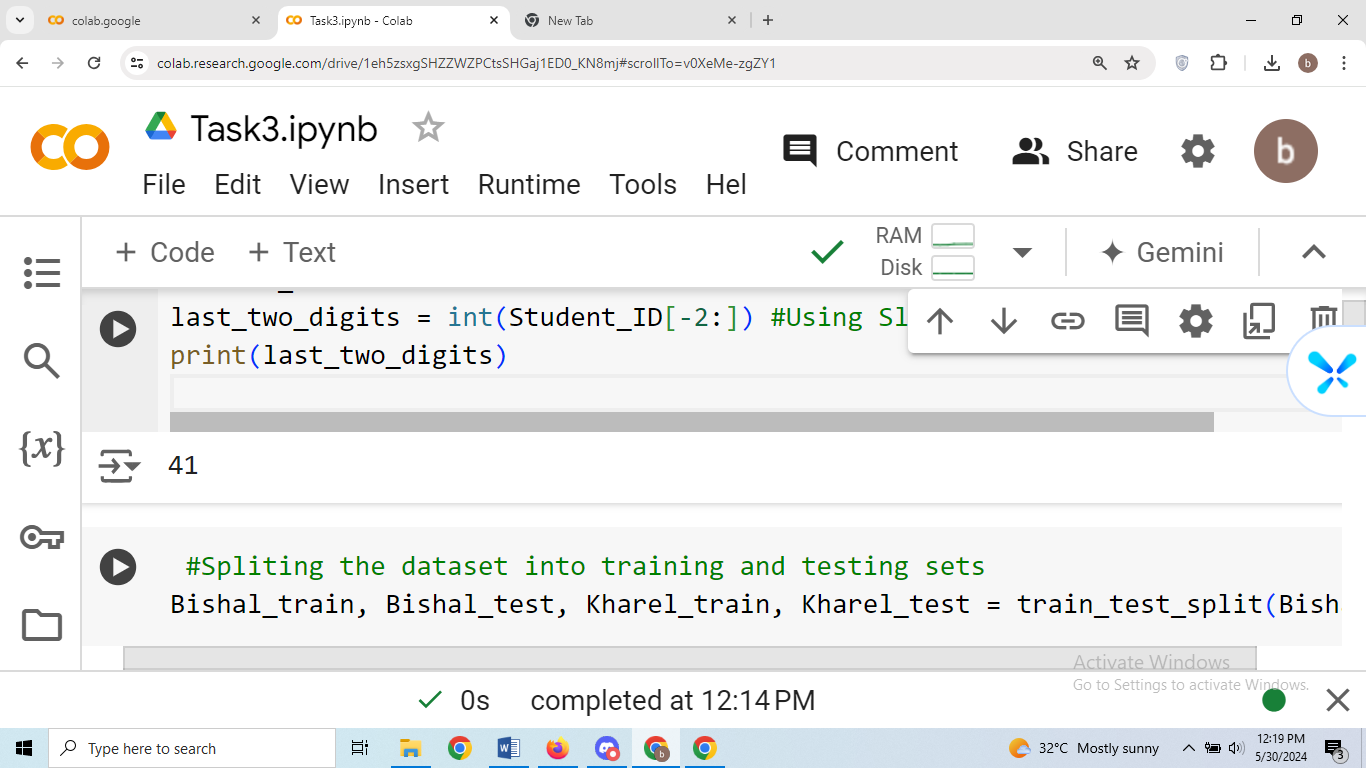
**Step 1.3:** Displaying Student\_Name and splitting the last two digits of student\_ID:



**Input:**

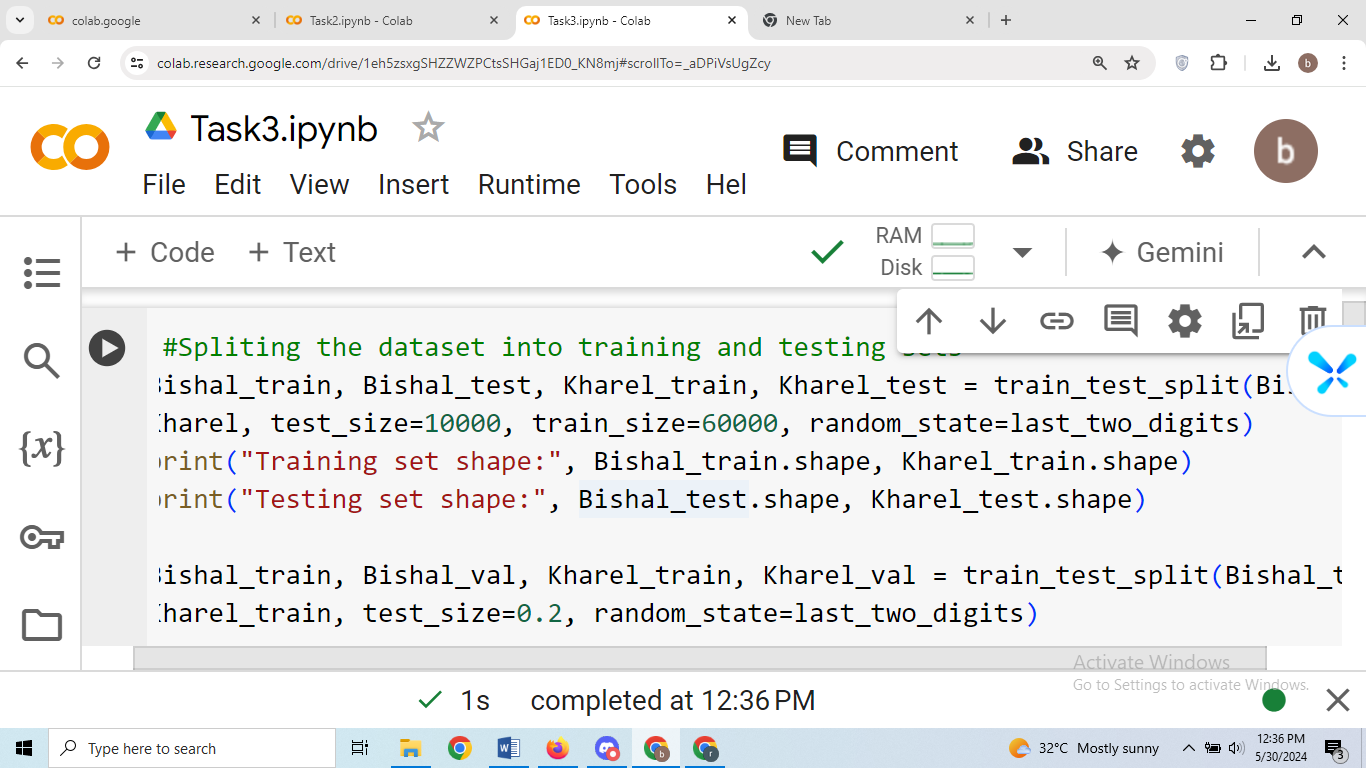
****

**Output:**

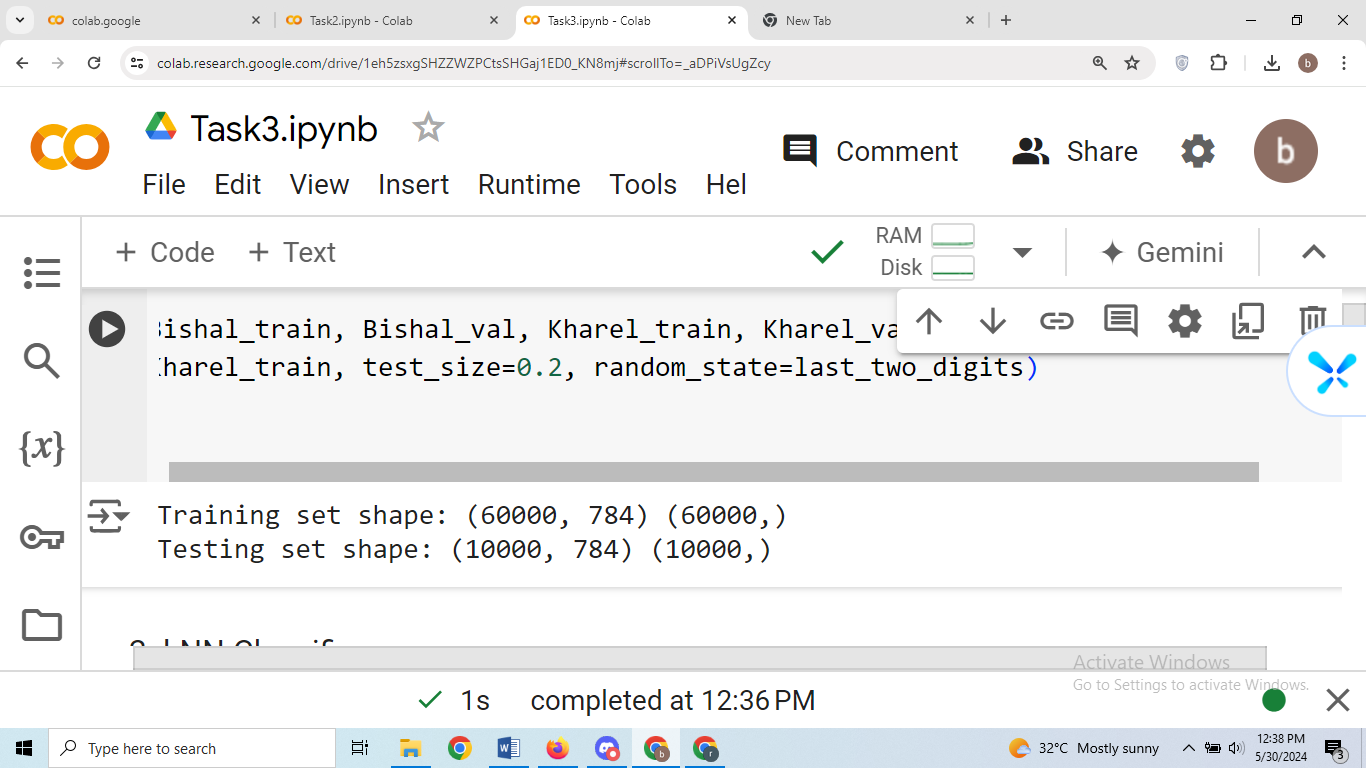
****

**Step 1.4:** Splitting the data into the train test split using sklearn.model\_learn.train\_test\_split function applying condition 1 and 2.

**Input:**



**Output:**

****

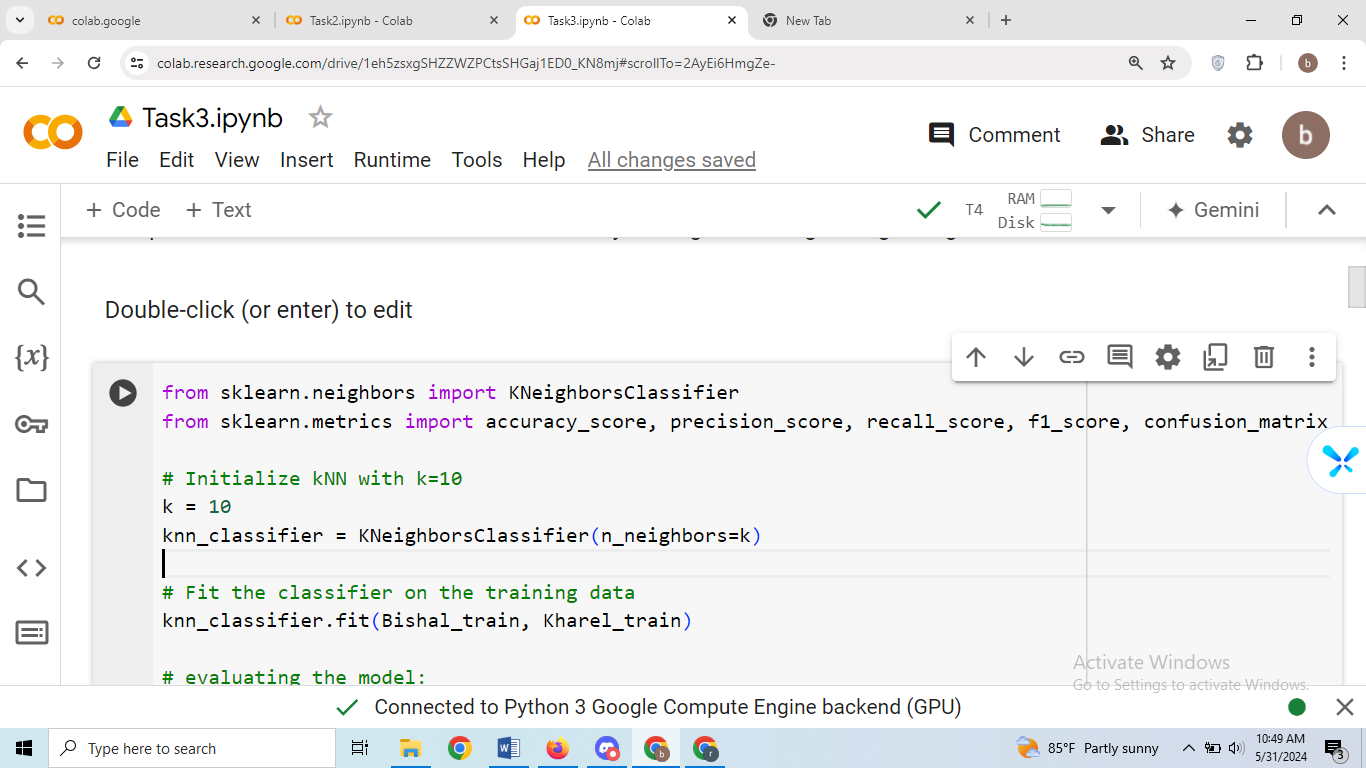
**2. kNN Classifier**

a. Set k=10. Use the kNN classifier from scikit-learn, explaining the function parameters and their implications.

b. Evaluate using metrics from the workshop. Discuss why specific metrics were chosen and what they indicate about the model.

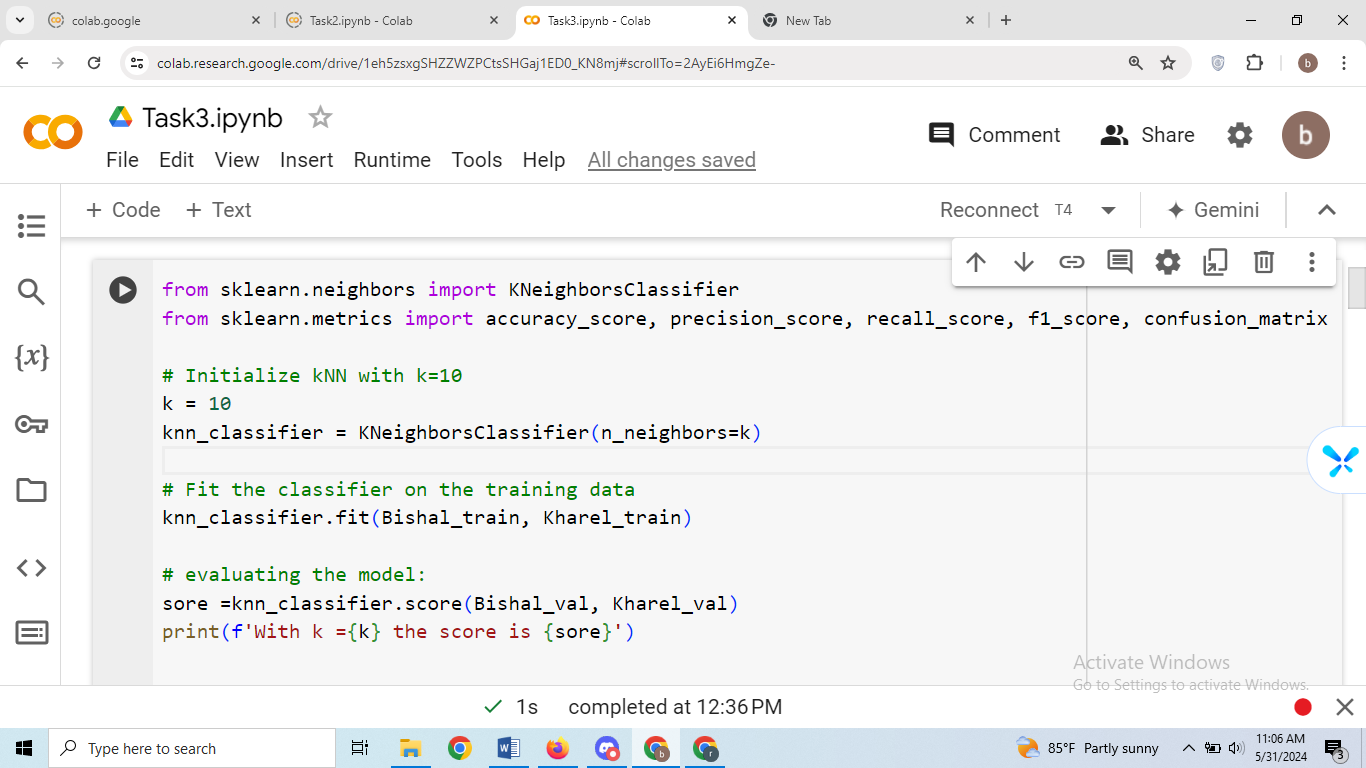
c. Experiment with different k values. Discuss any findings and insights regarding the choice of k

**Step 2.1:** Loading the required python libraries:

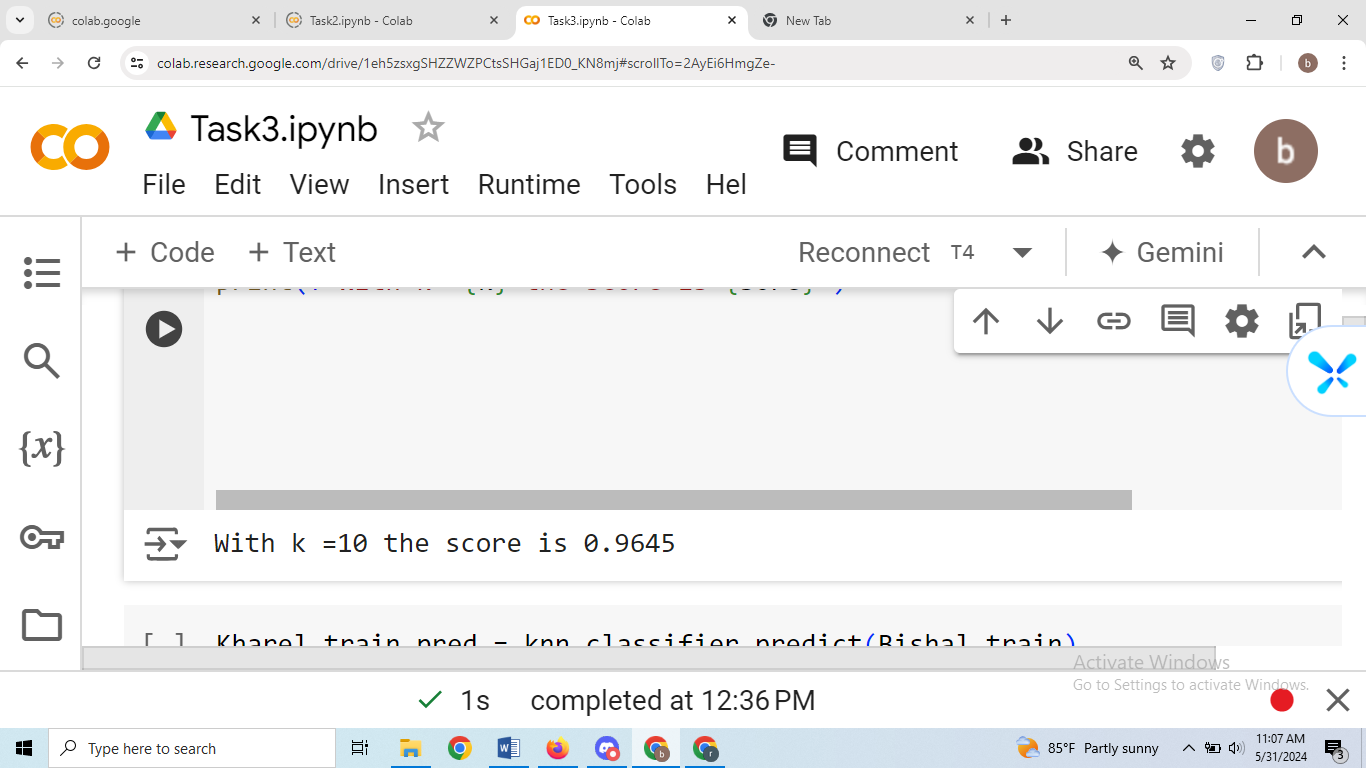


**Step 2.2:** Setting no of neighbors k = 10,using kNN classifier from scikit-learn and explaining the function parameters and its implications.

**Input:**

****

**Output:**

****

**Functional Parameters and its implication:**

k-Nearest Neighbors (kNN) Classifier

The k-Nearest Neighbors (kNN) algorithm is a simple, non-parametric, and lazy learning algorithm used for classification and regression. It is based on the principle that similar data points are close to each other in the feature space.

  
According to the figure KNN classifier helps to classification of images according to the features of label.

Parameters of KNeighborsClassifier

1. n\_neighbors:

* + This parameter specifies the number of neighbors to use for classification.
  + Implication: Setting k=10 means that the algorithm will consider the 10 nearest neighbors to a data point when making a classification decision. If the majority of these neighbors belong to a particular class, the data point will be classified as belonging to that class. A smaller k value can make the model more sensitive to noise in the data (overfitting), while a larger k value can smooth out the decision boundary (underfitting).

2. weights (default='uniform'):

* + This parameter determines the weight function used in prediction.
  + Setting to 'uniform', all points in each neighborhood are weighted equally.
  + Implication: Using weights='uniform' (the default) means each of the 10 neighbors contributes equally to the decision.

3. algorithm (default='auto'):

* + This parameter specifies the algorithm used to compute the nearest neighbors.
  + Implication: The 'auto' option will attempt to determine the best algorithm based on the values passed to fit method. It balances between computational efficiency and accuracy.

1. leaf\_size (default=30):
   * This parameter affects the speed of the construction and query processes in the BallTree or KDTree algorithms.
   * Implication: The default value works well for most cases. A smaller leaf size will result in a more complex tree structure.

5. p (default=2):

* + This parameter specifies the power parameter for the Minkowski distance metric.
  + Implication: Using the Euclidean distance (p=2) is a common choice for many applications, as it captures the straight-line distance between points by using the formula.

d = sqrt {(x\_2 - x\_1) ^2 + (y\_2-y\_1) ^2}

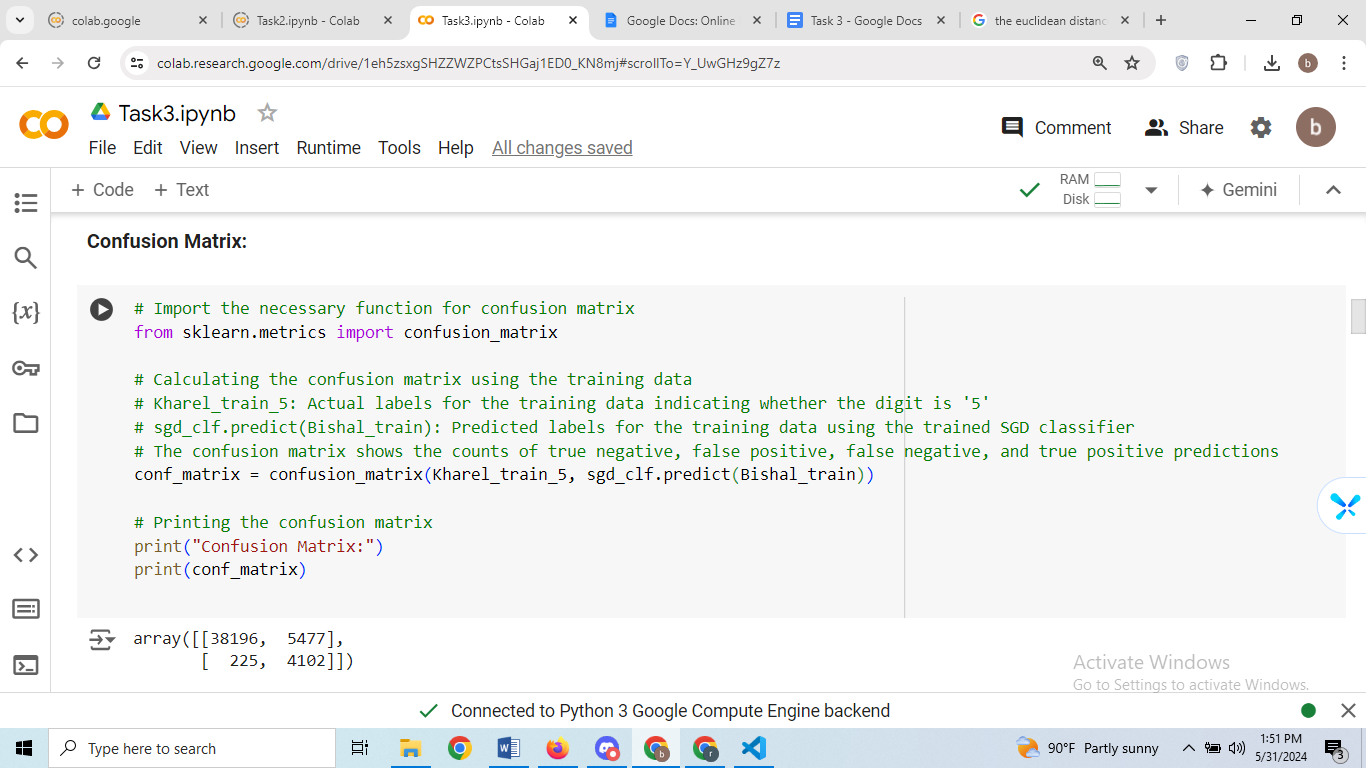
d = distance

(x\_1, y\_1) = coordinates of the first point

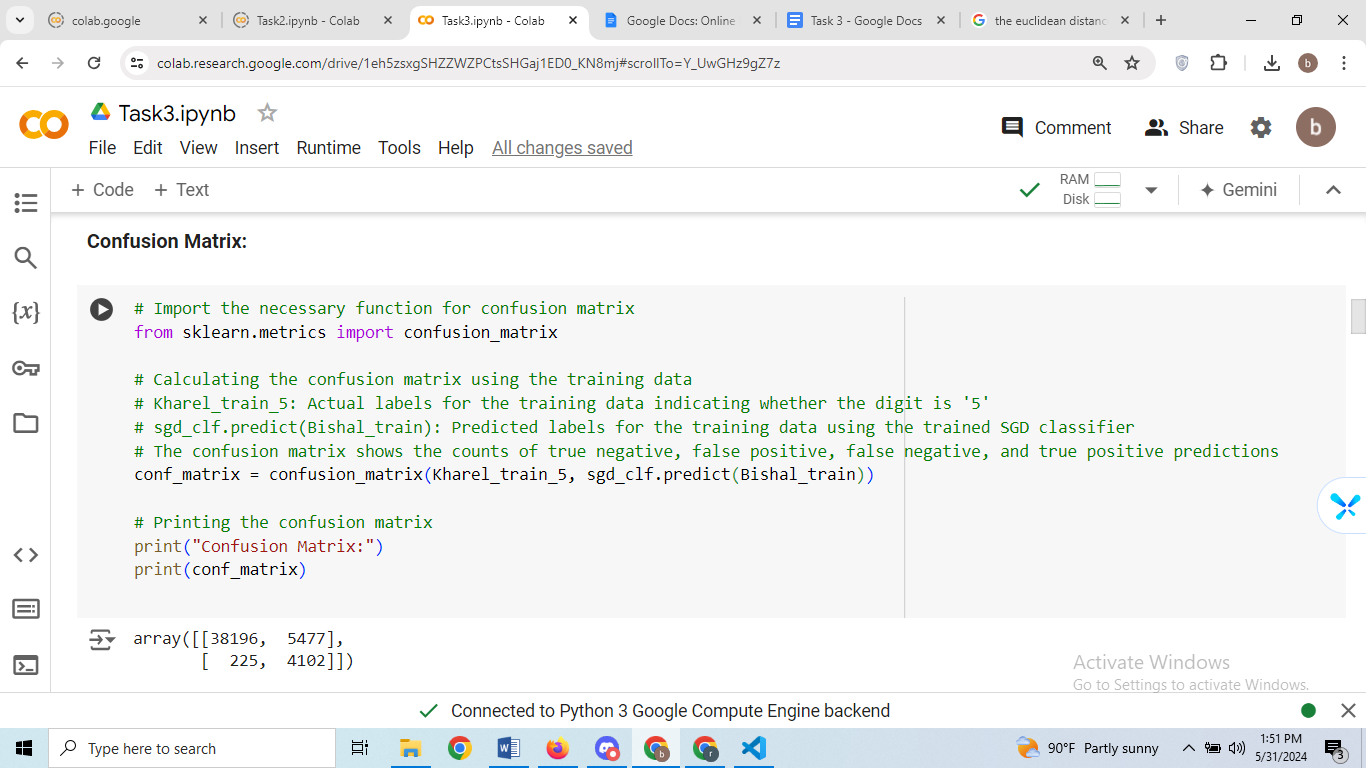
(x\_2, y\_2) = coordinates of the second point

**Step 2.3:** Evaluating confusion Matrix:

**Input:**

****

**Output:**

****

**Evaluation:**

* True Negative (TN): The number of instances that are actually negative (not '5') and predicted as negative by the classifier. In this case, there are 38,196 instances.
* False Positive (FP): The number of instances that are actually negative but predicted as positive ('5') by the classifier. In this case, there are 5,477 instances.
* False Negative (FN): The number of instances that are actually positive ('5') but predicted as negative by the classifier. In this case, there are 225 instances.
* True Positive (TP): The number of instances that are actually positive ('5') and predicted as positive by the classifier. In this case, there are 4,102 instances.

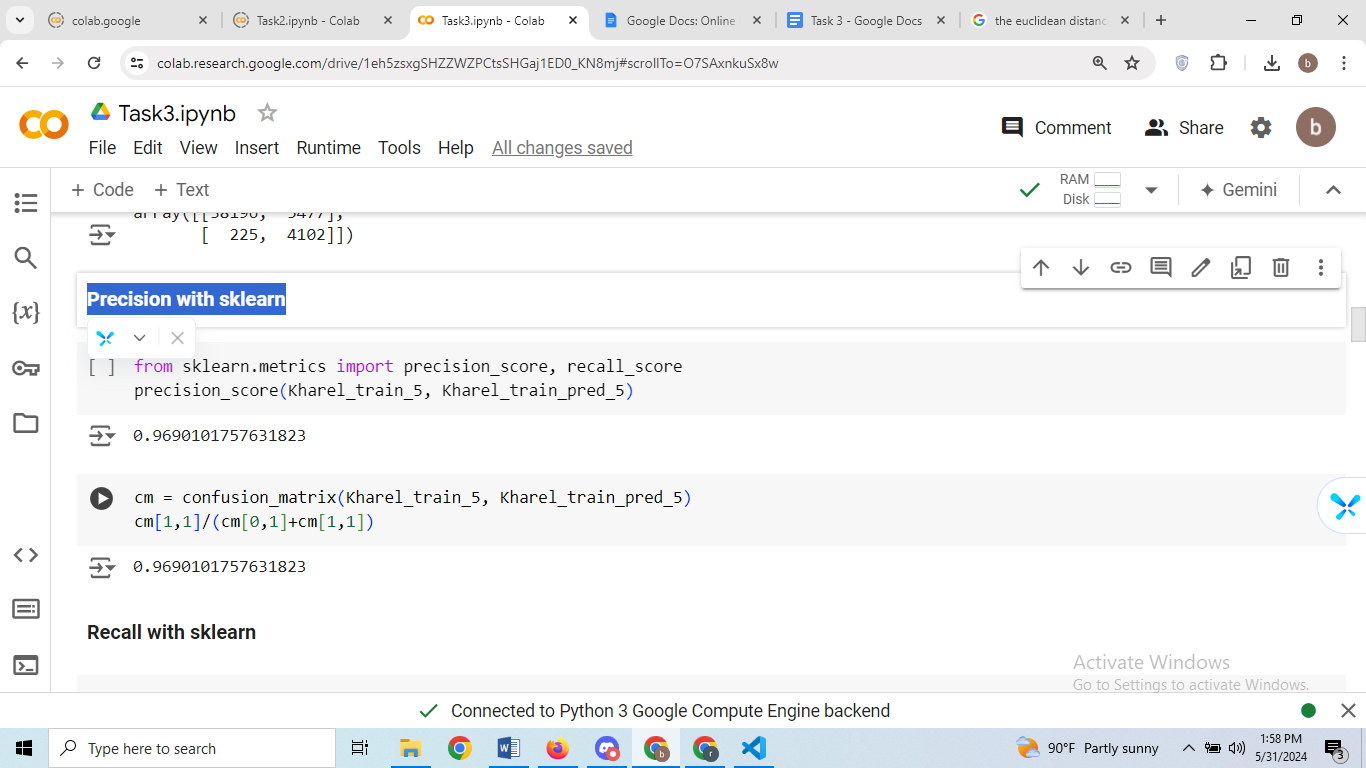
The classifier correctly predicted 38,196 instances as negative and 4,102 instances as positive.

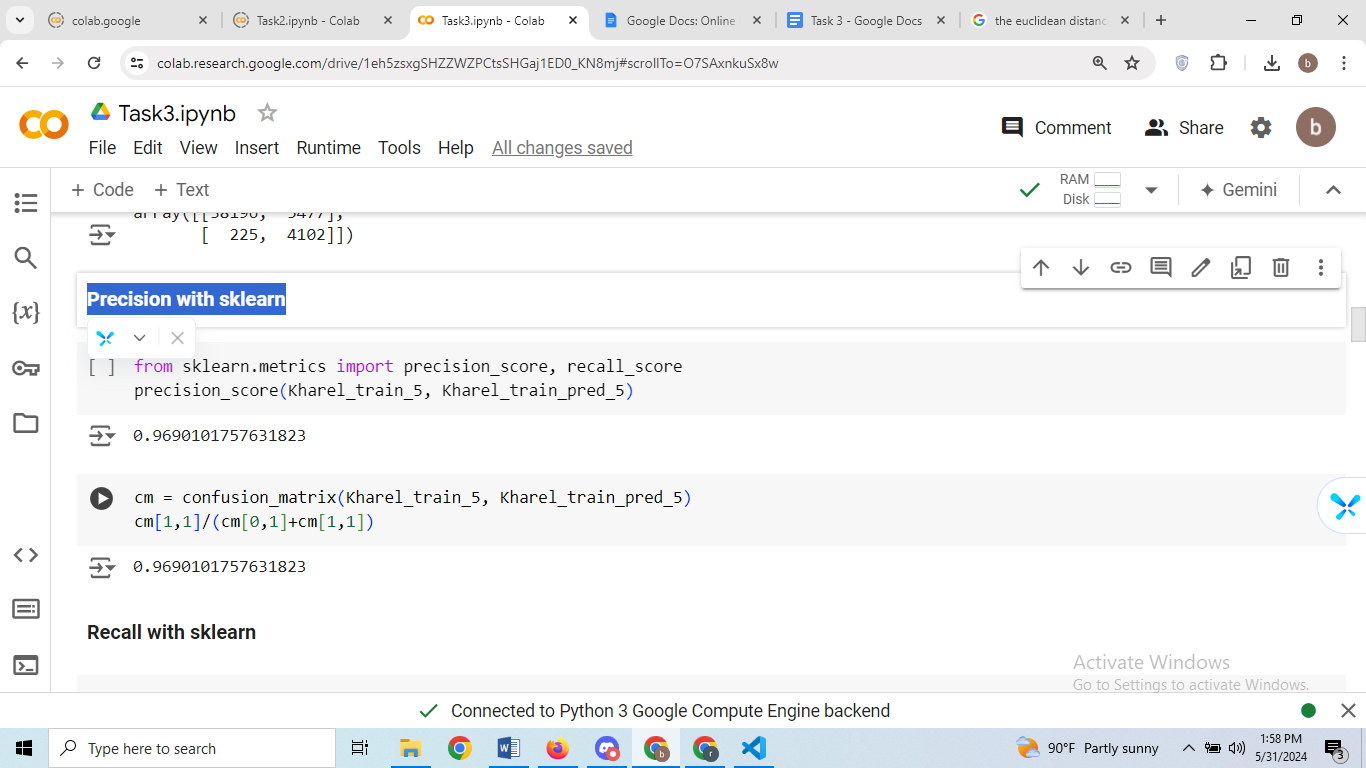
The classifier incorrectly predicted 5,477 instances as positive when they were actually negative, and 225 instances as negative when they were actually positive.

The confusion matrix provides a detailed breakdown of the kNN classifier's predictions, including true positives, true negatives, false positives, and false negatives. It offers insights into where the model makes errors and its performance across different classes.

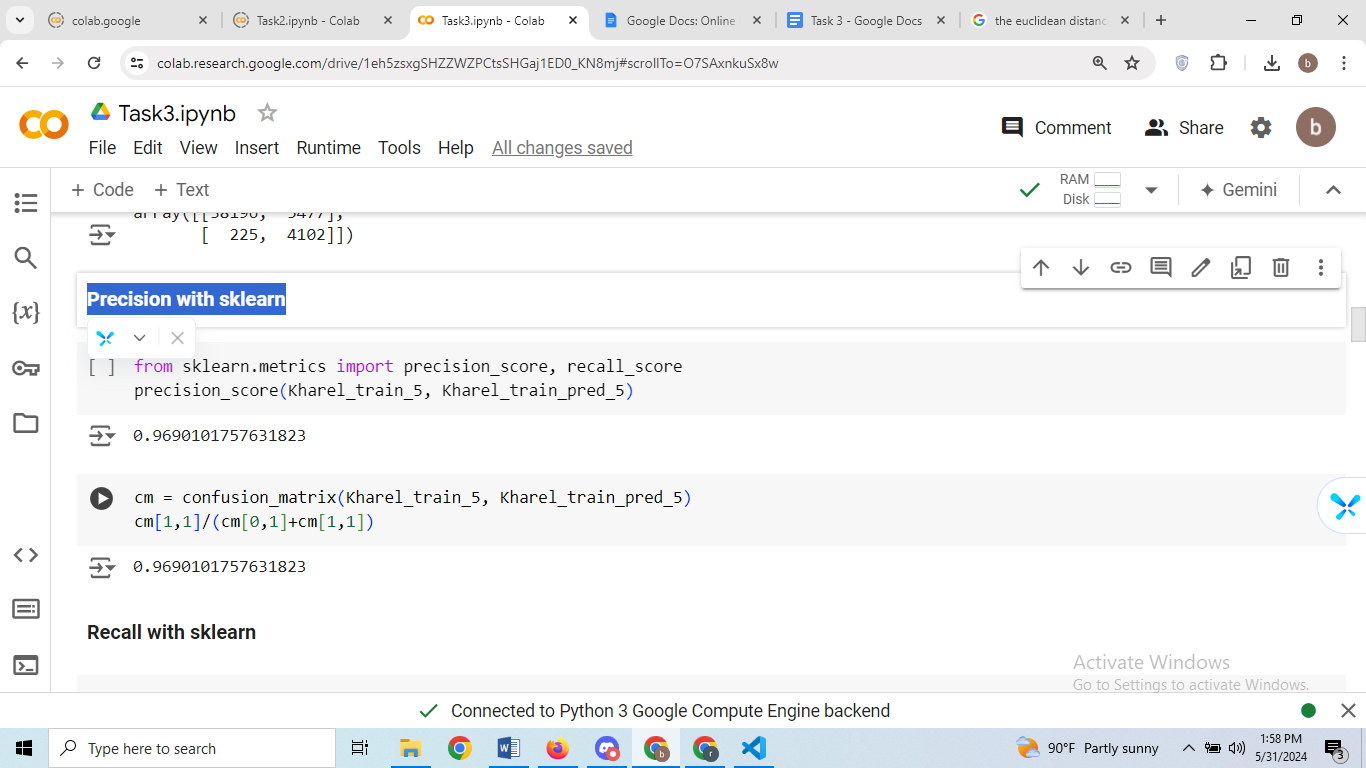
**Step 2.4: Precision with sklearn**

**Input:**





**Output:**

****

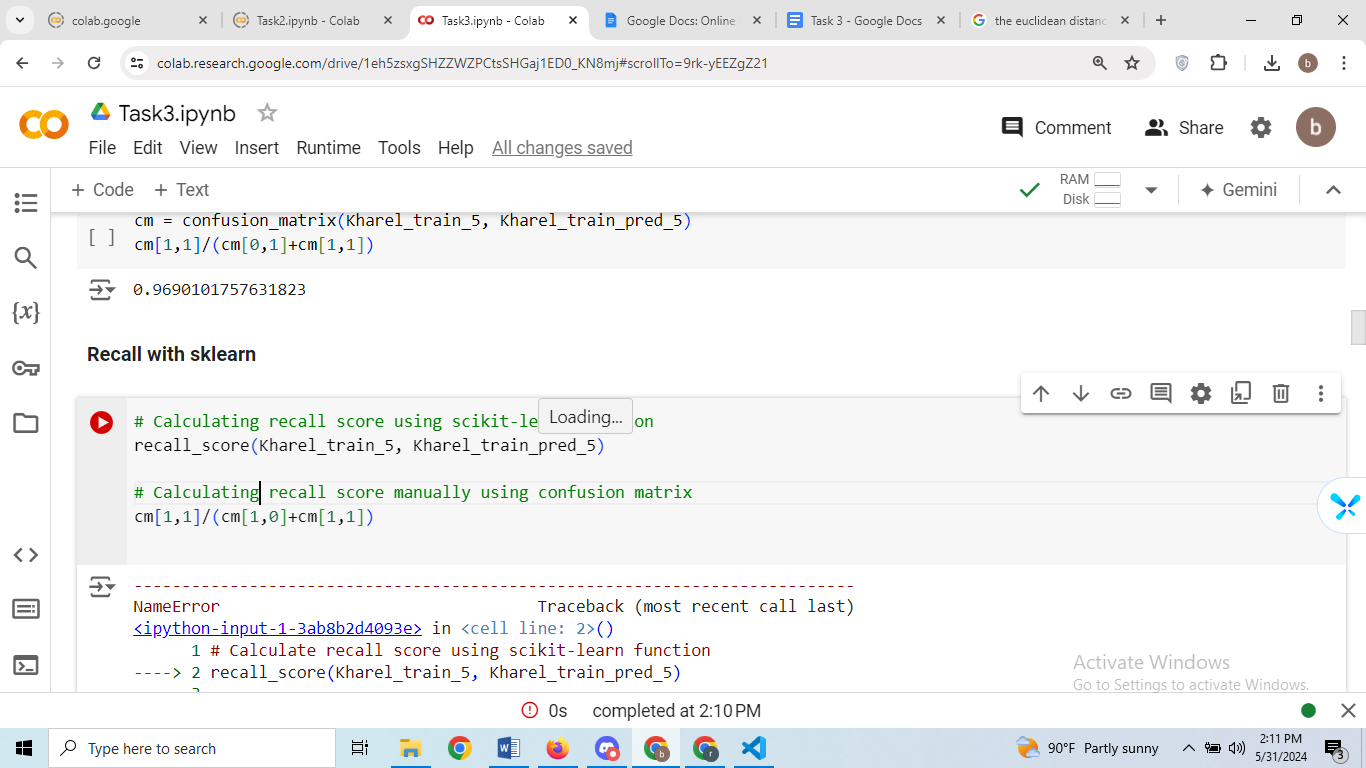
**Evaluation:**

cm = confusion\_matrix(Kharel\_train\_5, Kharel\_train\_pred\_5): It calculates the confusion matrix using the actual labels (Kharel\_train\_5) and the predicted labels (Kharel\_train\_pred\_5) for the training data. The confusion matrix provides a detailed breakdown of the model's predictions, including true positives, true negatives, false positives, and false negatives.

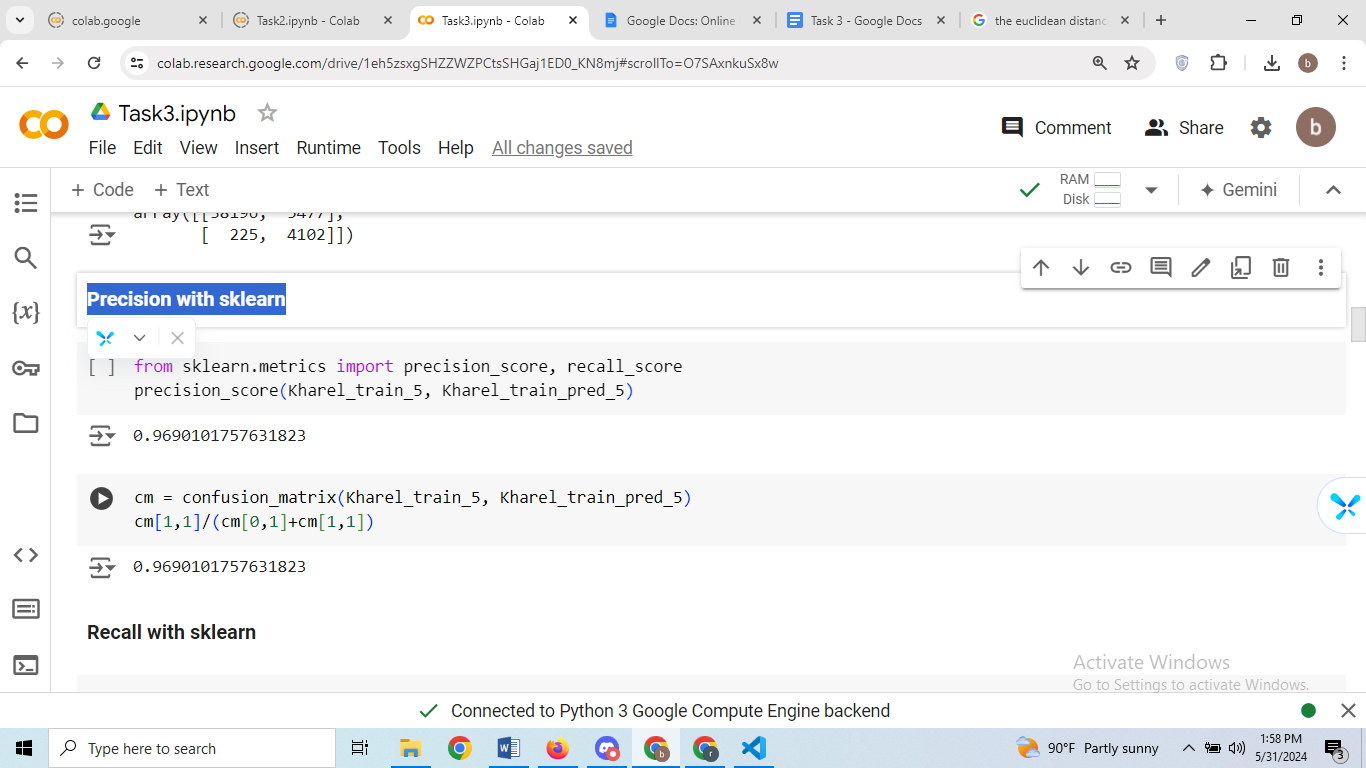
cm[1,1]/(cm[0,1]+cm[1,1]): This line computes precision manually using the counts from the confusion matrix. It calculates the proportion of true positive predictions out of all positive predictions made by the model. Specifically, it divides the number of true positive predictions by the sum of false positives and true positives. This calculation yields the precision score.

**Step 2.5 : Recall with sklearn**

**Input:**

****

**Output:**

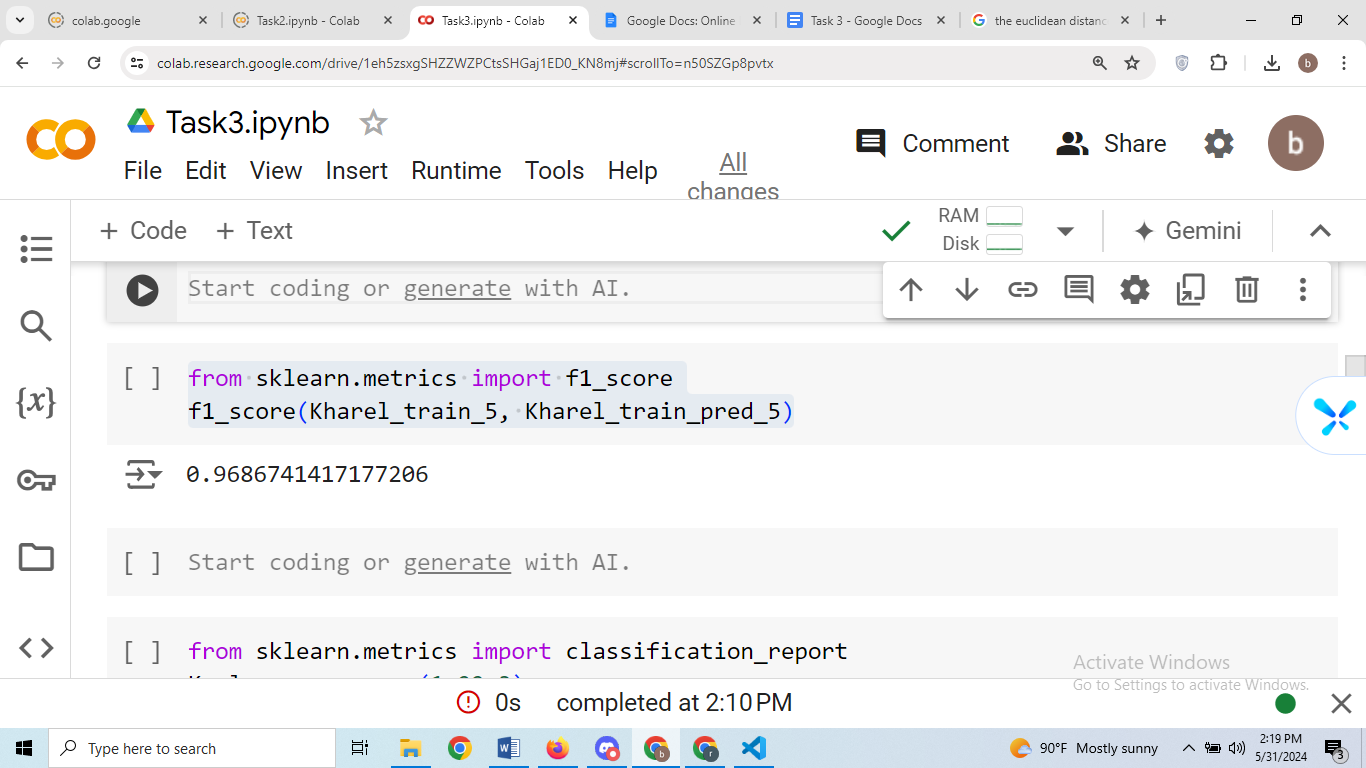
****

**Evaluation:**

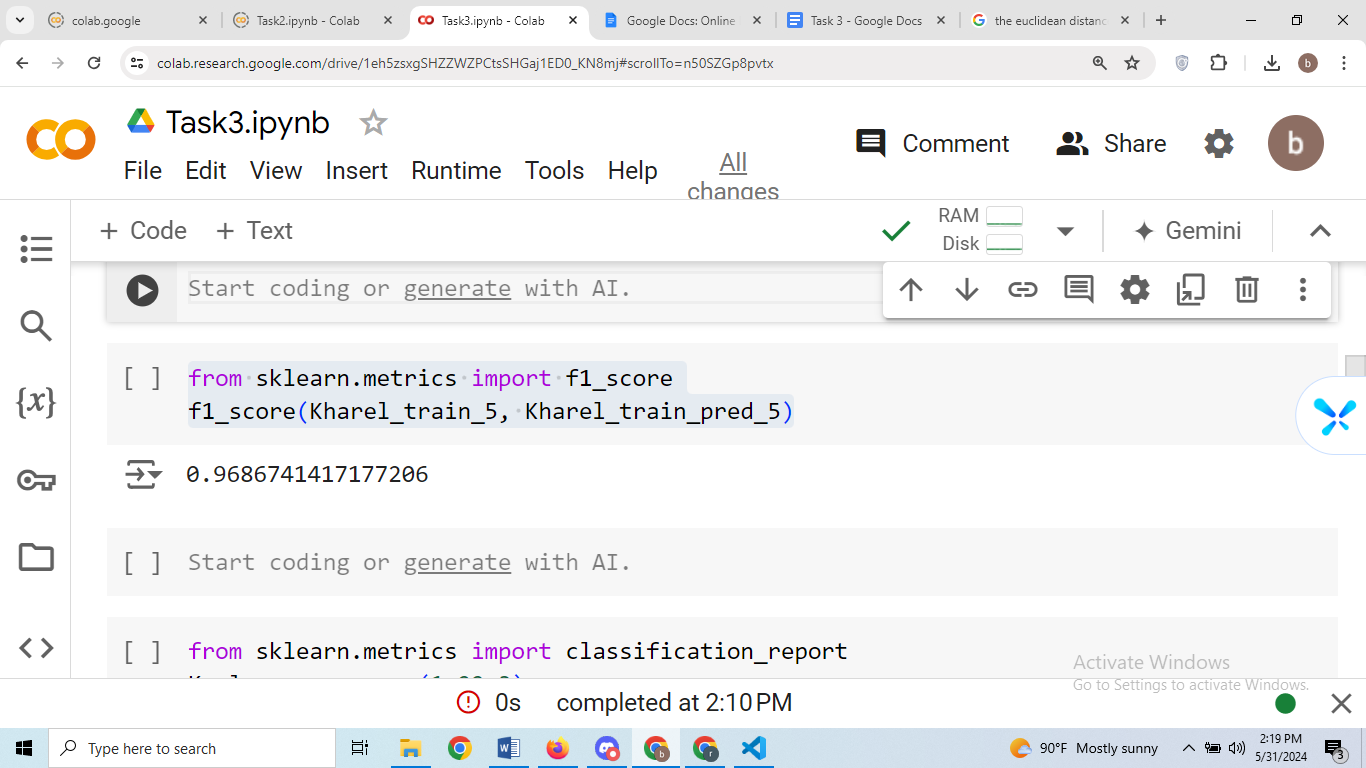
* **cm[1,1] retrieves the count of true positive predictions from the confusion matrix, representing the number of instances correctly classified as positive ('5').**
* **(cm[1,0]+cm[1,1]) calculates the sum of false negatives and true positives. False negatives are instances that are actually positive but incorrectly classified as negative.**
* **The division cm[1,1]/(cm[1,0]+cm[1,1]) computes the proportion of true positive predictions out of all actual positive instances in the dataset, yielding the recall score**
* **The output of 0.96 is the recall score obtained when the second line of code is executed manually. It indicates that approximately 96% of actual positive instances (in this case, instances labeled as '5') were correctly identified by the classifier. This value suggests that the kNN classifier has a high recall rate, meaning it effectively captures most of the positive instances in the dataset.**

**Step 2.6 : Calculating F1 score**

**Input:**

****

**Output:**

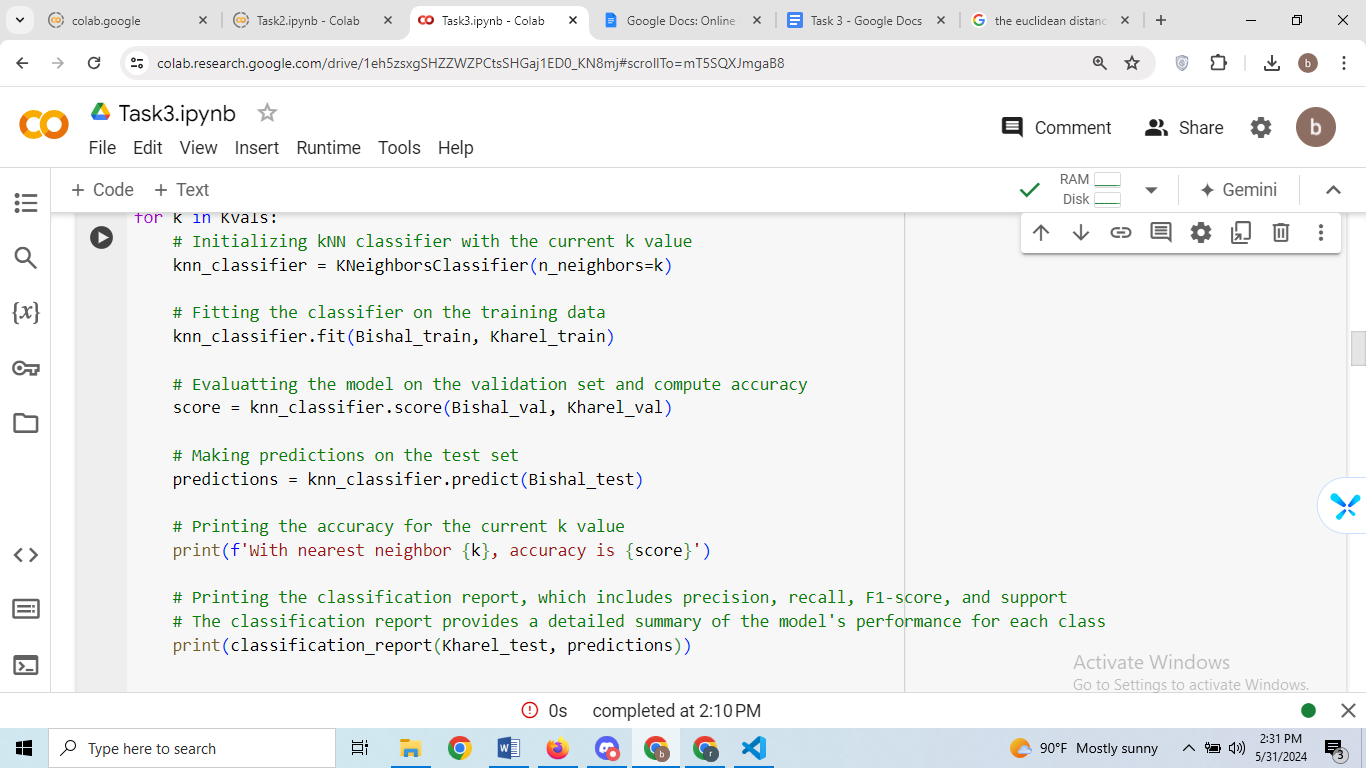
****

**Evaluation:**

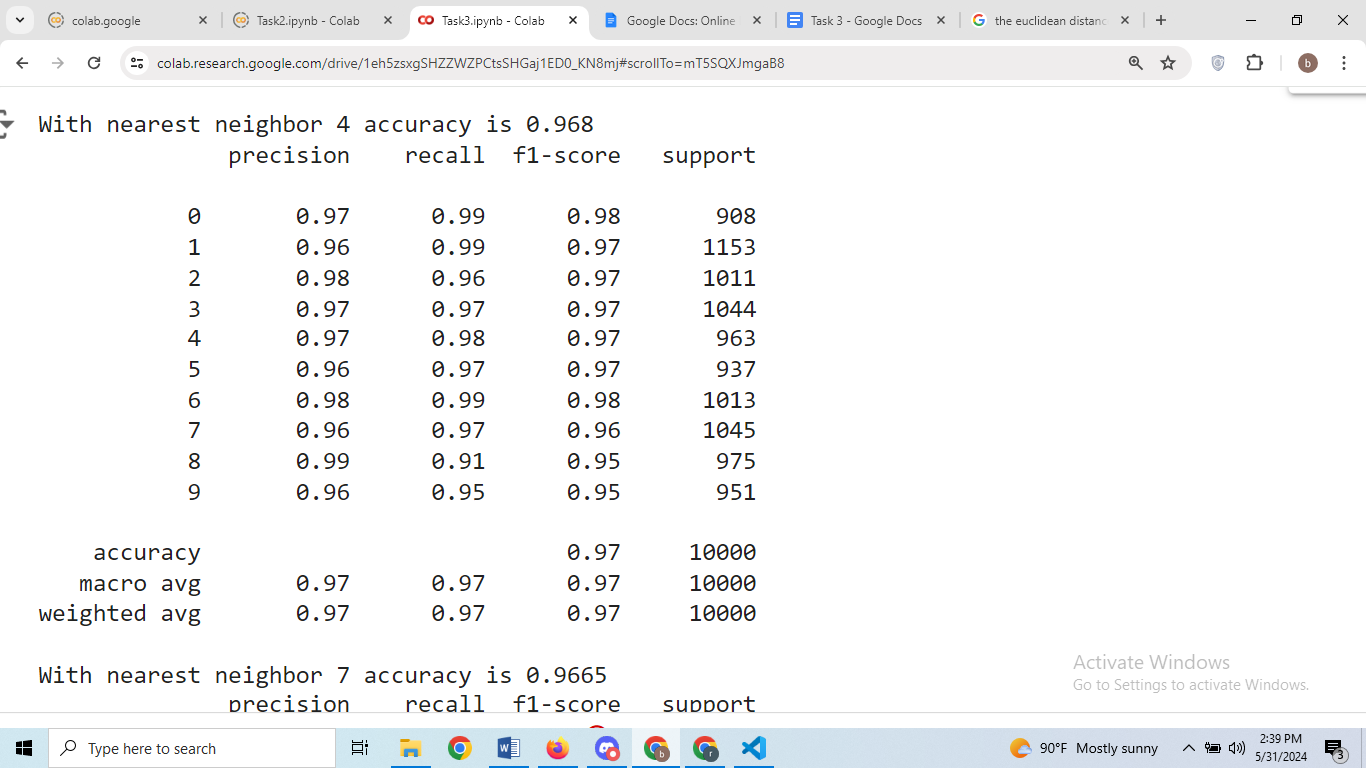
* The f1\_score function computes the harmonic mean of precision and recall, providing a single metric that balances both precision and recall. It is a useful measure for assessing the overall performance of the classifier while considering both false positives and false negatives.
* The output of 0.96 represents the F1 score obtained when the commented line of code is executed. This value indicates the overall effectiveness of the kNN classifier in correctly classifying positive instances (in this case, instances labeled as '5') while minimizing both false positives and false negatives. A high F1 score suggests that the classifier achieves a good balance between precision and recall, indicating strong performance overall.

**Step 2.8:** Experimenting with different k value:  
**Input:**





**Output:**

****

****

**Evaluation:  
1. Accuracy for Different k Values:**

* For each value of k (1, 4, 7, 10, 13, 16, 19), the output displays the accuracy of the kNN classifier on the test set.
* The accuracy measures the proportion of correctly classified instances out of all instances in the test set.
* As the value of k increases, the accuracy generally decreases slightly, indicating a potential trade-off between accuracy and the complexity of the model.

**2. Classification Report for Each k Value:**

* Following the accuracy for each k value, the output provides a classification report for the predictions made by the kNN classifier.
* The classification report includes precision, recall, F1-score, and support for each class (digits 0 through 9), as well as macro and weighted averages.
* Precision measures the proportion of true positive predictions out of all positive predictions for each class.
* Recall measures the proportion of true positive predictions out of all actual instances of each class.
* F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics.
* Support indicates the number of actual instances of each class in the test set.
* The macro and weighted averages provide overall metrics across all classes, with the weighted average considering the class imbalance.

**3. Overall Analysis:**

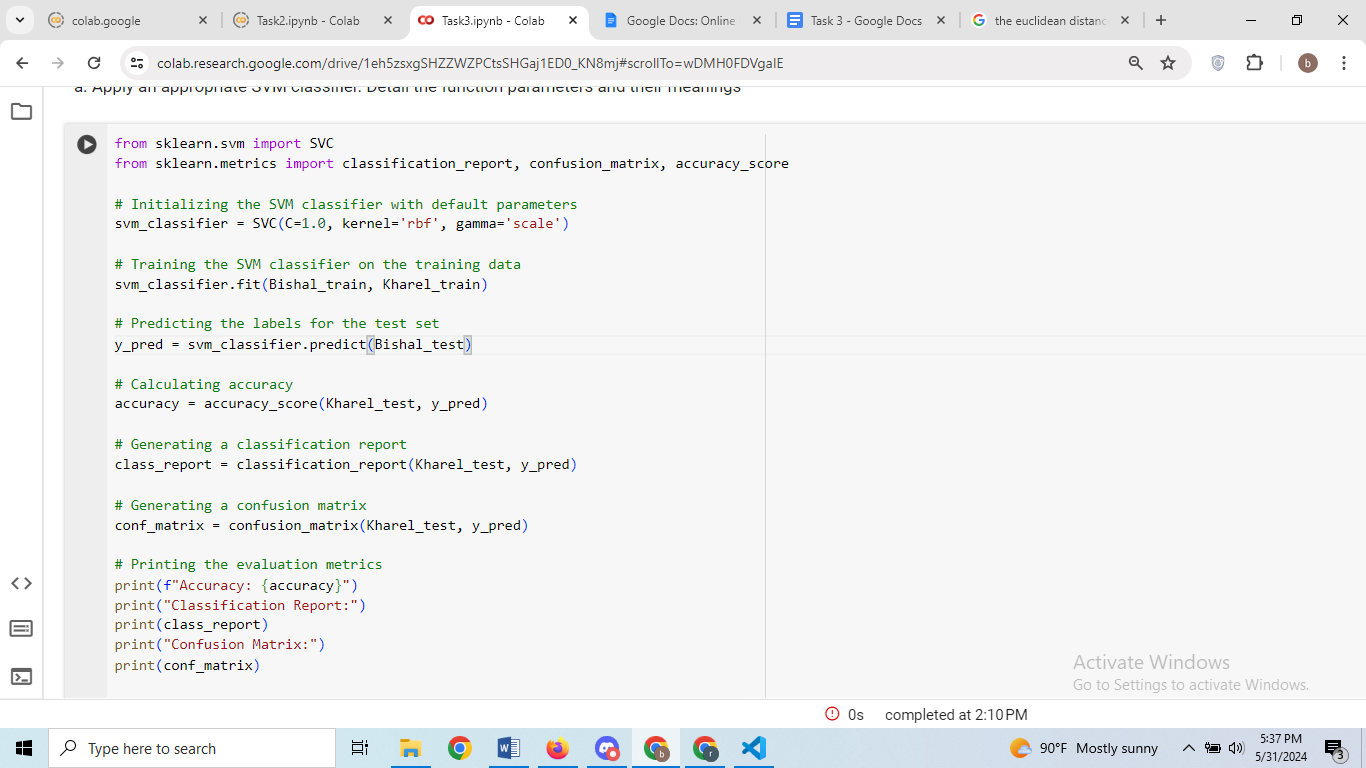
* The kNN classifier achieves high accuracy (>96%) for all tested values of k.
* Precision, recall, and F1-score are generally high for most classes across different values of k, indicating good performance in correctly identifying digits.
* The classification reports suggest that the classifier performs well across different classes, with minor variations in performance metrics for different k values.
* Overall, the kNN classifier demonstrates consistent and reliable performance on the test set, with varying levels of complexity determined by the choice of k.

**3. SVM Classifier**

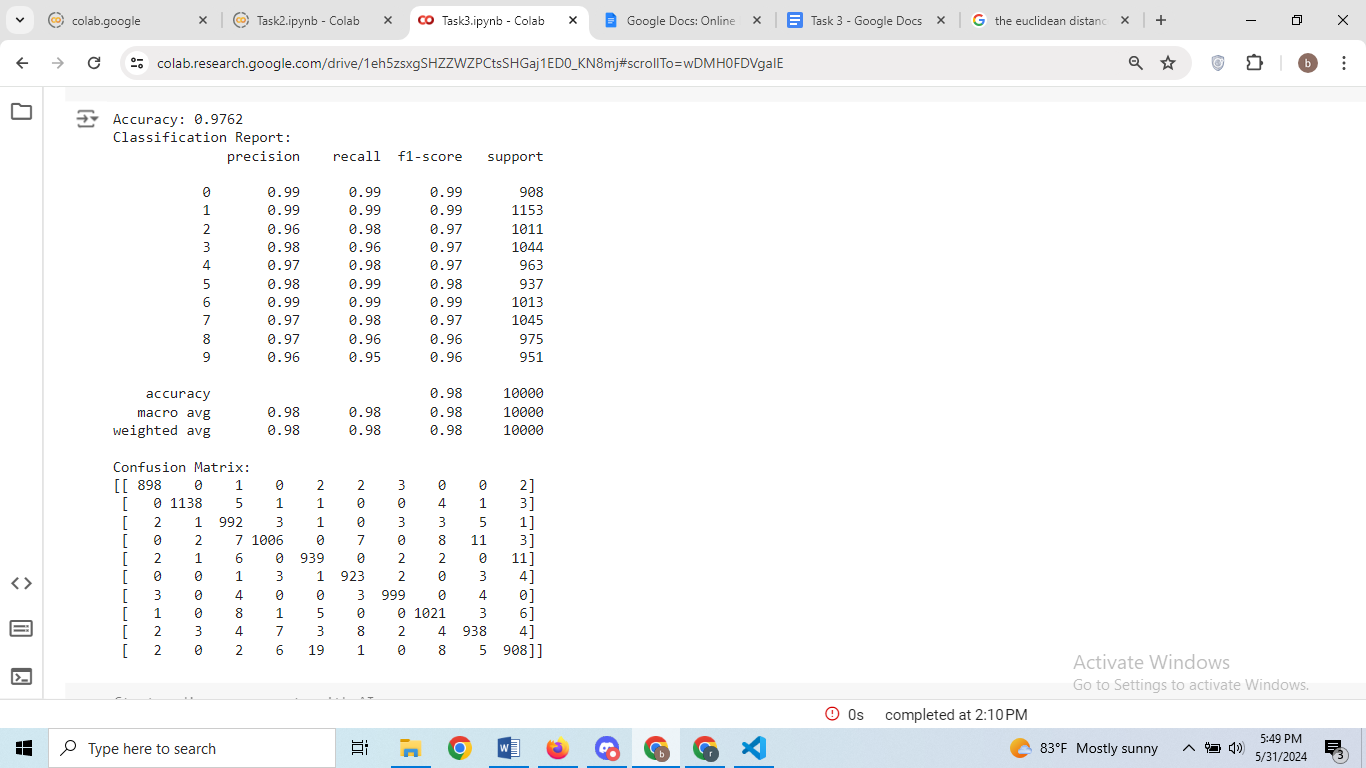
a. Apply an appropriate SVM classifier. Detail the function parameters and their meanings.

**Step 3.1:**

**Input:**

****

**Output:**

****

**Evaluation:**

**Parameters of SVC:**

* C=1.0: Regularization parameter. It controls the trade-off between achieving a low training error and a low testing error (generalization). A smaller value of C creates a larger margin, meaning a simpler decision boundary, whereas a larger value of C creates a smaller margin, allowing the model to fit the training data more precisely.
* kernel='rbf': Specifies the kernel type to be used in the algorithm. 'rbf' stands for the radial basis function, which is a popular kernel for SVMs. Other options include 'linear', 'poly', 'sigmoid', etc.
* gamma='scale': Kernel coefficient for ‘rbf’, ‘poly’, and ‘sigmoid’. When gamma is set to 'scale', it uses 1 / (n\_features \* X.var()) as the value of gamma. This parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.

**Accuracy:**

0.9762: This means that 97.62% of the predictions made by the SVM classifier were correct. This is a high accuracy rate, indicating that the classifier performs well on the test data.

**Classification Report:**

The classification report provides detailed metrics for each class (0 to 9).

**Precision:** Precision is the proportion of true positive predictions out of all positive predictions made by the model.

**Recall**: Recall is the proportion of true positive predictions out of all actual positive instances.

**F1-Score:** The F1-score is the harmonic mean of precision and recall, providing a balance between the two.

**Support**: The number of actual instances of each class in the test set.

**Per-Class Metrics:**

Class 0: Precision = 0.99, Recall = 0.99, F1-Score = 0.99, Support = 908

Class 1: Precision = 0.99, Recall = 0.99, F1-Score = 0.99, Support = 1153

Class 2: Precision = 0.96, Recall = 0.98, F1-Score = 0.97, Support = 1011

Class 3: Precision = 0.98, Recall = 0.96, F1-Score = 0.97, Support = 1044

Class 4: Precision = 0.97, Recall = 0.98, F1-Score = 0.97, Support = 963

Class 5: Precision = 0.98, Recall = 0.99, F1-Score = 0.98, Support = 937

Class 6: Precision = 0.99, Recall = 0.99, F1-Score = 0.99, Support = 1013

Class 7: Precision = 0.97, Recall = 0.98, F1-Score = 0.97, Support = 1045

Class 8: Precision = 0.97, Recall = 0.96, F1-Score = 0.96, Support = 975

Class 9: Precision = 0.96, Recall = 0.95, F1-Score = 0.96, Support = 951

**Averages:**

**Accuracy**: 0.9762 (overall accuracy)

**Macro Avg**: Precision = 0.98, Recall = 0.98, F1-Score = 0.98 (average metrics for all classes, treating all classes equally)

**Weighted Avg:** Precision = 0.98, Recall = 0.98, F1-Score = 0.98 (average metrics for all classes, weighted by support of each class)

**Confusion Matrix:**

The confusion matrix shows the counts of true positive, false positive, true negative, and false negative predictions for each class. Each row represents the true class, and each column represents the predicted class.

**Key Points from the Confusion Matrix:**

**Diagonal Values**: These are the true positive counts (e.g., 898 true positives for class 0, 1138 true positives for class 1, etc.).

**Off-Diagonal Values:** These are the counts of misclassifications (e.g., 2 instances of class 0 were misclassified as class 2, 1 instance of class 1 was misclassified as class 2, etc.).

Overall, the SVM classifier demonstrates high performance with an accuracy of 97.62%, and it performs well across all classes, as indicated by high precision, recall, and F1-scores for each class. The confusion matrix further confirms that the majority of predictions are correct, with relatively few misclassifications.

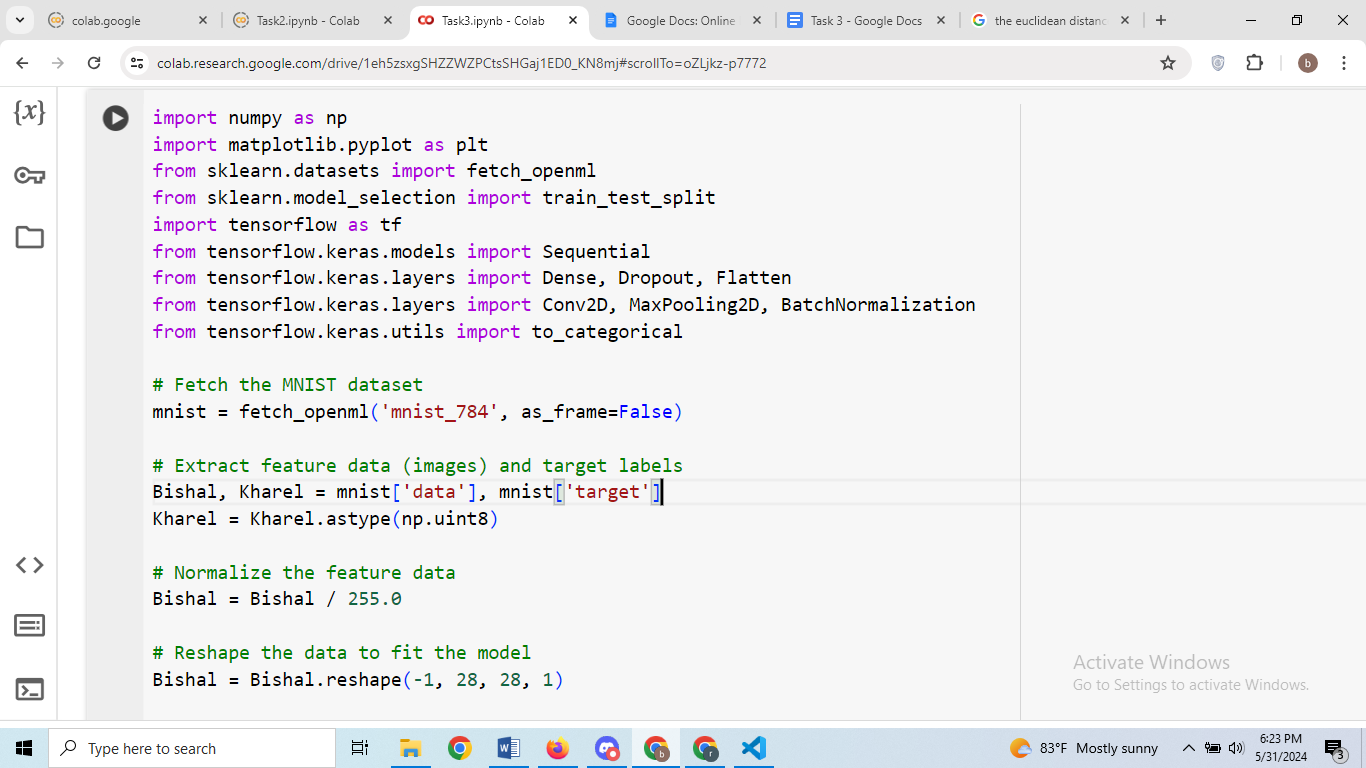
**4. Network Classification and Data Ethics**

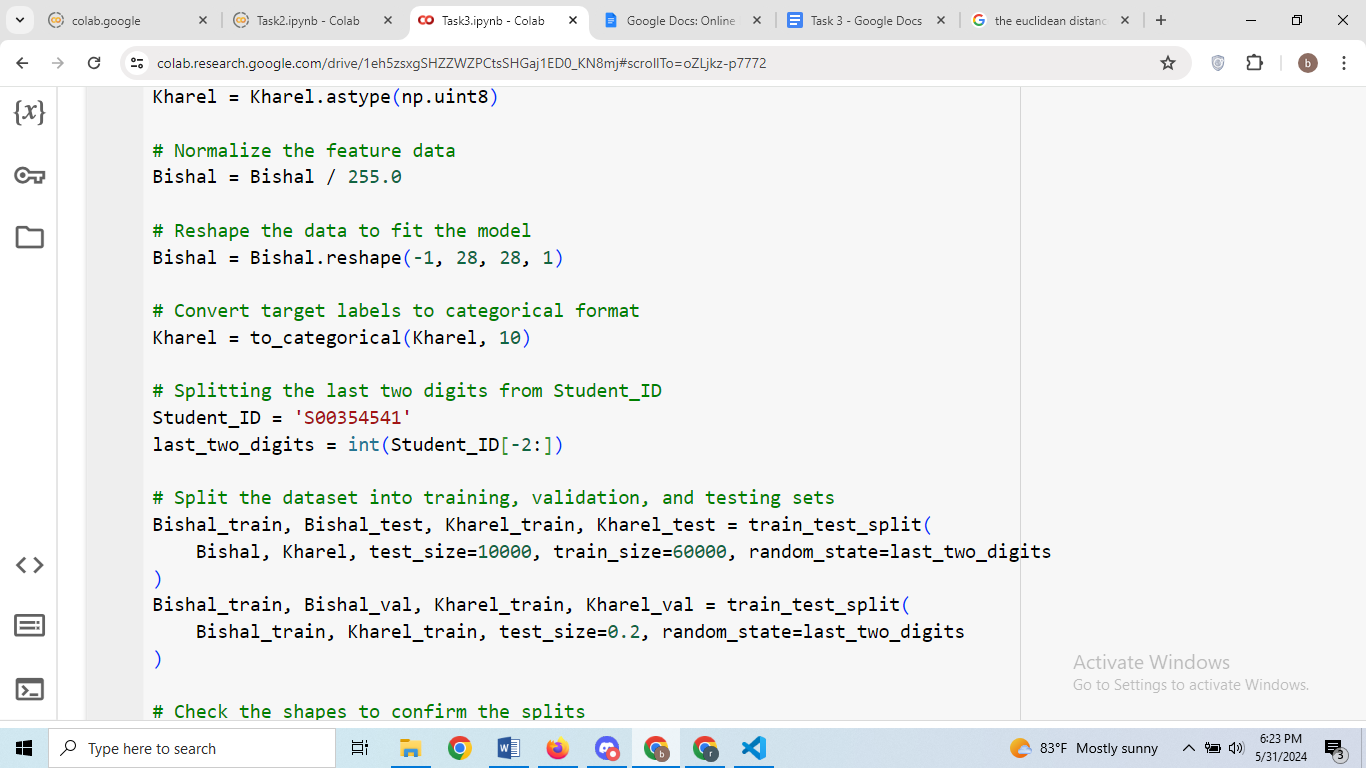
a. Implement on the MNIST dataset a Neural Network for classification using an appropriate framework like TensorFlow or PyTorch.

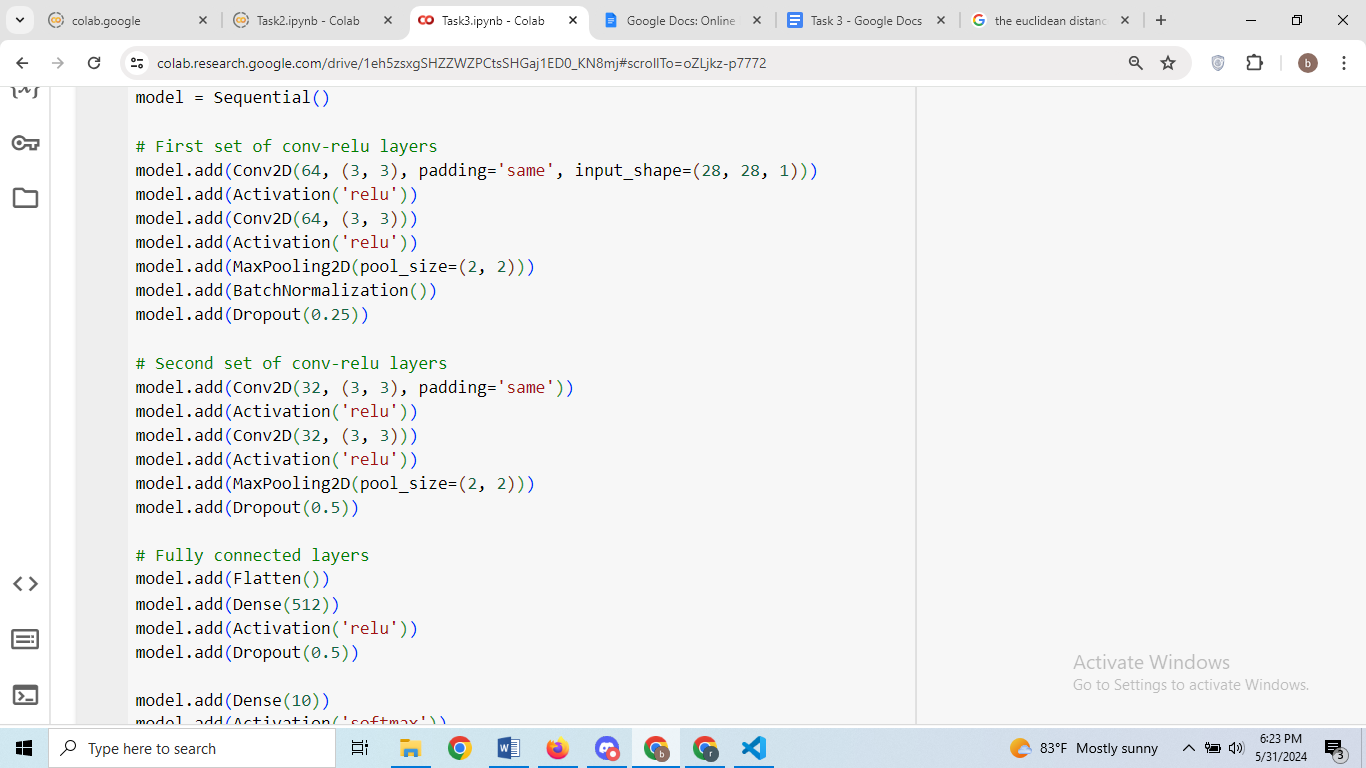
b. Discuss the choice of hyperparameters, and training process.

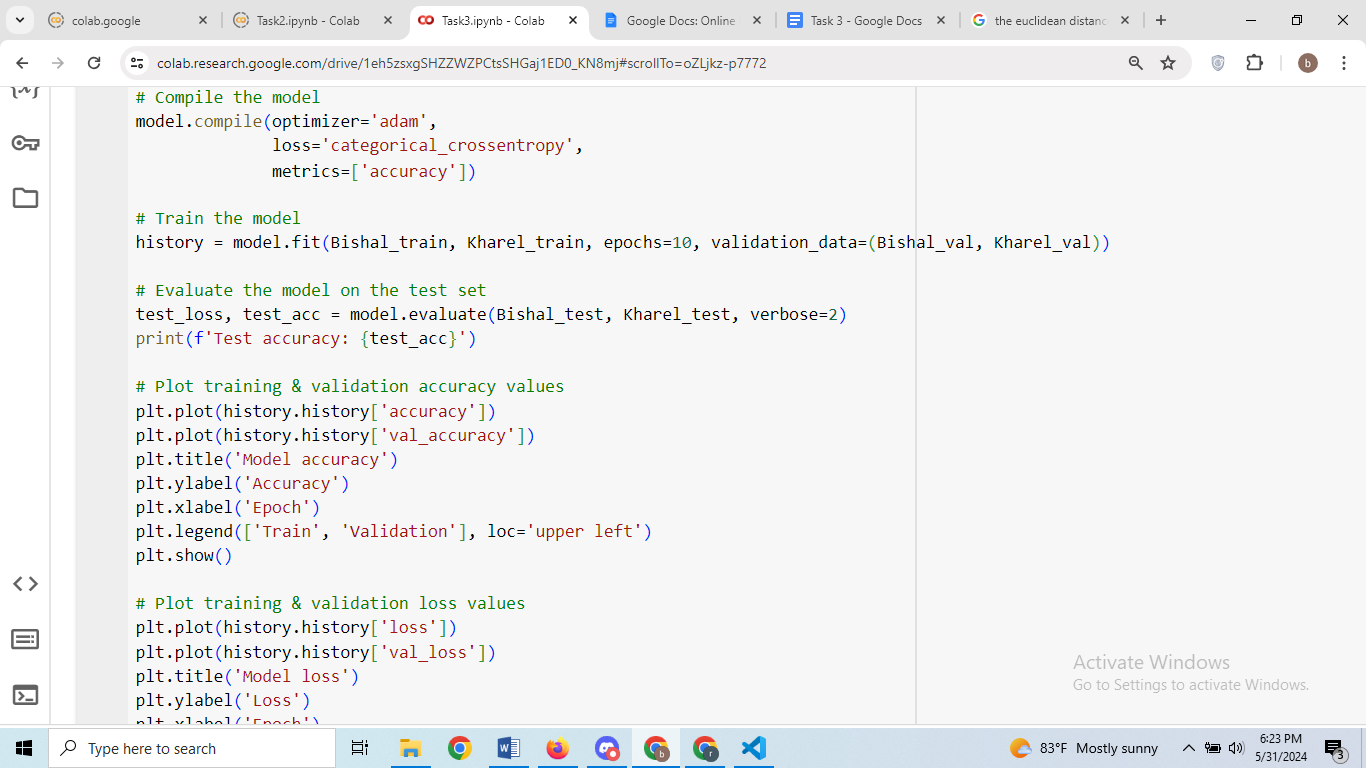
c. Discuss potential biases that can occur in data and model predictions.

**Input:**

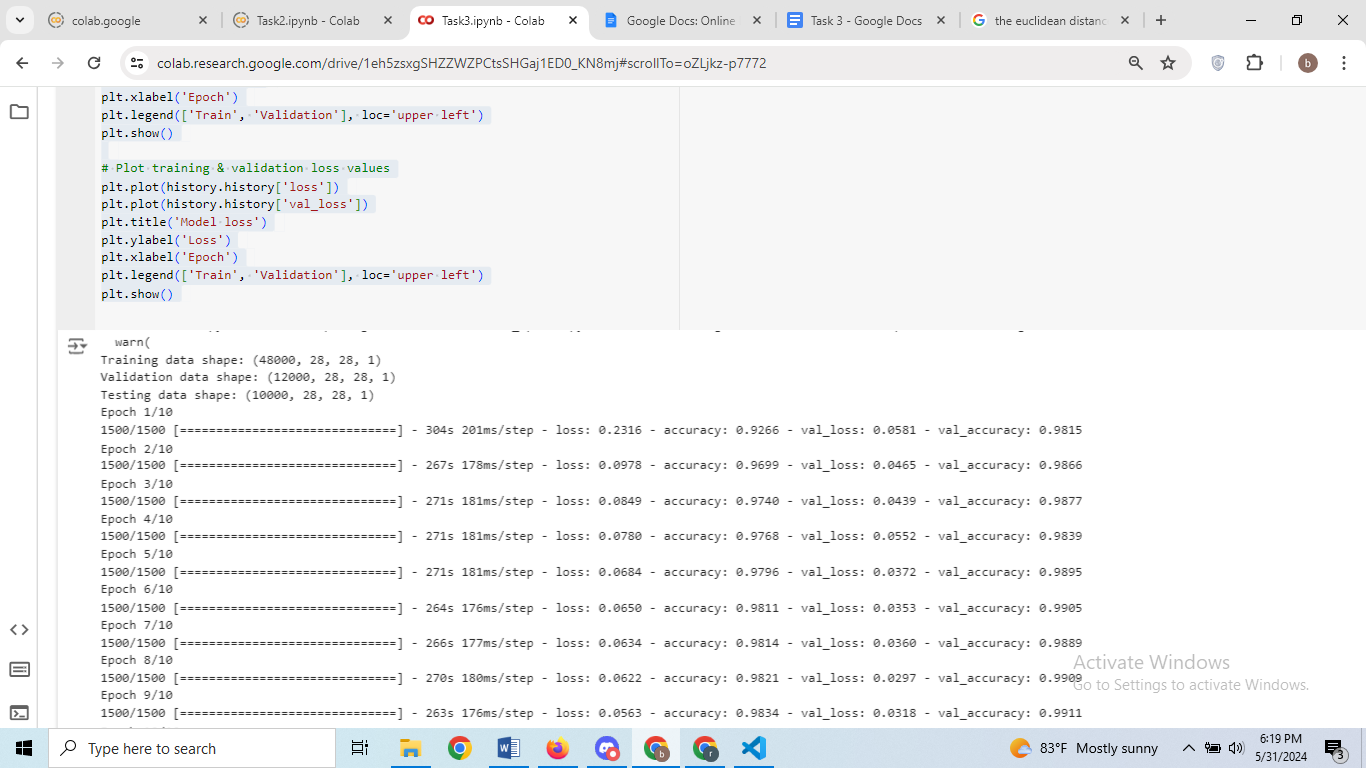
****

****

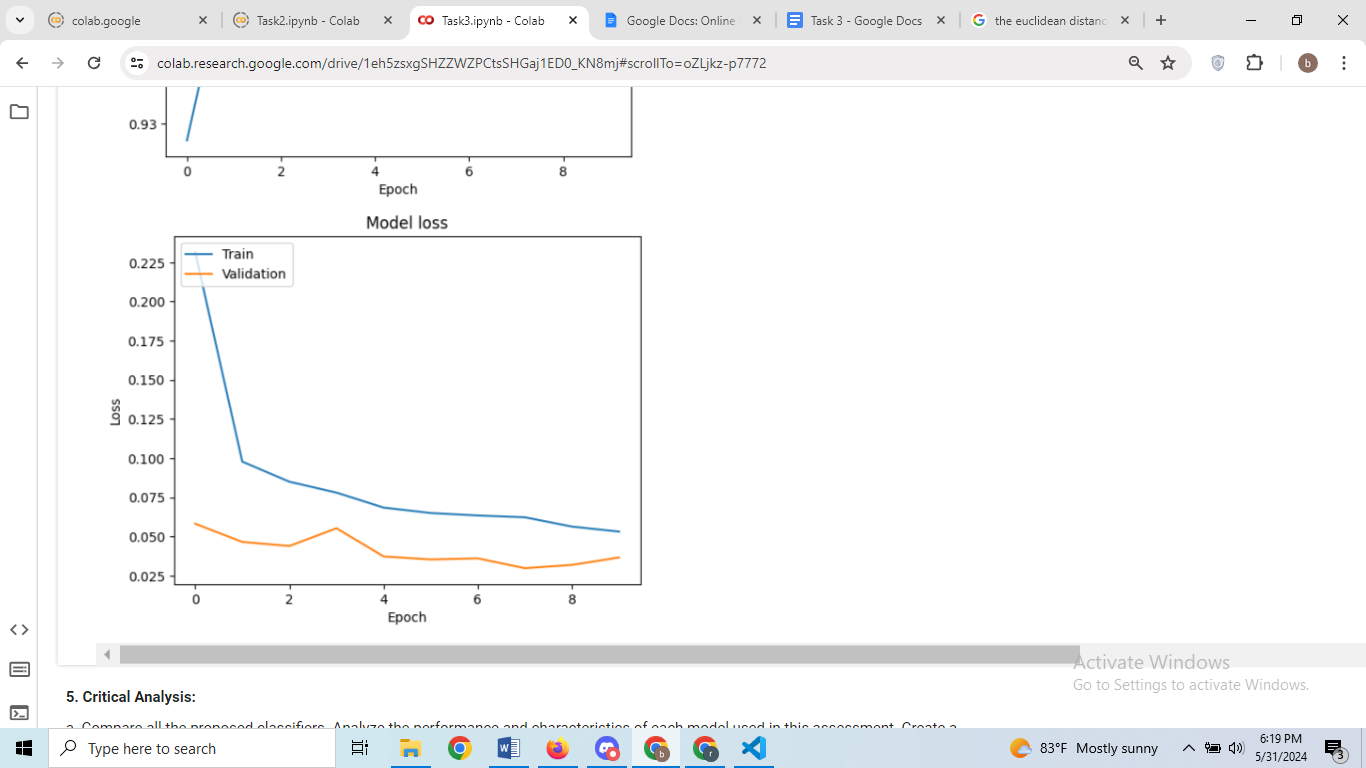
****

****

**Output:**

****

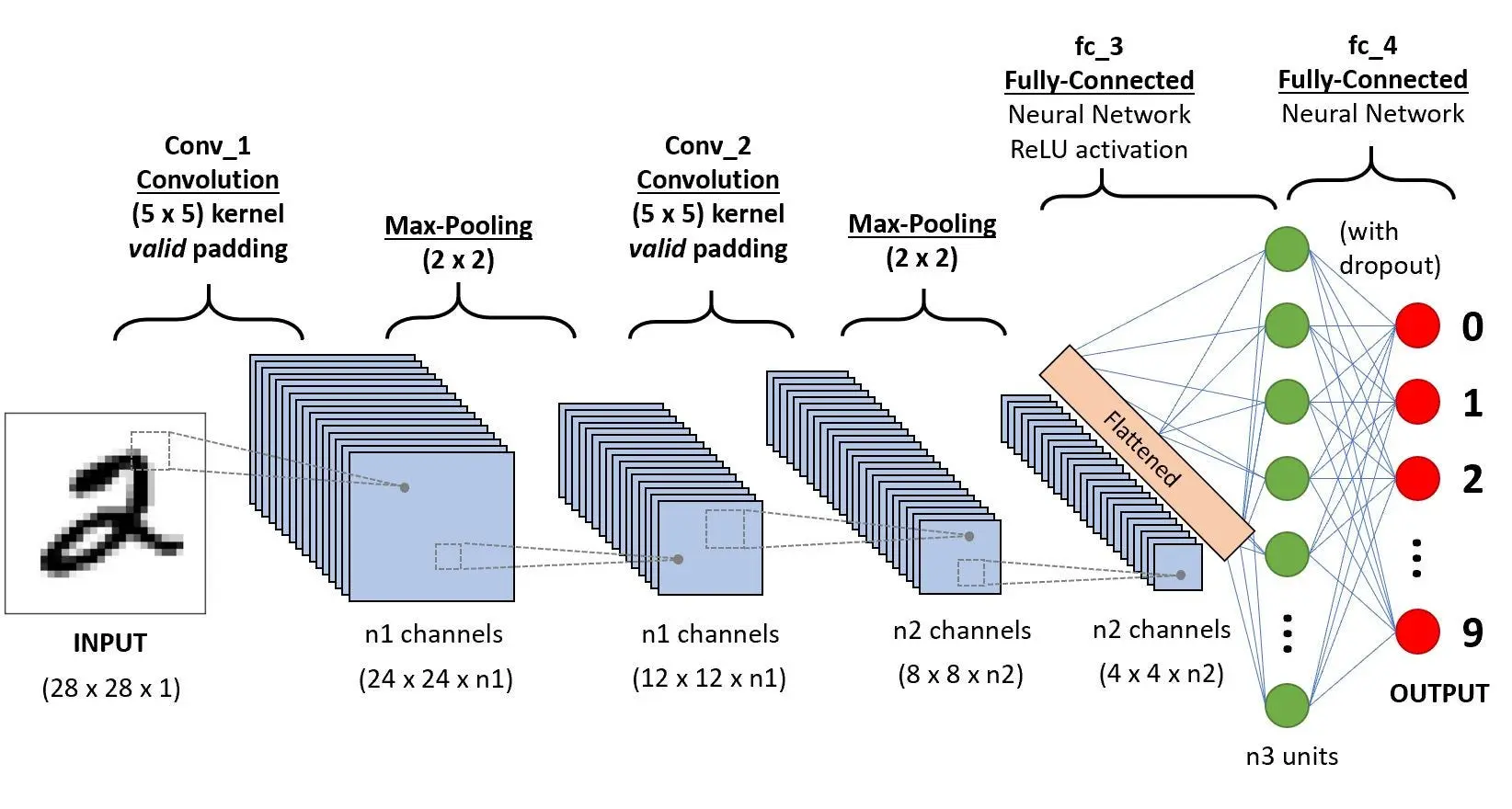
****

****

**(a) Implementing a Neural Network on the MNIST dataset using TensorFlow**

Convolutional Neural Network (CNN) is used to classify the MNIST dataset using TensorFlow and Keras. A CNN is a type of Deep Learning neural network architecture commonly used in Computer Vision.

Working Architecture of CNN:



Convolutional Neural Networks (CNNs) are a type of deep learning model particularly effective for image classification and recognition tasks. Their architecture is designed to automatically and adaptively learn spatial hierarchies of features from input images. The input layer of a CNN receives the raw pixel values of an image, such as a 28x28 pixel grayscale image. Convolutional layers apply filters (or kernels) that slide over the image to extract features like edges, textures, and patterns, producing feature maps. Each convolutional layer is typically followed by an activation function, usually ReLU (Rectified Linear Unit), which introduces non-linearity into the model, enabling it to learn more complex patterns.

Pooling layers, commonly using max pooling, then reduce the spatial dimensions of the feature maps while retaining the most important information, making the network more manageable and less prone to overfitting. Batch normalization layers can be used to normalize the activations, stabilizing and accelerating the training process. Dropout layers are often added to prevent overfitting by randomly setting a fraction of input units to zero during training, which forces the network to learn redundant representations.

After several convolutional and pooling layers, the output is flattened into a single vector, which is fed into fully connected (dense) layers. These layers perform high-level reasoning, with each neuron connected to every neuron in the previous layer. The final fully connected layer uses a softmax activation function to produce a probability distribution over the classes, resulting in the final classification output.

For example, in an MNIST digit classification task, the input layer would receive 28x28 grayscale images. The architecture might include two sets of convolutional and pooling layers followed by batch normalization and dropout layers. This would be followed by a flattening layer and a couple of fully connected layers, with the final layer using softmax activation to classify the images into one of ten digit categories. This structure allows CNNs to effectively learn and recognize patterns in image data, making them the backbone of many advanced image recognition systems.

**Data Preparation:**

* The MNIST dataset is fetched from fetch\_openml.
* The images are normalized by dividing by 255.0.
* The data is reshaped to fit the input shape required by the CNN (28x28x1).
* The labels are converted to categorical format using to\_categorical.

**2. Data Splitting:**

The dataset is split into training (60,000 samples), validation (12,000 samples from training), and test sets (10,000 samples).

**3.Model Design:**

A Sequential model is created.The model consists of two sets of convolutional and pooling layers with ReLU activations and dropout layers for regularization.The final layers are fully connected (dense) layers leading to a softmax layer for classification.

**4. Model Compilation and Training:**

The model is compiled with the Adam optimizer and categorical cross-entropy loss.

The model is trained for 10 epochs with the validation set used for monitoring performance.

**5.Model Evaluation:**

The model is evaluated on the test set to get the test accuracy.

Training and validation accuracy and loss are plotted over epochs.

**b. Discuss the choice of hyperparameters, and training process.**

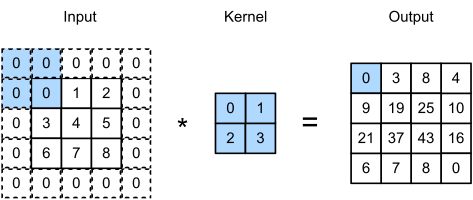
Hyperparameters:

**Number of Filters:**

The first set of convolutional layers uses 64 filters, and the second set uses 32 filters. This choice provides a balance between computational efficiency and model capacity.

**Kernel Size:**

The kernel size is set to (3, 3), a common choice that captures local patterns in the image effectively.



Actually Image is convert in pixel values form (0 to 255) then in our code we have use the 3\*3 kernel which map each to the 3\*3 pixel of image the it extract the feature depending upon the provide pooling. Max pooling or minimum.

**Pooling Size:**

MaxPooling with pool size (2, 2) is used to down sample the feature maps, reducing the spatial dimensions while retaining important information.

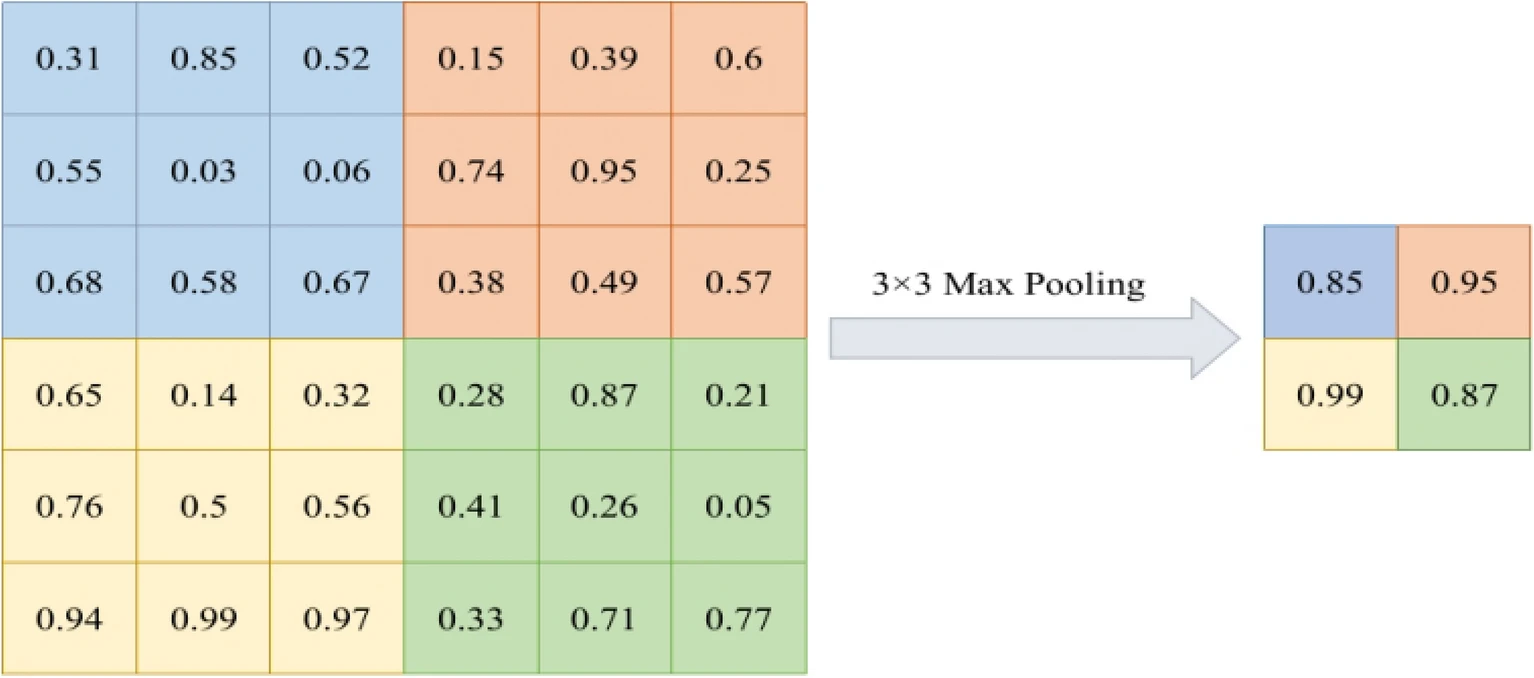
Working Mechanism:

Max pooling is a commonly used feature extraction operation, typically applied in convolutional neural networks. It reduces the spatial dimensions of features by selecting the maximum value within each small window or region. Specifically, for each small window or region, max pooling selects the maximum value and replaces the original data with it. This operation helps to preserve the most prominent features in an image, such as edges and textures. Compared to average pooling, max pooling is better at capturing local feature information. As a result, max pooling29 is widely used in image processing and computer vision tasks.

Math behind it:

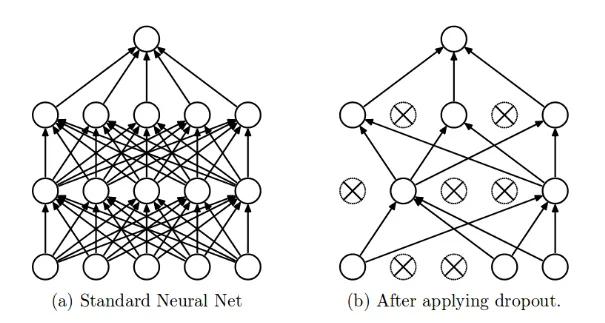
Output(i,j) = max(input(i+m, j+n))

* input(i,j) is the value at position (𝑖,𝑗) in the input feature map.
* (m,n) iterates over the pooling window dimensions.
* output(i,j) is the maximum value within the window applied to the position (𝑖,𝑗) in the input feature map.



**Dropout Rate:**

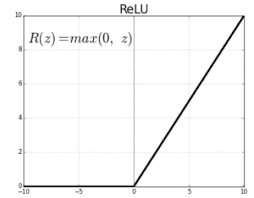
Dropout rates of 0.25 and 0.5 are used after convolutional layers and dense layers, respectively, to prevent overfitting by randomly dropping neurons during training. When output in passed from on layer to another layer in forward propagation dropout rate deactivated when it passes information in backward propagation then dropout function is activated to avoid the extra output which prevent from model overfitting.



**Dense Layer:**

A dense layer with 512 units and ReLU activation is used before the final output layer to capture complex patterns. Dense Layer is fully connected layer of neuron. In this project I have used 512 neuron to train the model.

In task 3 ReLU Activation function has used. ReLU is used to prevent exponential growth in the computation required to operate the neural network. And its derivative values ranges from 0 to 1.It solve the vanishing gradient problem.



From figure We conclude the during back propagation derivative value of function will be either 0 or 1.If 1 out is passed to the next neuron else not.

**Optimizer:**

The Adam optimizer is used for its adaptive learning rate properties, which help in faster convergence.

**Loss** **Function**:

Categorical cross-entropy is used as it is suitable for multi-class classification problems.

**Batch Normalization:**

Added to improve the stability and performance of the network by normalizing the input to each layer.

**Training Process:**

The model is trained for 10 epochs, which is a reasonable starting point for initial training. The number of epochs can be adjusted based on the performance on the validation set.

Validation data is used to monitor the model's performance and prevent overfitting.

The accuracy and loss plots help visualize the training progress and make decisions about potential adjustments in hyperparameters.

**Output Analysis:**

**Training and Validation Performance:**

**Epoch 1:**

Loss: 0.2316

Accuracy: 0.9266

Validation Loss: 0.0581

Validation Accuracy: 0.9815

**Epoch 2:**

Loss: 0.0978

Accuracy: 0.9699

Validation Loss: 0.0465

Validation Accuracy: 0.9866

**Epoch 3:**

Loss: 0.0849

Accuracy: 0.9740

Validation Loss: 0.0439

Validation Accuracy: 0.9877

**Epoch 4:**

Loss: 0.0780

Accuracy: 0.9768

Validation Loss: 0.0552

Validation Accuracy: 0.9839

**Epoch 5:**

Loss: 0.0684

Accuracy: 0.9796

Validation Loss: 0.0372

Validation Accuracy: 0.9895

**Epoch 6:**

Loss: 0.0650

Accuracy: 0.9811

Validation Loss: 0.0353

Validation Accuracy: 0.9905

**Epoch 7:**

Loss: 0.0634

Accuracy: 0.9814

Validation Loss: 0.0360

Validation Accuracy: 0.9889

**Epoch 8:**

Loss: 0.0622

Accuracy: 0.9821

Validation Loss: 0.0297

Validation Accuracy: 0.9909

**Epoch 9:**

Loss: 0.0563

Accuracy: 0.9834

Validation Loss: 0.0318

Validation Accuracy: 0.9911

**Epoch 10:**

Loss: 0.0531

Accuracy: 0.9851

Validation Loss: 0.0366

Validation Accuracy: 0.9897

Final Test Performance:

Loss: 0.0366

Accuracy: 0.9898

Overall, the model shows a significant improvement in both training and validation accuracy across epochs, ending with a high test accuracy of approximately 98.98%. The consistent decrease in validation loss and the high validation accuracy indicate that the model is performing well on unseen data. Using neural network we get more accuracy in model.

**(c) Discussing Potential Biases in Data and Model Predictions**

Potential Biases in Data:

**Class Imbalance:**

If some digits (classes) appear more frequently than others in the training set, the model may become biased towards those classes, leading to poor performance on underrepresented classes.

**Sampling Bias:**

The dataset may not represent all variations of handwriting styles, leading to biased predictions when encountering unfamiliar styles.

**Data Quality:**

If the dataset contains noise or errors (e.g., mislabelled images), the model's predictions may be biased or inaccurate.

**Potential Biases in Model Predictions:**

**Overfitting:**

If the model is too complex, it may overfit to the training data and fail to generalize to new, unseen data, leading to biased predictions on the test set.

**Training-Validation-Test Split:**

The way data is split can introduce biases. If the random split does not properly represent the overall distribution, it can affect model performance.

**Feature Representation:**

The choice of feature representation (e.g., normalizing pixel values) can introduce biases if it does not capture essential aspects of the data relevant for classification.

**Algorithmic Bias:**

The model may learn biases present in the training data, such as systematically misclassifying certain digits due to their representation in the dataset.

Mitigation Strategies:

Ensure balanced representation of all classes in the training data.

Use data augmentation techniques to increase the diversity of training samples.

Regularly evaluate the model on a diverse validation set to monitor and address biases.

Incorporate techniques like cross-validation to ensure the model generalizes well.

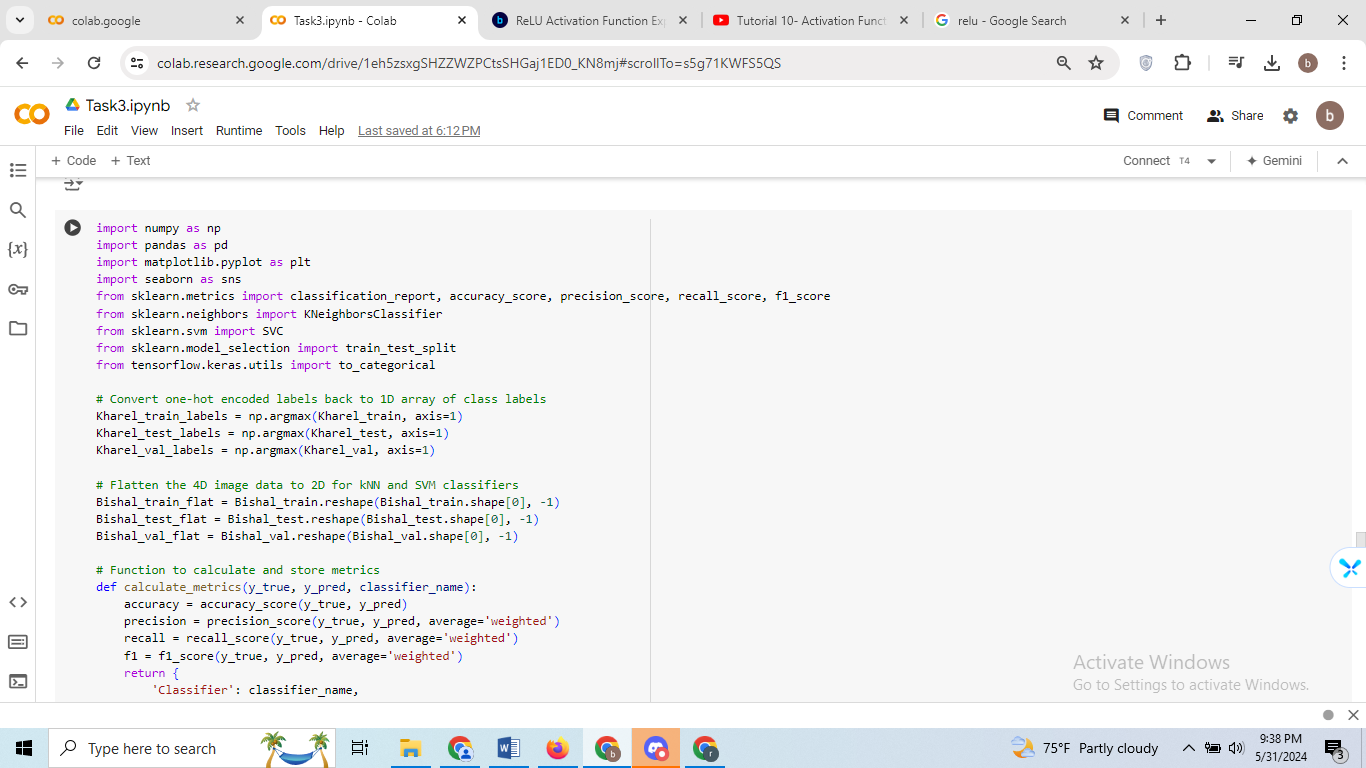
Continuously monitor model performance and retrain with updated data to minimize biases.

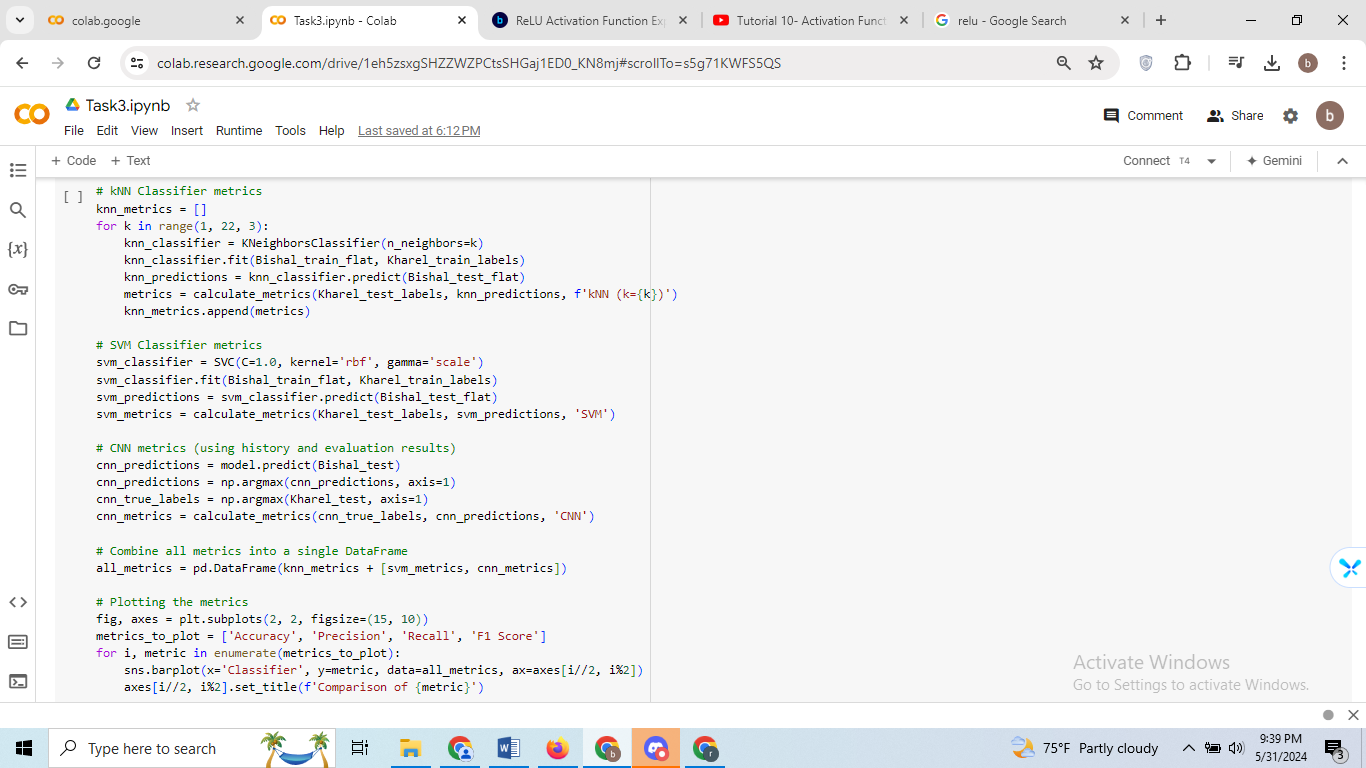
By being aware of these biases and implementing strategies to mitigate them, the reliability and fairness of the neural network's predictions can be improved.

**5. Critical Analysis:**

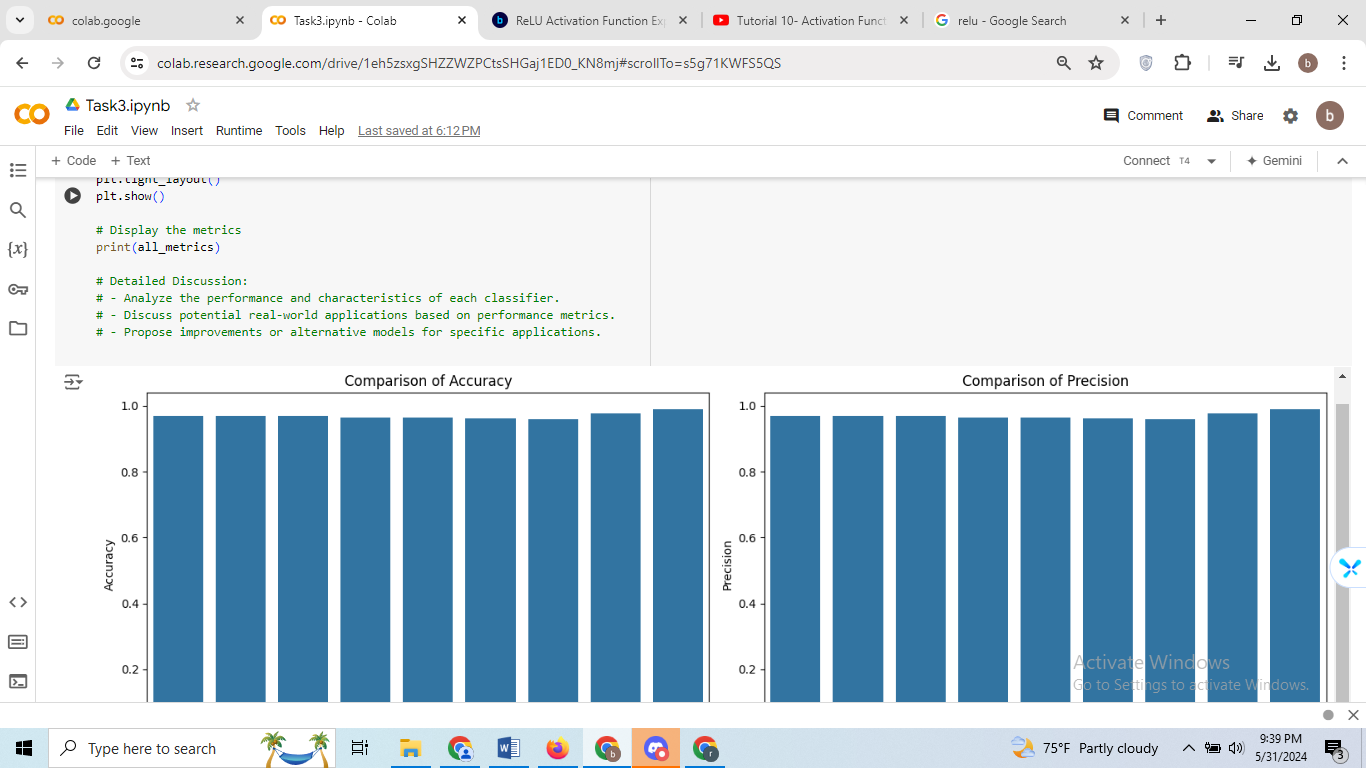
a. Compare all the proposed classifiers. Analyze the performance and characteristics of each model used in this assessment. Create a comprehensive visualization that compares their accuracies, precision, recall, and F1 scores. Discuss the potential real-world applications of each model based on their performance metrics and propose improvements or alternative models for those applications.

**Input:**

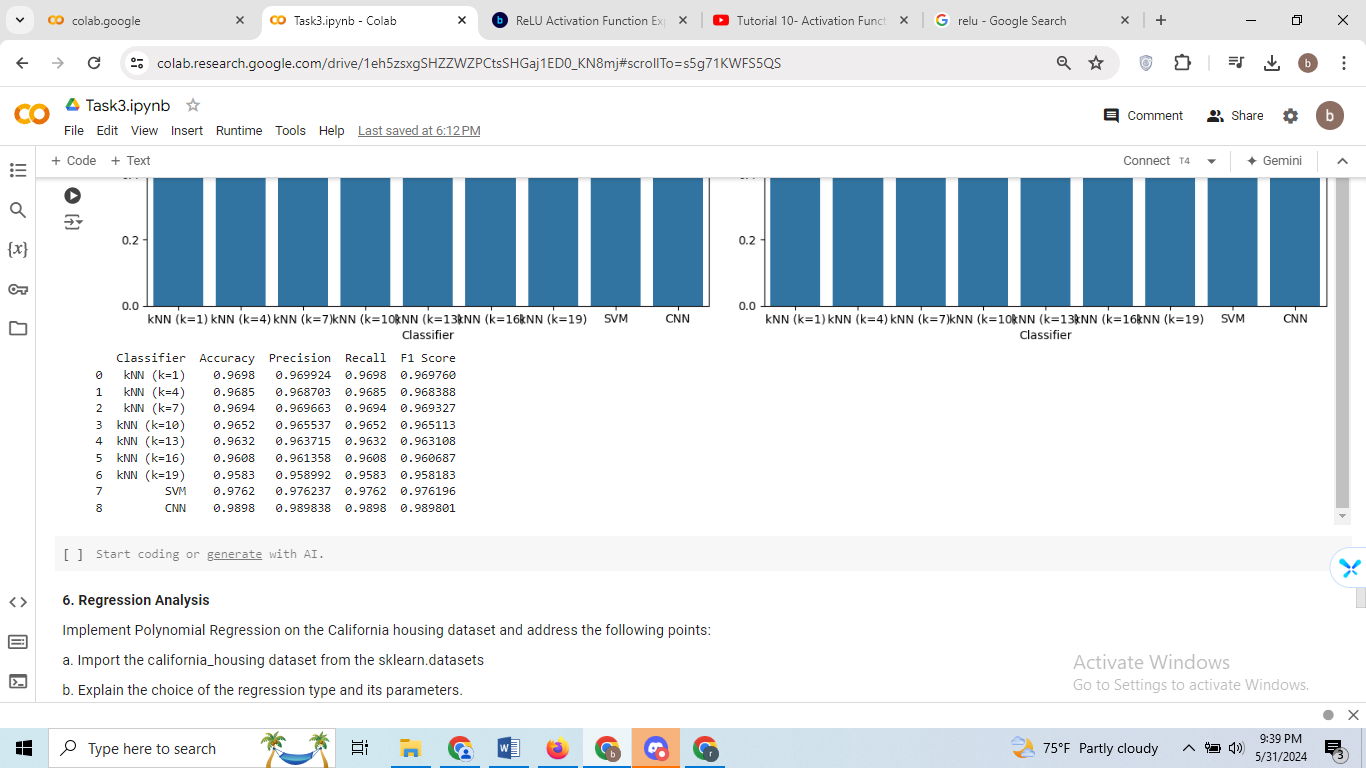
****

****

**Output:**

****

****

****

**k-Nearest Neighbors (kNN)**

**Potential Real-World Applications:**

kNN is particularly useful in scenarios where simplicity and interpretability are key, and quick approximate solutions are sufficient. Some applications include:

* Recommendation Systems: kNN can be employed to recommend products or content based on the similarity to items previously liked by the user.
* Pattern Recognition: Basic tasks such as handwriting recognition or simple image classification can benefit from kNN due to its straightforward approach.

**Proposed Improvements:**

* Weighted kNN: Using a weighted kNN, where closer neighbors have more influence on the classification, can improve accuracy.
* Dimensionality Reduction: Applying techniques like Principal Component Analysis (PCA) can reduce the complexity and improve performance by focusing on the most significant features.
* Hybrid Models: Combining kNN with other models or algorithms can leverage its strengths while compensating for its weaknesses.

**Support Vector Machine (SVM)**

**Potential Real-World Applications:**

SVM is highly suitable for applications that require high accuracy and the ability to handle high-dimensional data, including:

* Medical Diagnosis: SVM can be used for detecting diseases from medical images or datasets due to its robustness and reliability.
* Fraud Detection: Its ability to create precise decision boundaries makes SVM effective in distinguishing fraudulent activities from legitimate ones.

**Proposed Improvements:**

* Kernel Exploration: Experimenting with different kernels such as polynomial or sigmoid can enhance the model's ability to capture complex relationships.
* Hyperparameter Tuning: Fine-tuning parameters like the regularization term (C) and kernel coefficient (gamma) can further optimize performance.
* Ensemble Methods: Using an ensemble of SVMs trained on different subsets of data or with different parameters can improve overall accuracy and robustness.

**Convolutional Neural Network (CNN)**

**Potential Real-World Applications:**

CNNs excel in tasks involving complex image and video data, making them ideal for:

* Facial Recognition: CNNs can accurately identify and verify individuals in security and authentication systems.
* Object Detection: In autonomous vehicles, CNNs are crucial for detecting and classifying objects in real-time.
* Medical Imaging: CNNs are effective in diagnosing conditions from MRI scans, X-rays, and other medical images by learning intricate features and patterns.

**6. Regression Analysis**

Implement Polynomial Regression on the California housing dataset and address the following points:

a. Import the california\_housing dataset from the sklearn.datasets

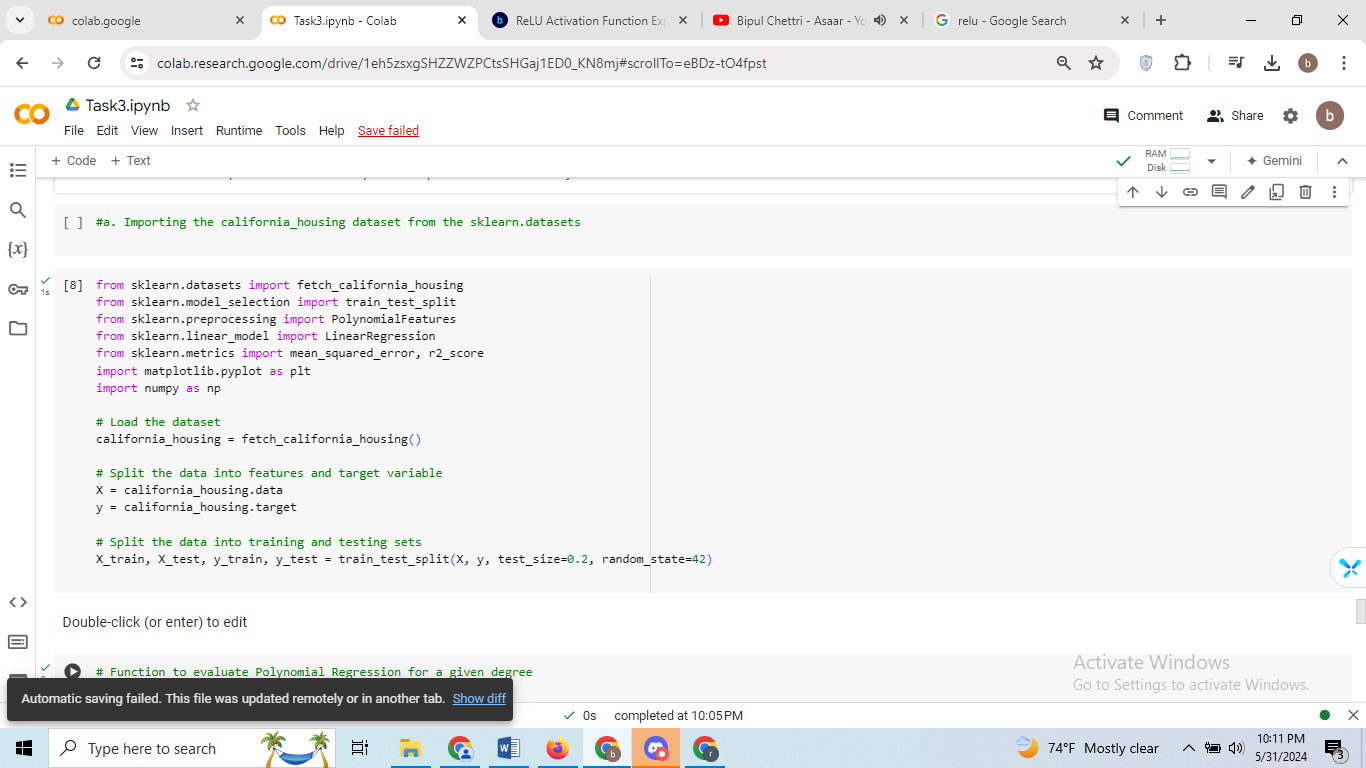
b. Explain the choice of the regression type and its parameters.

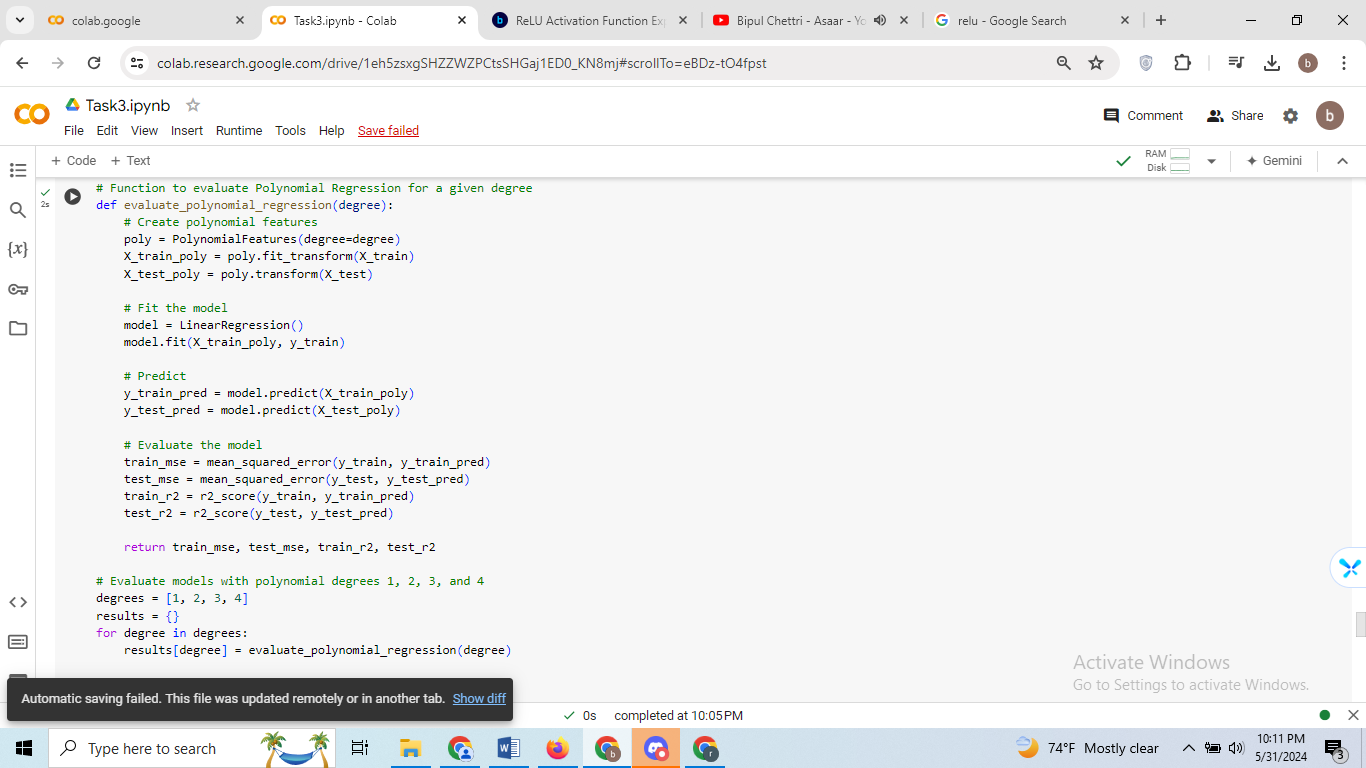
c. Evaluate its performance and discuss the results in the context of the chosen model.

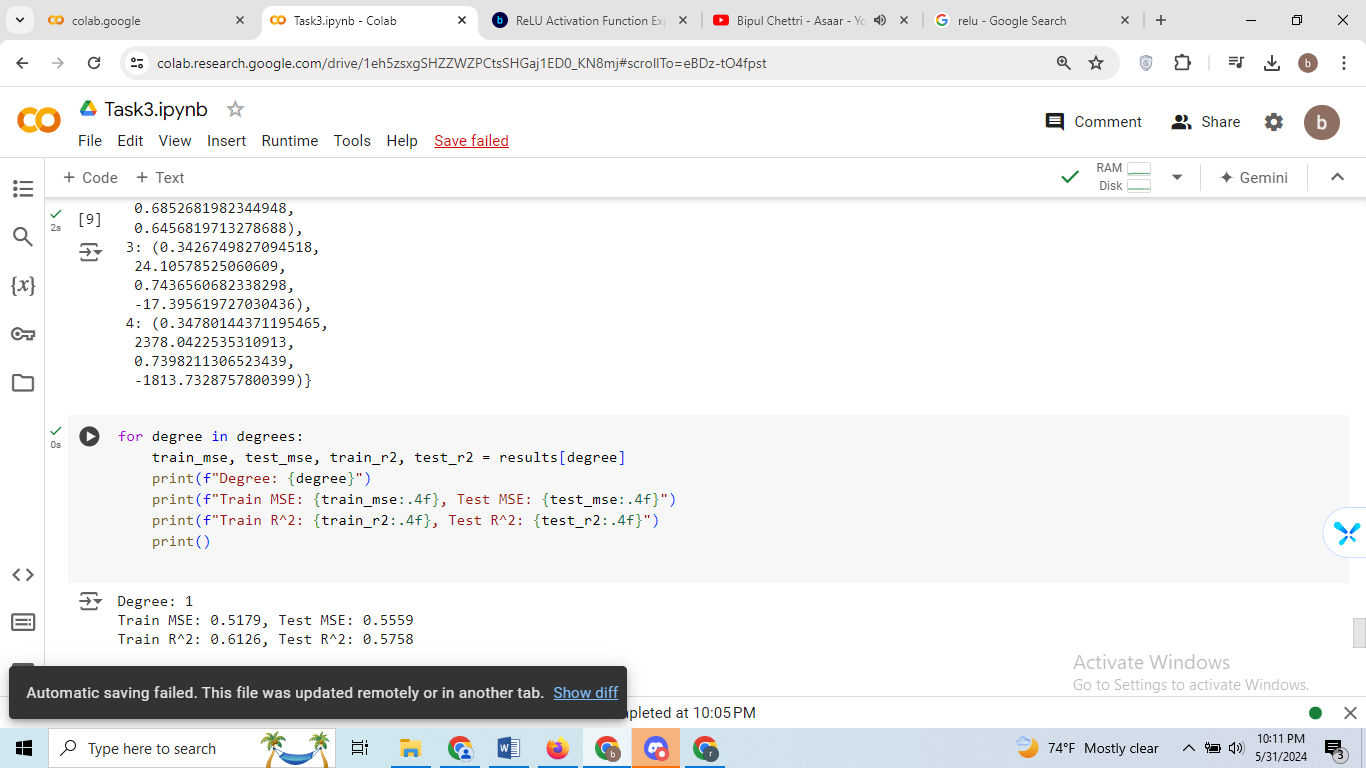
d. Analyze the impact of different polynomial degrees on the model's performance.

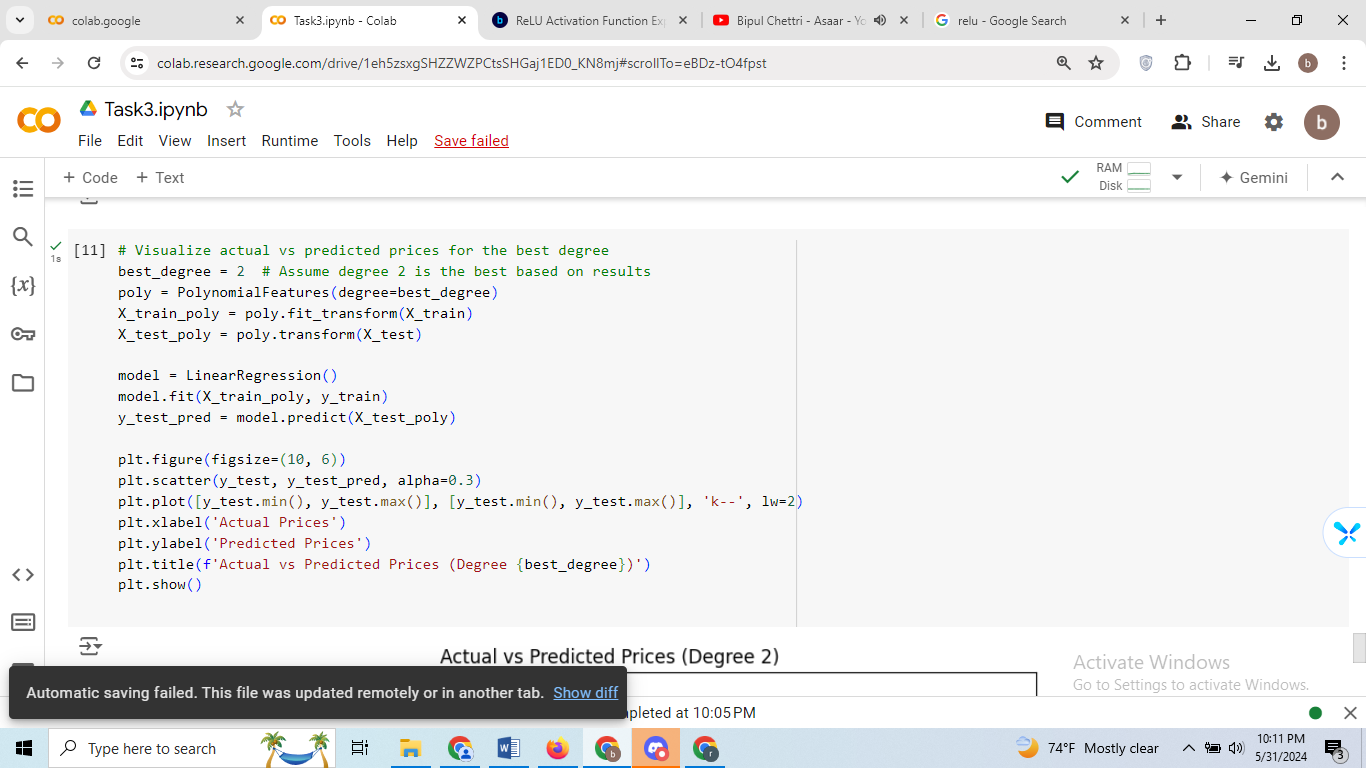
e. Visualize the relationship between actual vs. predicted prices and discuss any visible trends or anomalies

**Input:**

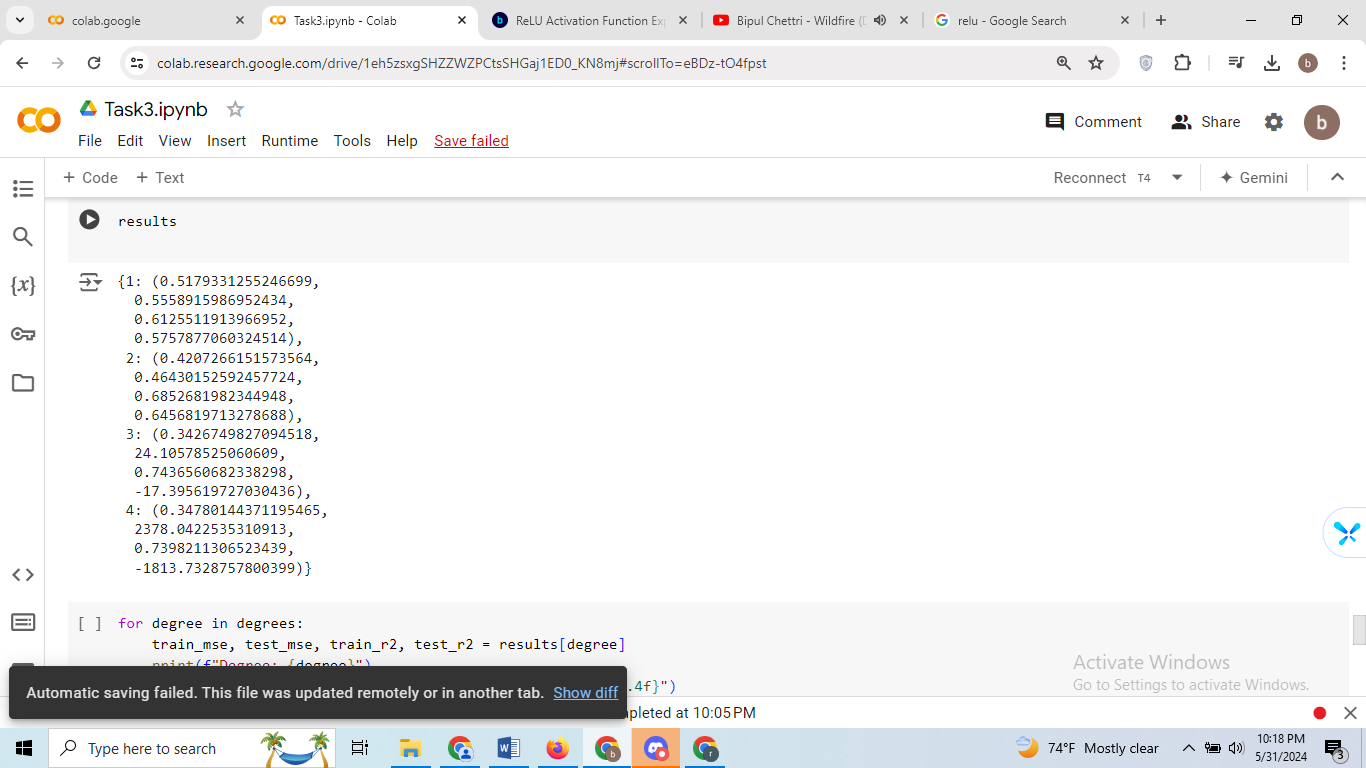
****

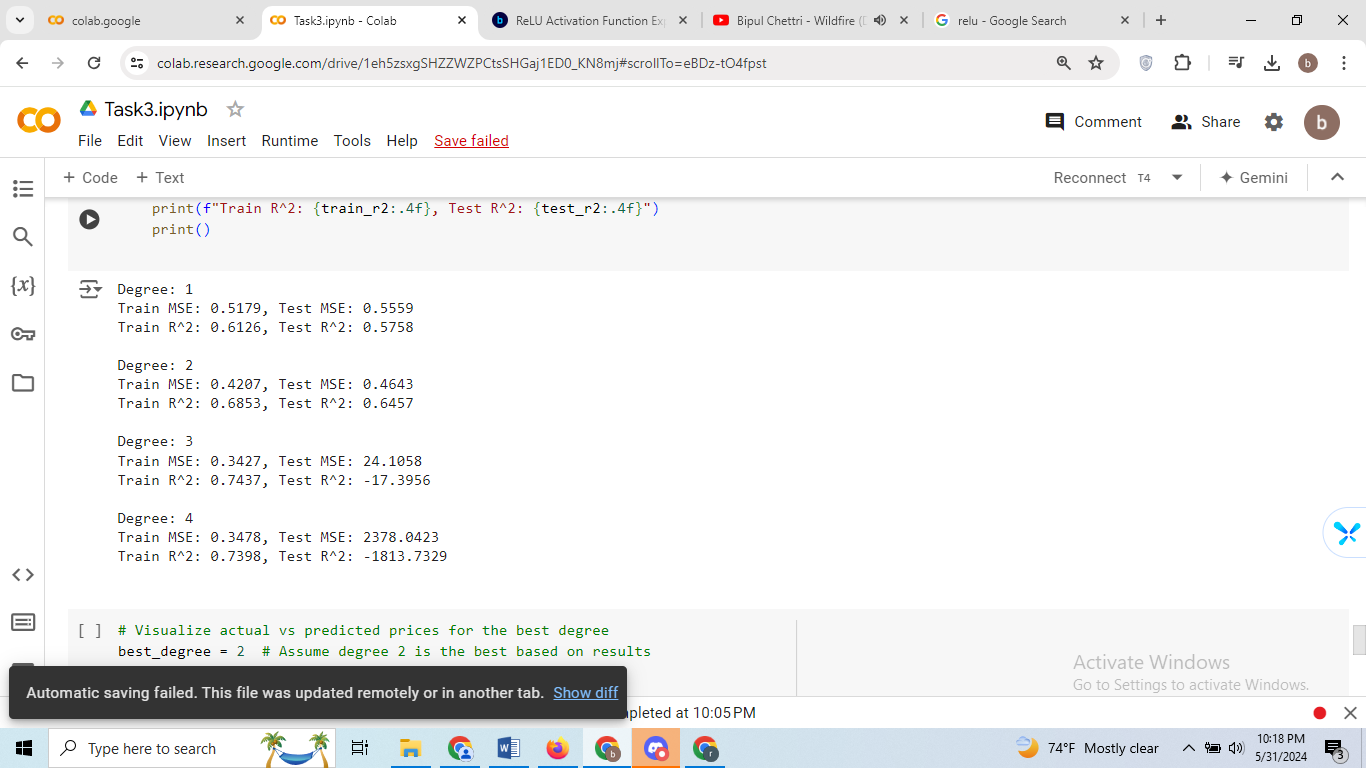
****

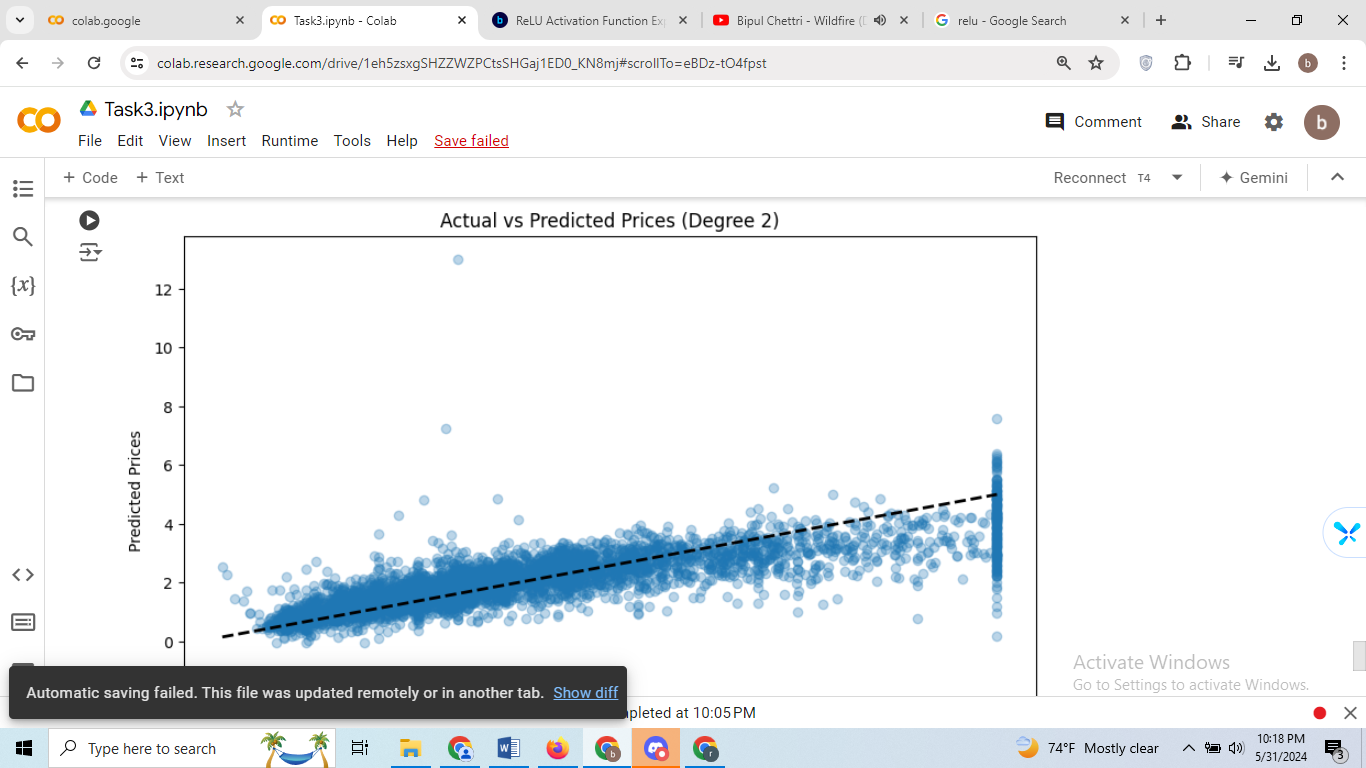
****

****

**Outputs:**

****

****

****

**Output Analysis:**

**Degree 1:**

* Train MSE: Mean Squared Error (MSE) on the training set is 0.5179.
* Test MSE: MSE on the testing set is 0.5559.
* Train R^2: R-squared (R^2) score on the training set is 0.6126, indicating that the model explains 61.26% of the variance in the training data.
* Test R^2: R^2 score on the testing set is 0.5758, suggesting that the model performs reasonably well on unseen data, explaining 57.58% of the variance.

**Degree 2:**

* Train MSE: MSE on the training set is 0.4207.
* Test MSE: MSE on the testing set is 0.4643.
* Train R^2: R^2 score on the training set is 0.6853, indicating that the model explains 68.53% of the variance in the training data.
* Test R^2: R^2 score on the testing set is 0.6457, indicating that the model explains 64.57% of the variance in the testing data.

**Degree 3:**

* Train MSE: MSE on the training set is 0.3427.
* Test MSE: MSE on the testing set is 24.1058.
* Train R^2: R^2 score on the training set is 0.7437, indicating that the model explains 74.37% of the variance in the training data.
* Test R^2: R^2 score on the testing set is -17.3956. This negative R^2 score suggests that the model performs worse than a horizontal line (the mean of the target variable) in predicting the testing data. It indicates severe overfitting.

**Degree 4:**

* Train MSE: MSE on the training set is 0.3478.
* Test MSE: MSE on the testing set is 2378.0423.
* Train R^2: R^2 score on the training set is 0.7398, indicating that the model explains 73.98% of the variance in the training data.
* Test R^2: R^2 score on the testing set is -1813.7329. Similar to degree 3, this negative R^2 score suggests severe overfitting, indicating that the model performs worse than a horizontal line in predicting the testing data.

Overall, we observe that the model with degree 2 performs the best, achieving the lowest MSE and highest R^2 score on the testing set. Models with degrees 3 and 4 suffer from severe overfitting, as evidenced by the significantly higher MSE and negative R^2 score on the testing set.