Slide division:

# Stephen: 1-4

Hello everyone, today I'm excited to present our machine learning project, which is centered around digit recognition. We participated in the Kaggle competition called "Digit Recognizer" which uses the well-known MNIST dataset. Our main objective was to create a model that could accurately recognize handwritten digits. Let's dive into the process we followed.

The MNIST dataset is comprised of 42,000 training images and 28,000 test images, with each image being a handwritten digit represented as a 28x28=784 pixel image. Our task was to predict the corresponding digit for each image in the test dataset using the labeled data in the training dataset.

We began with exploratory analysis to discover insights from the MNIST dataset, which would help inform our modeling process. This involved visualizing sample digits to understand data representation, analyzing digit class distribution using histograms, and studying digit intensity distribution. We also specifically analyzed the digit "4" representation and applied PCA to reduce dimensionality while extracting essential features.

For the modeling phase, we followed a structured pipeline. First, we imported libraries and loaded the data. Next, we preprocessed and visualized the data. To reduce dimensionality, we performed PCA, and then used GridSearchCV to find the best hyperparameters for our RandomForestClassifier. After evaluating the classifier and plotting feature importances, we predicted labels for the test set and created a submission file. Lastly, we visualized test images and displayed the predicted labels.

We also made use of the RACI framework to assign roles and tasks within our project effectively. This helped us streamline our work process and ensure everyone on the team had clear responsibilities.

One of the main challenges we faced during this project was the high dimensionality of the feature space. To mitigate the effects of the curse of dimensionality, we applied PCA to reduce the feature space. Additionally, we employed regularization techniques to prevent overfitting and ensure the model generalizes well to new, unseen data.

Overall, by conducting a thorough exploratory analysis and following a structured pipeline for modeling, we were able to develop a machine learning model that can accurately predict handwritten digits.

Now, let me discuss one of our key exploratory analysis steps: visualizing the dataset. In slide a.1|2, we focus on the size of the dataset and the presence of missing values.

To begin with, we displayed the size of both the training and test sets. This helped us understand the amount of data available for building and evaluating our model. During this phase, we also checked for any null or missing values in the dataset. Ensuring data integrity is crucial in building reliable models, so detecting and addressing missing values, if any, would have been a necessary step.

Furthermore, we made use of the matplotlib library to display the digits as grayscale images. By reshaping the pixel values into 28x28 images, we were able to visualize how the handwritten digits appeared. This process gave us a deeper understanding of the data representation, allowing us to better inform our preprocessing and modeling decisions.

# Jennifer: 5-7

**Slide 5**

In the same framework of data exploration was necessary determinate how were distributed the different instances(digits) among the 10 labels.

* Value\_counts counted the number of instances in each of the 10 labels of the training set. Then, the classes were sorted in ascending order using sort\_index(), and a histogram with 11 bins was created to visualize the distribution = instances are well balanced,

Aspects as the intensitive are key aspect for those kind of this computer vision problems ( "1" would have less intensity on average to a complexer digits)

* The intensity average of all the instances in the training set was calculated and this information was added as a new column to the training set. train['intensity'] = train.iloc[:, 1:].mean(axis=1)
* Then, the dataset was grouped in each digit label and the mean intensity for each group was computed. intbylabel = train.groupby('label')['intensity'].mean().reset\_index()
* These averages were plotted in a bar plot to visualize the level of intensity by each label.

**intensity may be a tool to predict numbers since some labels have higher or lower brightness ( example 0 -1) however people have different ways of writing digits so we can create a graph that shows the distribution of the average intensity based on the label of each digit.**

**Slide 6**

* Grouped the training set by digit label and computes the mean intensity for each group, storing the results in a new Pandas series called intbylabel
* (KDE) plot is created to show the distribution of intensity values for each digit label in the training set. The instances in the training set that correspond to each label are selected, and the Seaborn library's 'kdeplot' function is used to create a density plot for each label. The resulting plots are displayed on a shared plotting area, with each plot labeled with its corresponding digit label.
* ​​We used GridSpec and subplots to create an array of individual plots for each label.

Some digits have higher variability than others. digit "4" has less normal distribution (due to different ways of writing) it, so we decided to visualize images of digit 4  **were necessary deepen in the analysis of this digit.**

**SLIDE 7**

* We created an exclusive sub-data set called 'train4' which contained the instances from the rows from the training set that belonged to label number 4. Then, we created a 3x3 matrix which contained 9 images from the 'train4' and our loop iterate through each subplot and displayed an image of the number 4 in the corresponding subplot. Each image was retrieved from the 'train4' dataframe as a matrix of pixels, converted into a 28x28 pixel matrix, and horizontally flipped using the 'flip' function. The image was then displayed in the subplot using the 'imshow.

**Some people write the number 4 by "closing" the digit, while others write it leaving the upper part open. This explains the variability of this dig**it

# Carolina: 8, 17,18,19

**Slide 8**

We performed some data exploration techniques on our dataset, starting with the calculation of a correlation matrix to measure the linear relationship between pixel values. We then created a heatmap to visualize this matrix and gain insight into the patterns of pixel values.

To reduce dimensionality and extract the most important features, we applied PCA to our preprocessed data. The reduced data was then plotted on a scatterplot in which each dot represented a digit, and the color indicated its corresponding label.

The variance explained by the two principal components was found to be 0.169, suggesting that they only captured a small amount of the overall variability in the data. Despite this, the insights gained from the PCA were valuable in helping us understand the underlying patterns of the data.

**Slide 17**

In this slide, we're showcasing the process of visualizing and displaying predicted labels from our machine learning model. We start by loading the submission data and extracting the first ten predicted labels. We then print these labels horizontally and visualize the corresponding ten test images.

Now, what makes this process interesting is that sometimes, our model can misclassify certain images. For example, it might predict a 0 as a 4. This is where model evaluation and improvement come into play. By analyzing such misclassifications, we can fine-tune our model and improve its accuracy.

**Slide 18**

We achieved a cross-validation accuracy of 0.9416 using PCA and the RandomForestClassifier algorithm. However, we observed some misclassifications, such as "0" being classified as "4", indicating that there is still room for improvement. This was explained by the explanatory analysis.

To enhance accuracy, we suggest experimenting with deep learning models, such as TensorFlow. This approach may also optimize the processing time of the overall process.

We utilized GridSearchCV to optimize RandomForestClassifier's hyperparameters, which resulted in an effective model. However, this process required substantial computing time, highlighting the need for striking a balance between model efficiency and computational resource requirements.

During our project, we encountered challenges in digit recognition due to the reliance on pixel brightness and the complexity of preprocessing the data.

We learned that data augmentation can be a powerful tool for improving the accuracy and performance of a machine learning model. It's essential to experiment with different techniques to find the most effective approach.

# Mario: 9-12

Slide 9:

In this project, as first step, we imported necessary libraries such as NumPy, pandas, matplotlib, and scikit-learn. Afterward, we loaded the train and test datasets using pandas' read\_csv function.

Next, we moved on to data preprocessing. We splitted the data into features (X\_train) and labels (y\_train), where the features contain pixel values of the images and the labels represent the actual digits. We then utilized two visualization functions, which helped us gain a better understanding of the data and ensure that it is prepared correctly for training our machine learning model.

Slide 10:

The first function, visualize\_dataset, visualizes the entire MNIST dataset by concatenating digit images into rows of 200 and stacking these rows vertically. The resulting grid is displayed using matplotlib. This function is sourced from an external GitHub repository.

The second function, visualize\_images, is inspired by the first one, but we have created our own version to visualize a grid of digit images from the MNIST dataset with a specified number of rows and columns. This function reshapes the data and plots the images in a grid format using matplotlib.

To visualize the images from the training dataset, we call the visualize\_images function with X\_train as the input data and specify 3 rows and 4 columns.   Slide 11:

In this part of our digit recognizer project, we normalized the data by dividing each pixel value by 255. This scales the pixel values to a range of 0 to 1, which helps improve the performance of our machine learning model.

Next, we performed Principal Component Analysis (PCA) to reduce the dimensionality of our dataset while retaining significant components. This step enabled faster and more efficient experimentations.

We fitted PCA with the normalized training data and then created a plot showing the cumulative explained variance ratio versus the number of principal components. To capture 95% of the variance in the dataset, we looked at the plot and found the point where the y-value is around 0.95. In our case, this corresponds to 200 components. Just remember that in our case PCA components can be adjusted to "100" instead of "200" in order to decrease the execution time. Generally, this modification results in a loss of information or variance derived from the original data. However, in our specific case, the loss is not particularly substantial.

Slide 12:

In the fourth step, we used GridSearchCV to find the best hyperparameters for the RandomForestClassifier. This process systematically compares different hyperparameters to improve categorization accuracy.

Before proceeding, we should note that when max\_depth is set to None in scikit-learn's RandomForestClassifier, the trees are allowed to grow fully. Although deeper trees can potentially overfit the training data, this risk is reduced in the case of a random forest, as it combines the results of multiple trees.

So, we proceed to define a hyperparameter grid to search for the optimal parameters for the RandomForestClassifier. The grid included various combinations of n\_estimators, max\_depth, min\_samples\_split, and min\_samples\_leaf. We then created a GridSearchCV object, performed a 5-fold cross-validated grid search using 4 concurrent jobs, and fitted it to the data.

# Riccardo: 13-16

In our approach to finding the best hyperparameters for our RandomForestClassifier and evaluating its performance, we used GridSearchCV to systematically determine the optimal hyperparameters. The goal was to compare the effectiveness of various hyperparameter combinations and improve our model's accuracy.

We defined a range of hyperparameters to test and then created a grid search object and fit it to the PCA-reduced training data. The grid search took around 9,534 seconds, or approximately 2 hours and 39 minutes, to find the optimal model and its parameters. While this process was time-consuming, it was crucial to ensure that we chose the best hyperparameters for our RandomForestClassifier.

We were able to identify the best parameters, which included no maximum depth, a minimum samples leaf of 1, a minimum samples split of 5, and 150 estimators. The "None" value for the 'max\_depth' parameter allowed the tree to grow without constraint, which led to better generalization due to the aggregation of multiple trees in a random forest model.

1. **max\_depth**: This hyperparameter controls the maximum depth of each tree in the random forest. When set to "None," it allows the tree to grow without constraint, meaning the tree can continue growing until all leaves are pure or contain fewer samples than the specified min\_samples\_split. Allowing the trees to grow without constraint might lead to overfitting in some cases, but in the context of a random forest model, the risk is reduced as the aggregation of multiple trees leads to better generalization.
2. **min\_samples\_leaf**: This hyperparameter determines the minimum number of samples required for a leaf node in each tree. Setting it to 1 means that even a single sample is enough for a leaf node to exist. This can create a more flexible model, capturing finer patterns in the data.
3. **min\_samples\_split**: This hyperparameter defines the minimum number of samples required to split an internal node in each tree. In our case, it is set to 5, meaning that at least 5 samples must be present in a node for it to be split. This parameter can help control the model's complexity and prevent overfitting.
4. **n\_estimators**: This hyperparameter sets the number of trees in the random forest. In our case, it is set to 150, which means the random forest model consists of 150 decision trees. Increasing the number of trees can improve the model's performance and stability, as the predictions are aggregated from a larger number of individual trees. However, adding more trees can also increase computational requirements and training time.

To evaluate our classifier, we calculated the cross-validation score, with our model achieving a score of 0.9416, demonstrating strong performance. We also plotted the feature importances after PCA to gain insights into the model, which could inform potential improvements or alternative algorithm choices.

The bar chart displayed the principal component indices and their importances, with every 10th index component shown. The chart revealed that upon reaching the 100th principal component, the importance of the subsequent components diminished considerably. This flattening of the plot indicated that these components had a lower relative importance, supporting our decision to use PCA for reducing the dimensionality of our dataset.

With our optimized classifier, we proceeded to predict labels for the test set. We applied PCA to the test data, predicted the labels using our best model, and then created a submission file containing our predictions. This submission file allowed us to evaluate the categorization accuracy, fulfilling the competition's objective.

In conclusion, our approach to finding the best hyperparameters for the RandomForestClassifier and evaluating its performance allowed us to optimize the model for the digit recognition task. The resulting classifier demonstrated strong accuracy in predicting handwritten digits, and the insights gained from visualizing feature importances after applying PCA informed our decision-making process and ultimately led to a more effective digit recognition model.