We'll start with looking at K-fold cross validation, a technique derived from its nature of plain cross validation, K-fold cv is used to evaluate a model's performance across generalized data on a more granular or targeted level . For instance Instead of relying on a single train-test split, K-fold cross-validation divides the dataset into *K* equally sized subsets, or "folds." [1] The model is then trained and tested *K* times, with a different fold used as the validation set in each iteration while the remaining *K*-1 folds are used for training. This ensures that every data point in the dataset is used for both training and validation exactly once. Overall providing a more reliable estimate of how well a model will perform on new, unseen data, helping to select the best model for a given situation. [2]

So that begs the question: why should we use K-Fold? The most obvious answer after just understanding K-fold cv is that it allows us to make efficient use of our training data. When working with small datasets, K-fold cross-validation is especially valuable because it ensures that every data point is used for both training and validation.[1] As this maximizes the data available for learning, which is crucial when our data is limited.The other less surface level aspect of K-fold is that it offers more accurate assessment by using multiple train-test splits. It reduces the chance that the model overfits a particular split, where the model performs well on the training set but struggles with new, unseen data. By averaging performance across different subsets of the data, the model's evaluation becomes less sensitive to variability in the data. [1] [2] With that being said when we use cross validation we shouldn't instinctively want to use K-fold cv. It certainly has a strong case to be used however, our data could result in some skewness when we partition the data. By and in any case of this happening a common replacement is by using stratified cross validation which keeps the distribution of each "fold" the same.

Lastly, what are the risks of skipping out on using such techniques such as K-fold cv?

Overfitting comes to mind when there is not proper validation, as a model on a single train-test split could be adequate, but any new data introduced could be horribly incorrect. Another risk is inability to distinguish data sensitivities, as a model's performance could be highly dependent on specific data points or subsets, which are not found in a single split. In addition the same can be said about only evaluating a model's performance on a single split may only capture a specific subset that in relation to the rest of the data might not coincide with high performance. [2]

One of machine learning's common case problems is the curse of dimensionality. [3] Here let's discuss how we can use and understand the pros and cons of each technique presented.PCA (Principal Component Analysis) and t-SNE (t-distributed Stochastic Neighbor Embedding) are both techniques used to reduce the number of dimensions in data, making it easier to visualize or analyze. However, they work differently and are used for different purposes.

They’re similar in the sense that both methods are unsupervised and have a common goal of reducing high-dimensional data into fewer dimensions. Furthermore both aim to keep as much important information as possible from the original dataset after reducing its dimensions. [2]

Where do their differences lie and how do they work? PCA is a linear method that simplifies the data by finding the main "directions" (called principal components) that capture the most variation in the data. [4] t-SNE on the other hand is a non-linear method that focuses on keeping data points that are close together in the original space still close in the reduced space. It’s especially good for visualizing clusters or groups in data. [4] When in the process of the reduction of space how does each focus on keeping structure in the data? PCA preserves the global structure of the data, meaning it looks at the overall shape of the data. Whilst t-SNE focuses on preserving the local structure, meaning it’s better at keeping relationships between nearby points, like clusters. [4] Furthermore PCA is deterministic, so it gives the same results every time you run it on the same data. t-SNE is non-deterministic, meaning it might give slightly different results each time you run it on the same data. Lastly, the difference in effectiveness and efficiency of PCA is faster and works well with larger datasets. t-SNE is slower and can be very time-consuming for large datasets.[4] Moreover PCA is quick and good at showing the overall data structure, but it struggles with non-linear relationships and is sensitive to outliers. t-SNE is great for visualizing clusters in complex data and handling non-linear relationships, but it’s slower, and the results can vary between runs. [4] It also requires some fine-tuning, like adjusting parameters.

Given these differences and efficiency constraints. When should we actually use either technique? In a general sense PCA is useful for simplifying data when you want to extract key features, filter noise, or do further analysis. It's often used in tasks like stock market predictions or genetic data analysis. Whereas t-SNE is mainly used for visualizing complex, high-dimensional data, such as in exploratory data analysis or fields like cancer research and music analysis, where seeing clusters of similar items is important.

A Gentle Introduction to k-fold Cross-Validation - MachineLearningMastery.com [1]
Cross Validation in Machine Learning - GeeksforGeeks [2]
Introduction to t-SNE: Nonlinear Dimensionality Reduction and Data Visualization | DataCamp [3]
Difference between PCA VS t-SNE - GeeksforGeeks [4]