Dimensionality Reduction and Manifold Learning

In machine learning problems, we often encounter datasets with a very large number of dimensions (features or columns). Dimensionality reduction techniques are used to reduce the number of dimensions or features within the data to a manageable or convenient number.

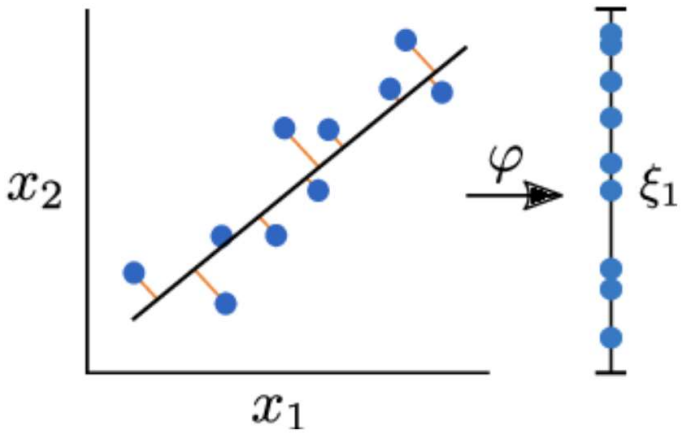
Applications of dimensionality reduction:

- Reducing size of data without loss of information
- Training machine learning models efficiently
- Visualizing high-dimensional data in 2/3 dimensions

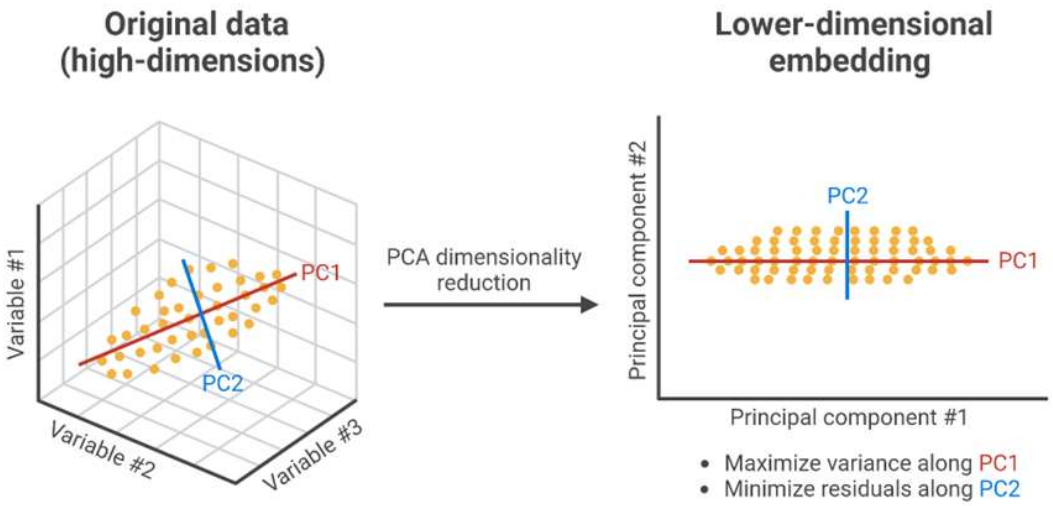
Principal Component Analysis (PCA)

Principal component is a dimensionality reduction technique that uses linear projections of data to reduce their dimensions, while attempting to maximize the variance of data in the projection. Watch this video to learn how PCA works: https://www.youtube.com/watch?v=FgakZw6K1QQ

Here's an example of PCA to reduce 2D data to 1D:



Here's an example of PCA to reduce 3D data to 2D:



Let's apply Principal Component Analysis to the Wine dataset.

In [27]:
```
from sklearn.decomposition import PCA
```

Reducing all the 13 components into 2.

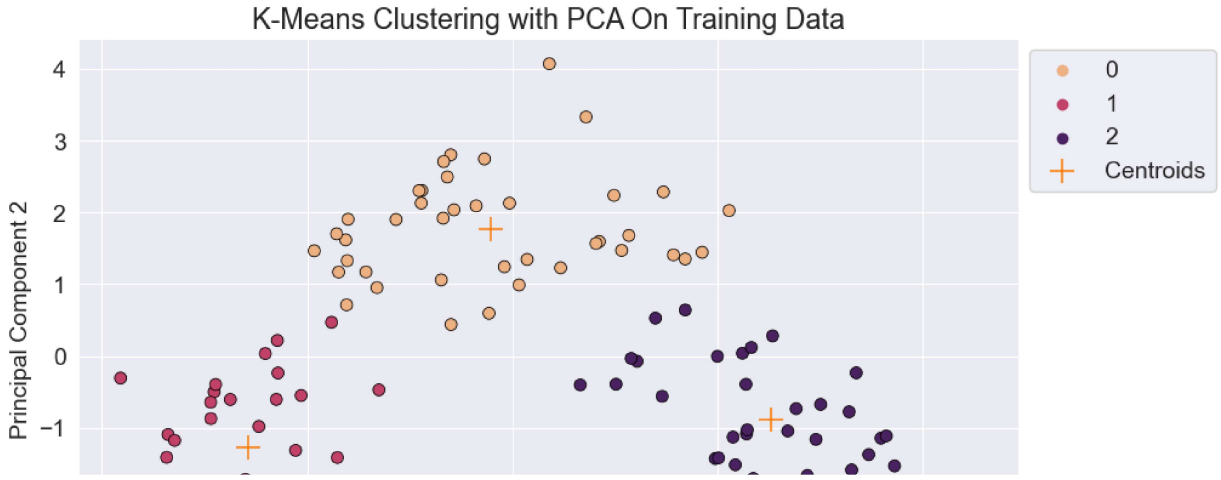
In [28]:
```
# Apply PCA to reduce dimensionality
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_val_pca = pca.fit_transform(X_val)
X_test_pca = pca.fit_transform(X_test)
```

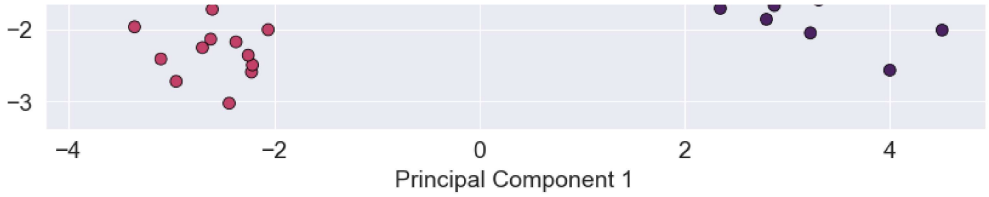
In [29]:
```
# Apply K-Means clustering on the PCA-transformed data
k_means_pca = KMeans(n_clusters=3, random_state=42,n_init=10)

labels_train_pca = k_means.fit_predict(X_train_pca)
train_pca_avg = silhouette_score(X_train_pca, labels_train_pca)

labels_val_pca = k_means.predict(X_val_pca)
val_pca_avg = silhouette_score(X_val_pca, labels_val_pca)

labels_test_pca = k_means.predict(X_test_pca)
test_pca_avg = silhouette_score(X_test_pca, labels_test_pca)
```

```
c:\Users\lenovo\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMea
ns is known to have a memory leak on Windows with MKL, when there are less chunks than availab
le threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [30]:
```
results.append({'Model Name' : 'PCA with Kmeans',
                'Training Set ': train_pca_avg,
                'Validation Set' : val_pca_avg,
                 'Testing Set': test_pca_avg})
```

In [31]:
```
# Visualize the clustering result
sns.scatterplot(x= X_train_pca[:, 0], y=X_train_pca[:, 1], data=X_train_pca, hue=labels_trai
sns.scatterplot(x= k_means.cluster_centers_[:, 0], y= k_means.cluster_centers_[:, 1], marker
plt.title('K-Means Clustering with PCA On Training Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```





As you can see, the PCA algorithm has done a very good job of separating different categories of wines using just 2 measures. The relative distance between the categories is also conveyed by the gaps between the clusters.

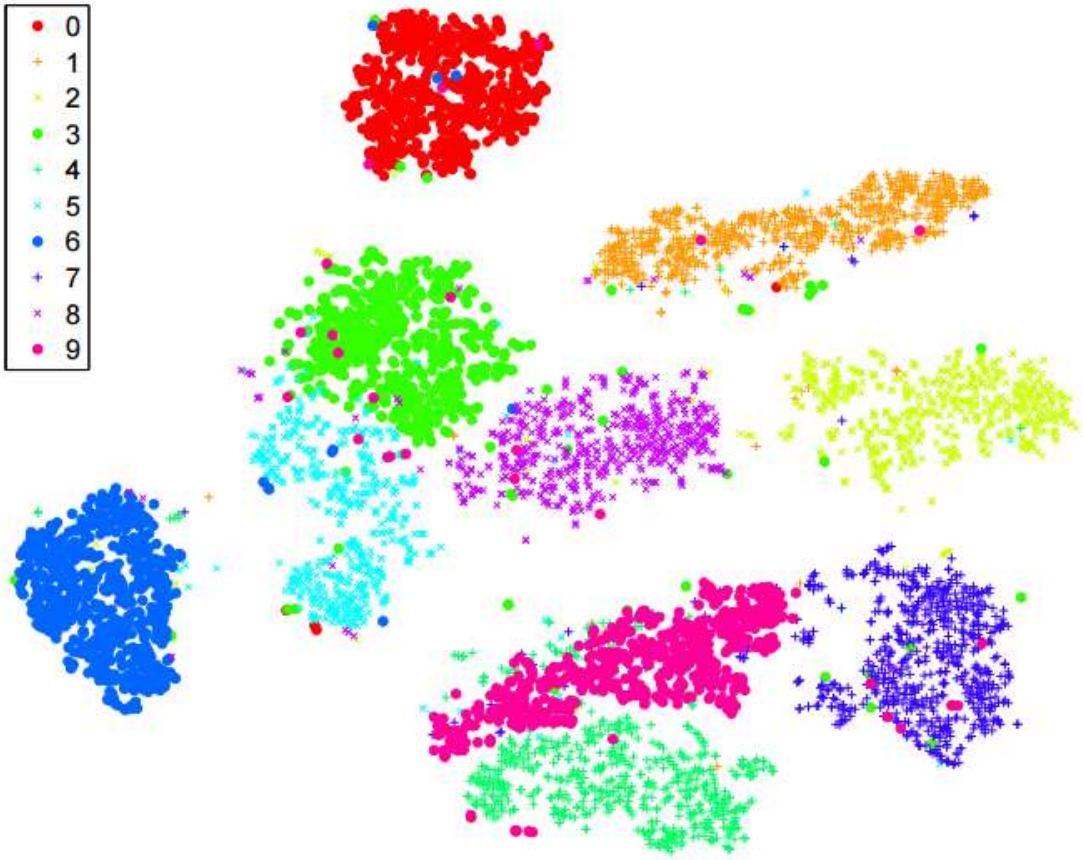
t-Distributed Stochastic Neighbor Embedding (tsne)

It's an unsupervised non-linear dimensionality reduction technique used for data exploration and visualizing high-dimensional data.

-- Unlike linear techniques (such as PCA), t-SNE allows us to separate data that cannot be separated by a straight line.

It provides intuition on how data is arranged in higher dimensions.

Often used to visualize complex datasets in two or three dimensions, helping us understand underlying patterns and relationships.



T-SNE is a Manifold Technique:

Manifold learning, also known as manifold techniques or methods, is a set of unsupervised learning

techniques used for dimensionality reduction and data visualization. The main idea behind manifold learning is to capture the underlying structure or geometry of high-dimensional data embedded in a lower-dimensional space.

How t-SNE Works:

Step 1: Pairwise Similarity Calculation

- Calculate pairwise similarity between all data points in the high-dimensional space using a `Gaussian kernel`.

The Gaussian kernel is effective in capturing non-linear relationships in data, and it measures the similarity between two data points based on their Euclidean distance.

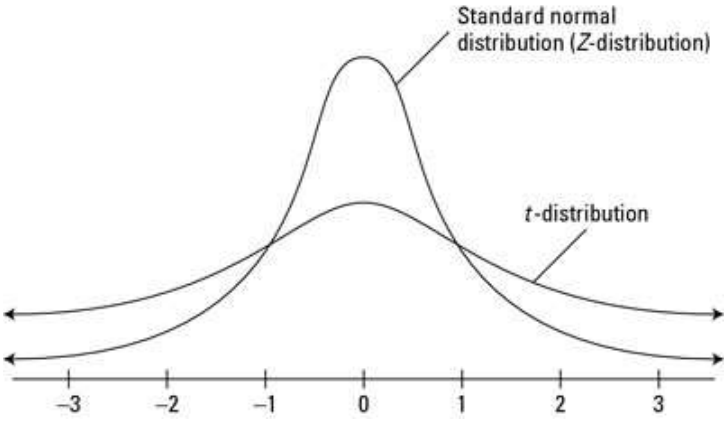
- Points far apart have a lower probability of being picked than close points i.e. Similarity is calculated using `Conditional Probability`.

Conditional probability is fundamental in Bayesian statistics, machine learning, and probabilistic modeling. It allows us to update our beliefs about an event based on new information.

Step 2: Mapping to Lower Dimensional Space

- Map higher-dimensional data points onto a lower-dimensional space while preserving pairwise similarities, using `Student's t distribution`.

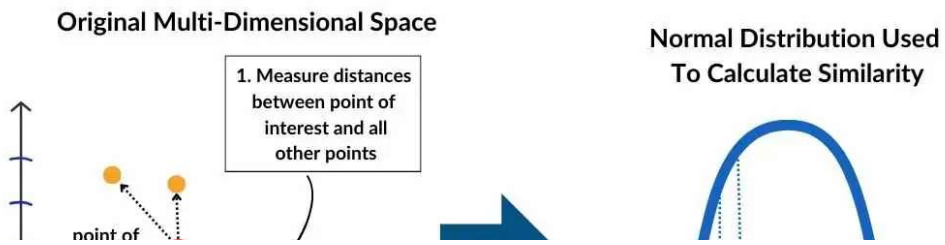
Student's t distribution is commonly used in hypothesis testing and confidence interval estimation when the sample size is small or when the population standard deviation is unknown.

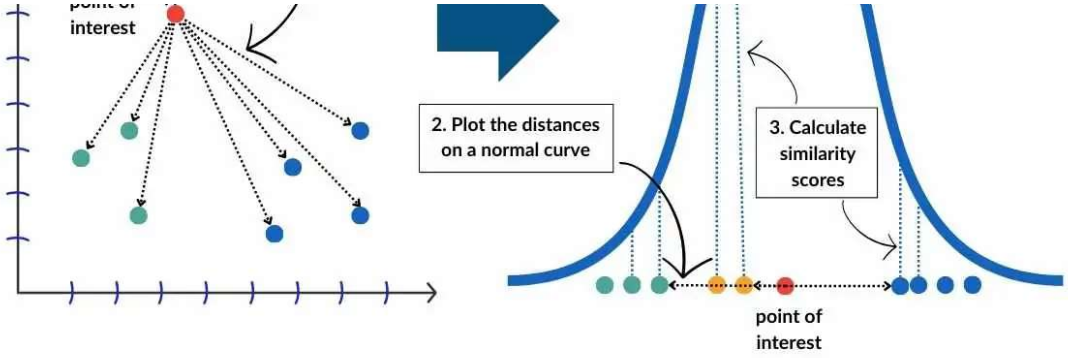


- Focuses on preserving small pairwise distances.

Step 3: Optimization

- Optimize similarity measures in both higher and lower dimensions.
- Balances preserving local structures and minimizing global distortion.





Space and Time Complexity

-- The space complexity of t-SNE is generally considered to be `O(N^2)`, where N is the number of data points.

-- The time complexity of t-SNE is approximately `O(N^2 logN)` to `O(N^3)`, where N is the number of data points.

Parameters:

1. **Perplexity**: A crucial parameter that balances preserving local and global structures. It roughly represents/ controls the effective number of neighbors that each point considers during the dimensionality reduction process.
2. **Learning Rate**: Controls the step size during optimization.

Use Cases:

Visualization: Primary use is for visualizing high-dimensional data in 2D or 3D.

Clustering Analysis: t-SNE is often used to identify natural clusters in the data.

Limitations:

- *Sensitivity to Perplexity*: Results can vary significantly with different perplexity values.
- *Computational Complexity:* t-SNE is computationally expensive, making it less suitable for large datasets.
- *Loss of Global Structure:* It may not preserve global structures well.

Implementation in Python:

t-SNE is implemented in Python libraries such as scikit-learn `(sklearn.manifold.TSNE)`.

Barnes-Hut Approximation:

- t-SNE can be computationally expensive for large datasets.
- Barnes-Hut approximation speeds up t-SNE by approximating pairwise similarities using a tree-based approach.
- Allows t-SNE to be applied on large real-world datasets

Allows t-SNE to be applied on large real-world datasets.

In [32]:
```
from sklearn.manifold import TSNE
```

In [33]:
```
# Apply PCA to reduce dimensionality
tsne = TSNE(n_components=2)
X_train_tsne = tsne.fit_transform(X_train)
X_val_tsne = tsne.fit_transform(X_val)
X_test_tsne = tsne.fit_transform(X_test)
```

After fitting and transforming data, we will display Kullback-Leibler (KL) divergence between the high-dimensional probability distribution and the low-dimensional probability distribution.

In [39]:
```
tsne.kl_divergence_
```

Out[39]:
```
0.018220379948616028
```

Low KL divergence is a sign of better results.

In [40]:
```
import plotly.express as px
```

We can also plot `kl_divergence_` to find optimal perplexity value.

In [45]:
```
perplexity = np.arange(5, 105, 5)
divergence = []

for i in perplexity:
    _model = TSNE(n_components=2, init="pca", perplexity=i)
    reduced = _model.fit_transform(X_train)
    divergence.append(_model.kl_divergence_)
fig = px.line(x=perplexity, y=divergence, markers=True)
fig.update_layout(xaxis_title="Perplexity Values", yaxis_title="Divergence")
fig.update_traces(line_color="red", line_width=1)
fig.show()
```

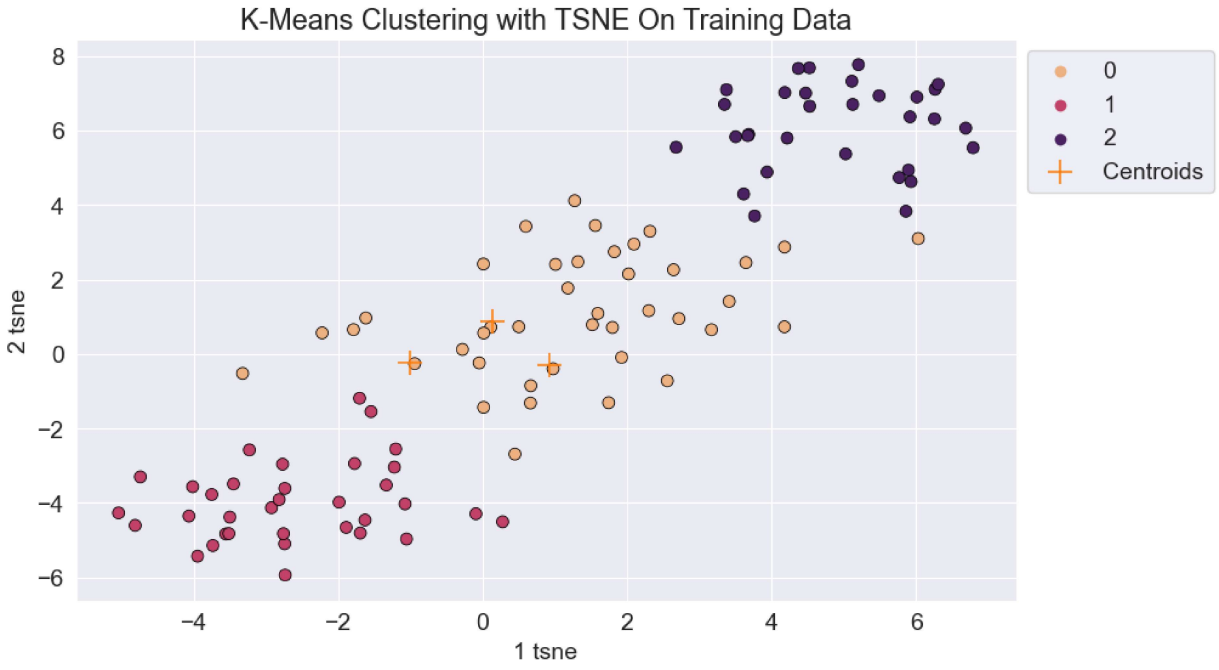
It is very difficult to find Optimal Perplexity Value for hypertuning. So, let's take the default perplexity value for applying K-Means clustering on the tSNE-transformed data.

In [35]:
```
# Apply K-Means clustering on the tSNE-transformed data
k_means_tsne = KMeans(n_clusters=3, random_state=42,n_init=10)

labels_train_tsne = k_means.fit_predict(X_train)
train_tsne_avg = silhouette_score(X_train_tsne, labels_train_tsne)

labels_val_tsne = k_means.predict(X_val)
val_tsne_avg = silhouette_score(X_val_tsne, labels_val_tsne)

labels_test_tsne = k_means.predict(X_test)
test_tsne_avg = silhouette_score(X_test_tsne, labels_test_tsne)
```

```
c:\Users\lenovo\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [46]:
```
# Visualize the clustering result
```

```
sns.scatterplot(x= X_train_tsne[:, 0], y=X_train_tsne[:, 1], data=X_train_tsne, hue=labels_t
sns.scatterplot(x= k_means.cluster_centers_[:, 0], y= k_means.cluster_centers_[:, 1], marker
plt.title('K-Means Clustering with TSNE On Training Data')
plt.xlabel('1 tsne')
plt.ylabel('2 tsne')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```



As you can see, the TSNE algorithm has done a great job of separating different categories of wines using just 2 tsne. The relative distance between the categories is also conveyed by the gaps between the clusters.

In [37]:
```
results.append({'Model Name' : 'TSNE with Kmeans',
                'Training Set ': train_tsne_avg,
                'Validation Set' : val_tsne_avg,
                 'Testing Set': test_tsne_avg})
```

In [38]:
```
results_df = pd.DataFrame(results)
results_df
```

Out[38]:

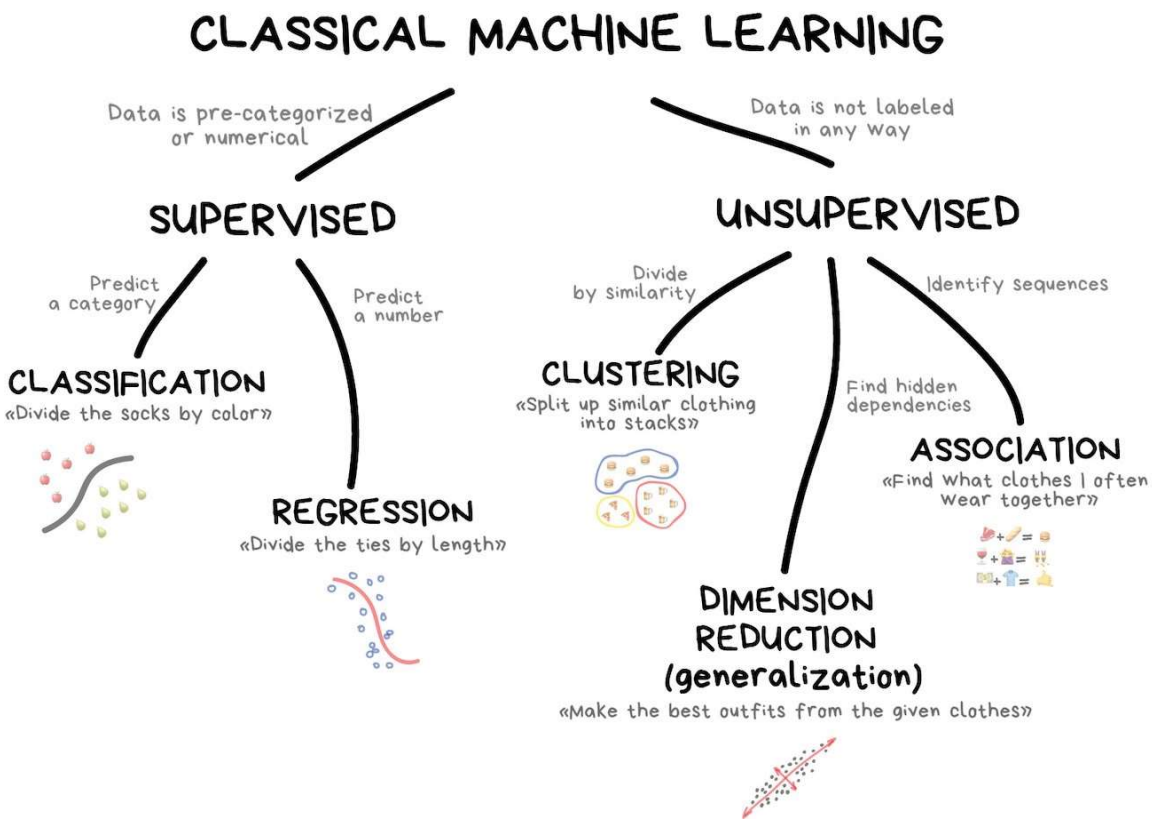
	Model Name	Training Set	Validation Set	Testing Set
0	Kmeans	0.280862	0.268975	0.291732
1	Agglomerative Clustering	0.280365	0.268975	0.276613
2	PCA with Kmeans	0.560373	0.457323	0.201893
3	TSNE with Kmeans	0.547412	0.569360	0.480826

The following conclusions have been drawn:

-- Among all the models, TSNE with K-means exhibits favorable results on a small dataset in classifying wine within the training dataset

-- And its performance on the test dataset is also as better than other models due to the limited size of the data.

Summary and References



The following topics were covered:

- Overview of unsupervised learning algorithms in Scikit-learn
- Clustering algorithms: K Means, Hierarchical clustering etc.
- Dimensionality reduction (PCA)
- Manifold Technique (t-SNE)