Statistical_ML_Hwk_3

May 31, 2023

1 Machine Learning - 3rd Assignment

1.1 Student: Sheedeh Sharif Bakhtiar

1.1.1 Student ID: 400422108

Code + data can also be found here on my GitHub

2 Exercise 1:

2.1 *What is the curse of dimensionality and how does it affect clustering?*

According to Wikipedia the "curse of dimensionality" refers to the various "hurdles" we may face when analyzing data in high-dimensional spaces that do not occur in low-dimensional spaces, such as three-dimensional physics space of everyday experience. The curse of dimensionality can result in an exponential increase in computational efforts requried for processing and analizying data.

This specific "curse" might cause many problems in fields such as machine learning, as the more dimensions there are, the "further" apart various data points might be from one another, given the extra dimensions to take into account. As the dimensionality increases, the number of data points required for good performance increases exponentially.

> It has been shown in this 1968 study by Hughes study from 1968 that increasing the number of dimensions also improves a classifier's performance *up until a certain point* — after which the classifier's performance deteriorates, so increasing the number of dimensions isn't always a *bad* idea. Therefore the curse of dimensionality is also sometimes referred to as the Hughes phenomenon.

Unfortunately, the curse of dimensionality has a direct effect on distance functions. As distance functions are *vital* in clustering algorithms, as clusters tend to be defined and adjusted using these distance metrics (i.e. Manhattan distance), the curse of dimensionality and the number of dimensions plays a critical role in clustering algorithms.

For example, for a given point A, let us assume $dist_{min}(A)$ is the distance between A and it's nearest neighbor, while $dist_{max}(A)$ is the distance between A and it's farthest neighbor.

In low-dimensional space, i.e. one-dimensional, two-dimensional or even three-dimensional space, we have:

$$\frac{dist_{max(A)} - dist_{min(A)}}{dist_{min(A)}} > 0$$

But as the dimensions increase, that is as $dim \to \infty$, we have:

$$lim_{dim \to \infty}(\frac{dist_{max(A)} - dist_{min(A)}}{dist_{min(A)}} > 0) \to \infty$$

In other words, for a d-dimensional space (with $d \to \infty$), given n random points, the $dist_{min(A)} \approx dist_{max(A)}$, meaning that any given pair of points are equidistant to one another.

Some solutions to this issue include using different distance metrics such as cosine similarity to replace Euclidean distance, or use dimensionality-reducing techniques such as PCA to help alleviate the problem.

Sources: - Wikipedia: Curse of Dimensionality - Towards Data Science: Curse of Dimensionality — A "Curse" to Machine Learning

Exercise 2: ## *In what cases would you use regular PCA, incremental PCA, randomized PCA, or random projection?*

Principal component analysis or PCA, is a statistical technique for reducing the number of dimensions in a dataset, thus increasing the interpretability of data while preserving the maximum amount of information and enabling the visualization of multidimensional data.

PCA works by extracting the important information from the data and to express this information as a set of summary indices called principal components. In summary, PCA finds hyperplanes in the k dimensional space that approximates the data as well as possible in the least squares sense. A hyperplane that is the least squares approximation of a set of data points make sthe variance of the coordinates on the line or plane as large as possible.

PCA is guarenteed to produce uncorrelated features, due to *how* the new features are constructed (which is beyond the scope of this exercise) and has no hyperparameters while being relatively fast, making it easy to use.

However, PCA comes with a number of drawbacks: - PCA should mainly be used for scenarios where variables are strongly correlated. If the relationship between variables is weak, PCA does not perform very well to reduce the number of dimensions in our dataset. - On a similar note, PCA assumes the relationships between features is linear. PCA may not perform very well when the relationship between datapoints is not linear. - PCA also requires normalization, as PCA is sensitive to scale. - PCA is also sensitive to outliers, and cannot handle missing values. - PCA is only suitable for continuous data. If our dataset consists of a mix of continuous and discrete features, perhaps another dimensionality reduction technique would be a better fit.

While PCA is relatively fast and easy to use, sometimes our dataset is too large to fit into memory. **Incremental PCA** can be used in such scenarios. Incremental PCA splits the dataset into mini-batches that can fit into memory. Then, each batch is fed into our incremental PCA algorithm one at a time.

Incremental PCA updates the principal components based on the new data points, without the need to recompute PCA from scratch each time, making it incredibly useful for very large datasets and at times, a more efficient and scalable approach to PCA, but it mgiht not be as accurate as computing the full PCA due to the inherent approximation of the incremental algorithm.

While vanilla PCA uses low-rank matrix approximation to estimate the principal components used in dimensionality reduction, for large datasets this method could be costly and difficult to scale. Much like incremental PCA, **Randomized PCA** was designed to help solve this specific problem.

Randomized PCA is a stochastic algorithm that quickly finds an approximation for the first n principal components, making it dramatically faster than the previously discussed versions of PCA, as the first n principal components can be approximated by computing the eigenvectors of the covariance matrix of the projected data. This makes randomized PCA significantly faster and more scalable than vanilla PCA.

Similar to PCA, random projection is a statistical technique for reducing the dimensionality of a dataset. This method is known for it's power, simplicity and low error rates when compared to other methods. Random projection is significantly faster and less complex than PCA, and is robust to outliers, as opposed to PCA that was very sensitive to outliers. However, this comes at the cost of a small loss in accuracy, as PCA maintains the best possible projection.

In summary: - Vanilla PCA should be used when we have a number of correlated, continuous features with no missing values as regular PCA is fast and easy to use. - Incremental PCA should be used when we have a dataset that's too large to fit into memory, however, this comes at the cost of slightly reduced accuracy. - Randomized PCA should be used when we have a dataset that's too large for vanilla PCA to handle. - Random projection is significantly faster than PCA methods, and is robust to outliers, so should our data contain many outliers or speed is an important factor (at the cost of a little bit of accuracy), then random projection may be the best choice for the task at hand.

> **Additional Notes**: According to the empirical results shown in this Kaggle notebook, incremental PCA is significantly slower than regular PCA and randomized PCA, and randomized PCA works better than regular PCA in almost all cases. However, the number of datapoints negatively impact the randomized PCA algorithm, and in cases where it's feasible, regular PCA is more feasible.

Sources: - Wikipedia: Principal Component Analysis - Sartorius: What Is Principal Component Analysis (PCA) and How It Is Used? - OriginLab: 17.7.1 Principal Component Analysis - Crunching the Data: When to use principal component analysis - AI Aspirant: Types of PCA - OpenGenus: Sparse and Incremental PCA - Wikipedia: Random Projection - ResearchGate: Does Random Projection have advantage(s) over PCA?

3 Exercise 3:

3.1 *Does it make sense to chain two different dimensionality reduction algorithms?*

It does make sense to *sometimes* chain two different dimentionality reduction algorithms, as it very much depends on the data we are working with.

For instance, if our dataset consists of a combination of continuous and discrete variables, we could apply PCA onto the continuous variables, and another algorithm such as MCA onto the discrete variables, thus *making up for the weaknesses of the other algorithm.* For instance, should our data exhibit non-linear relationships, we might apply one algorithm designed to capture said non-linear relationships, and another to further reduce the dimensionality.

Nonetheless, we should be mindful of each algorithm's strengths and weaknesses, as combining the wrong two algorithms might introduce unwanted complexity into our data.

In general, we can also chain dimensionality reduction algorithms when we have a high-dimensional dataset, and we can use one algorithm to reduce the dimensionality and then another to further reduce the dimensionality.

However, it should be noted that it's usually not beneficial to chain the same dimensionality reduction algorithm twice, i.e. applying PCA twice, as there is a measure of information loss that comes with these dimensionality reduction techniques, and this information could be unbalanced across subsets of the original features. By applying PCA on an already-PCA-reduced dataset, we'll be ignoring the fact that some principal components matter more than other principal components.

Sources - Brainly: - ChatGPT (2 Prompts: `"As a data scientist, does it make sense to chain two different dimensionality reduction algorithms together?"` and `"As a data scientist, when is it beneficial to chain two different dimensionality reduction algorithms together?"`) - Cross Validated: Can PCA be applied twice or more?

4 Exercise 4:

4.1 *What are the main assumptions and limitations of PCA?*

As explained extensively in Exercise 2, the main assumptions and limitations of PCA have been already listed. However, to reiterate:

- PCA should mainly be used for scenarios where variables are strongly correlated. If the relationship between variables is weak, PCA does not perform very well to reduce the number of dimensions in our dataset.
- On a similar note, PCA assumes the relationships between features is linear. PCA may not perform very well when the relationship between datapoints is not linear.
- PCA also requires normalization, as PCA is sensitive to scale.
- PCA is also sensitive to outliers, and cannot handle missing values.
- PCA is only suitable for continuous data. If our dataset consists of a mix of continuous and discrete features, perhaps another dimensionality reduction technique would be a better fit.

Sources are the same as Exercise 2

5 Exercise 5:

5.1 *How can clustering be used to improve the accuracy of the linear regression model?*

Linear regression is an incredibly simple method that is computationally simple and highly interpertable, but it isn't without it's drawbacks. For one, it works best when data is linearly separable, and it's unable to handle complex regions. One method for incorporating complexity in our linear regression model is to "do some of the work beforehand" and introduce a clustering algorithm *before* we fit our model to our data.

Clustering methods such as k-means find implicit sub-groups in the data that we might not be able to detect at first glance, as clustering algorithms group data points with similar characteristics to one another.

We can "stack" clustering and linear regression on top of each other to create a more specific linear regression model where the variations and subgroups in the data have been identified.

Additionally, we can use clustering methods to find outliers and anomalies that might've negatively impacted our model's performance.

In summary, clustering methods can add additional complexity to our linear regression model and capture implicit similarities between different data points for higher accuracy.

Sources: - Cross Validated: Cluster analysis followed by regression - Paperspace: Building sharp regression models with K-Means Clustering + SVR - ChatGPT (1 Prompt: `"As a data scientist, how can clustering be used to improve the accuracy of the linear regression model?"`)

6 Exercise 6:

6.1 *How is entropy used as a clustering validation measure?*

In general, the goal of clustering is to group a set of objects together such that objects in the same group are similar to one another. However, how good our clusters are is an important factor to take into account, as we'd like our clusters to be *useful* in our task at hand (such as describing our data).

There are multiple metrics we can employ to measure the "goodness" of a cluster (otherwise known as *cluster validation*) and one of them is entropy. Entropy in a clustering context measure the probability of a record in cluster i being classified as class i and has the following formula:

$$-\sum_{i=1}^{K} p_i log p_i$$

Entropy can be used as a method for measuring impurity. The entropy for a cluster is 0 when the probability of all data points belonging to a specific cluster is 1 — meaning that our cluster is *well-separated*. Therefore, the lower our average entropy is across our dataset, the better.

Therefore, we should strive to tune our clustering algorithm (e.g. for k-means, this might mean setting the right value for k) that we minimize our entropy as much as possible.

Sources: - Towards Data Science: Data Clustering Tutorial for Advanced - Visual Studio Magazine: Data Clustering Using Entropy Minimization

7 *Exercise 7:

7.1 *What is label propagation? Why would you implement it, and how?*

The label propagation algorithm (LPA) is a semi-supervised machine learning algorithm that assigns labels to previously unlabeled data points. We initially have a subset of datapoints that *have* labels or classifications, and these labels are then propagated to the unlabeled points using this algorithm.

Another way of looking at this algorithm is as an algorithm for finding communities in a graph. LPA detects these communities using the network structure as it's only guide, and propagates labels throughout the network.

Intuitively, it's *similar* to training a machine learning algorithm *that has learned a pattern in our data* onto training data and then predicting labels for test data, and assigning the predicted labels on the test data as "truth" and appending them to the training data. *Or*, it can be seen as a single label quickly becoming dominant in a densely connected group of nodes, but will have trouble crossing a sparsely connected region, with labels getting trapped inside a densely connected group of nodes.

According to Wikipedia, this process consists of 5 steps, which are as follows:

1. Initialize all the labels in the netowrk. For a given node x, $C_x(0) = x$
2. Set our time to it's first step. I.e. set $t = 1$
3. Arrange all the node in the network in a random order and set it to X.
4. For each $x \in X$ chosen in that specific order, let the label be the most frequent label occuring among the neighbors, i.e. $C_x(t) = f(C_{x_{i1}}(t), \dots, C_{x_{im}}(t), C_{x_{i(m+1)}}(t-1), \dots, C_{x_{ik}}(t-1)$. Select a label at random if there are multiple highest frequency labels.
5. If every node has a label that the maximum number of their neighbours have, then stop the algorithm. Else, set $t = t + 1$ and go to (3)

How the labels are initially set is where this algorithm becomes semi-supervised as opposed to completely unsupervised. If we set the initial labels of the data points that have labels to their labels and propagate these labels to the data points that do not have labels, then we'd have labeled the unlabeled data points successfully.

How we pick the nearest neighbors could be done through distance metrics, such as Euclidean distance. Implementing this algorithm is as simple as iterating through each unlabeled data point, finding it's nearest neighbors, and assigning the most frequent label to this data point, in a method that's incredibly similar to the more-familiar KNN (k-nearest neighbours) algorithm.

Sources: - Wikipedia: Label propogation algorithm - neo4j: Label Propagation - Alibaba Cloud: Label propagation

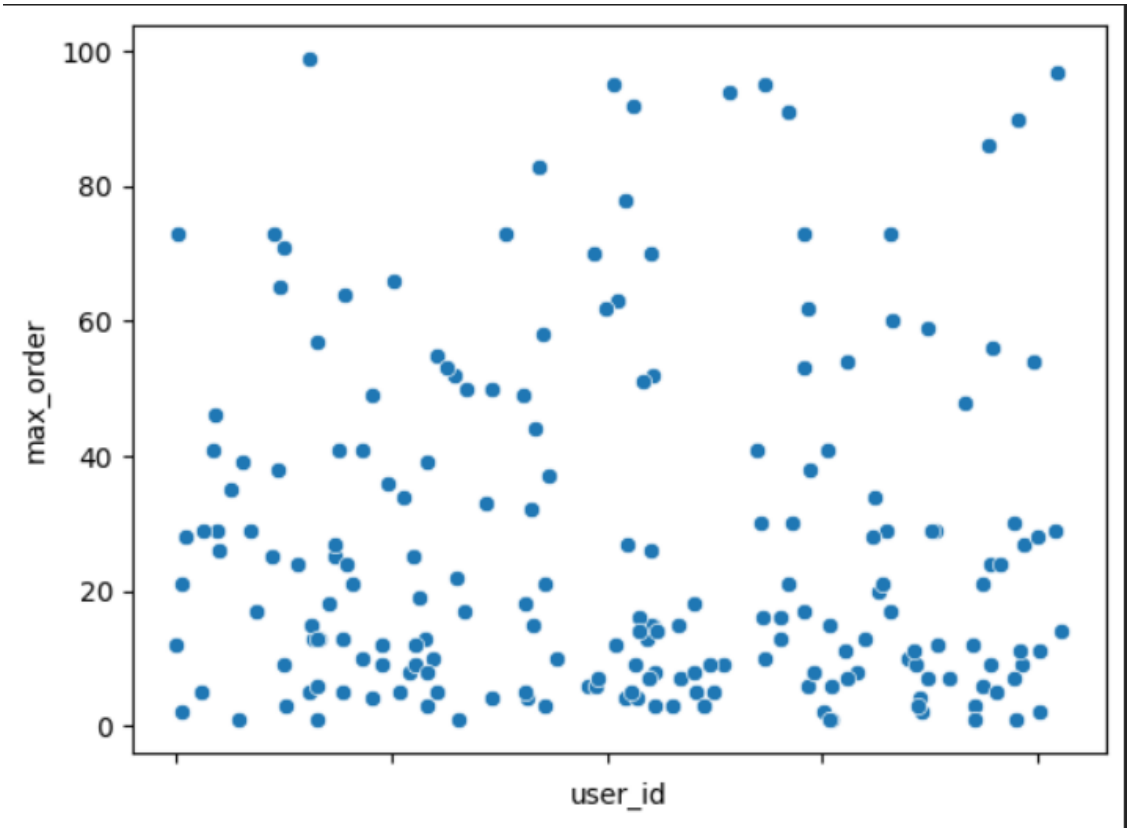
8 Exercise 8:

8.1 *You are going to work on the Supermarket dataset for predictive marketing . Your task is to use clustering algorithms to segment the customers into distinct groups based on their shopping behavior and demographics.*

8.1.1 *Explore and preprocess the dataset. This may involve handling categorical variables and normalizing or scaling numerical features and feature engineering.*

EDA

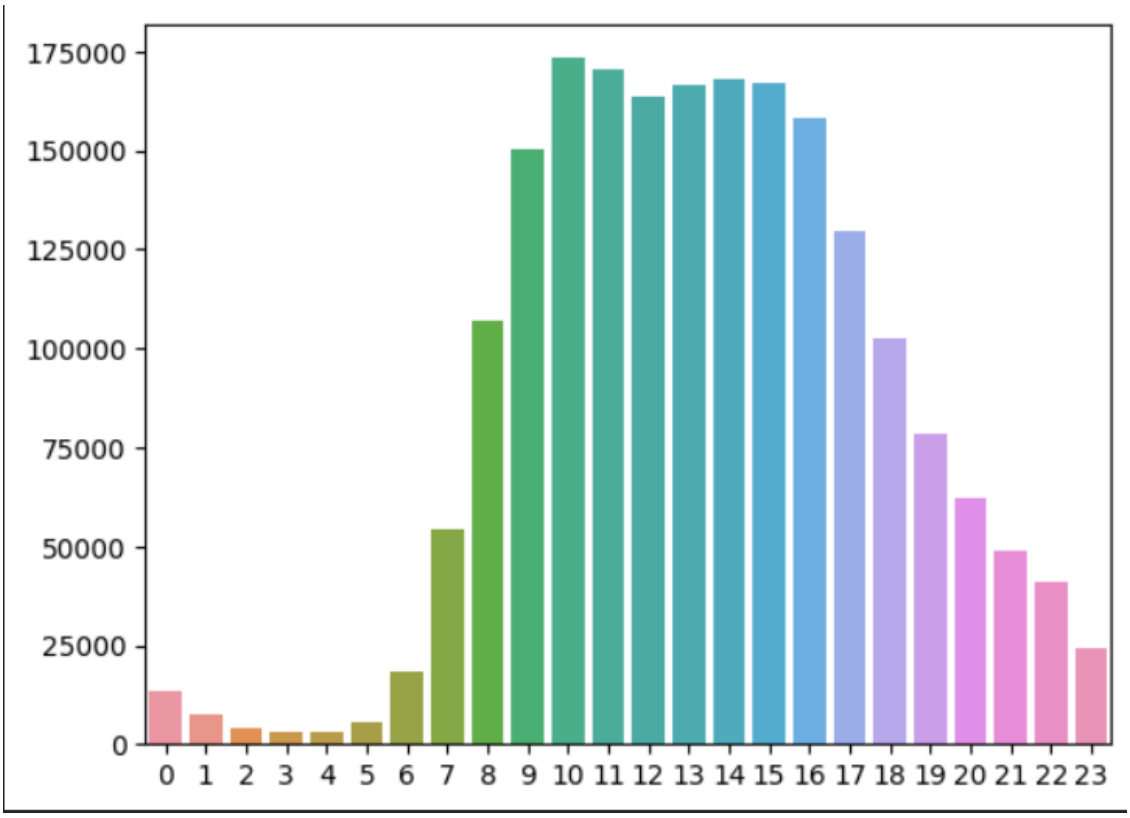
Average number of orders per customer If we plot 200 customer's max order number, we'll get something like this:



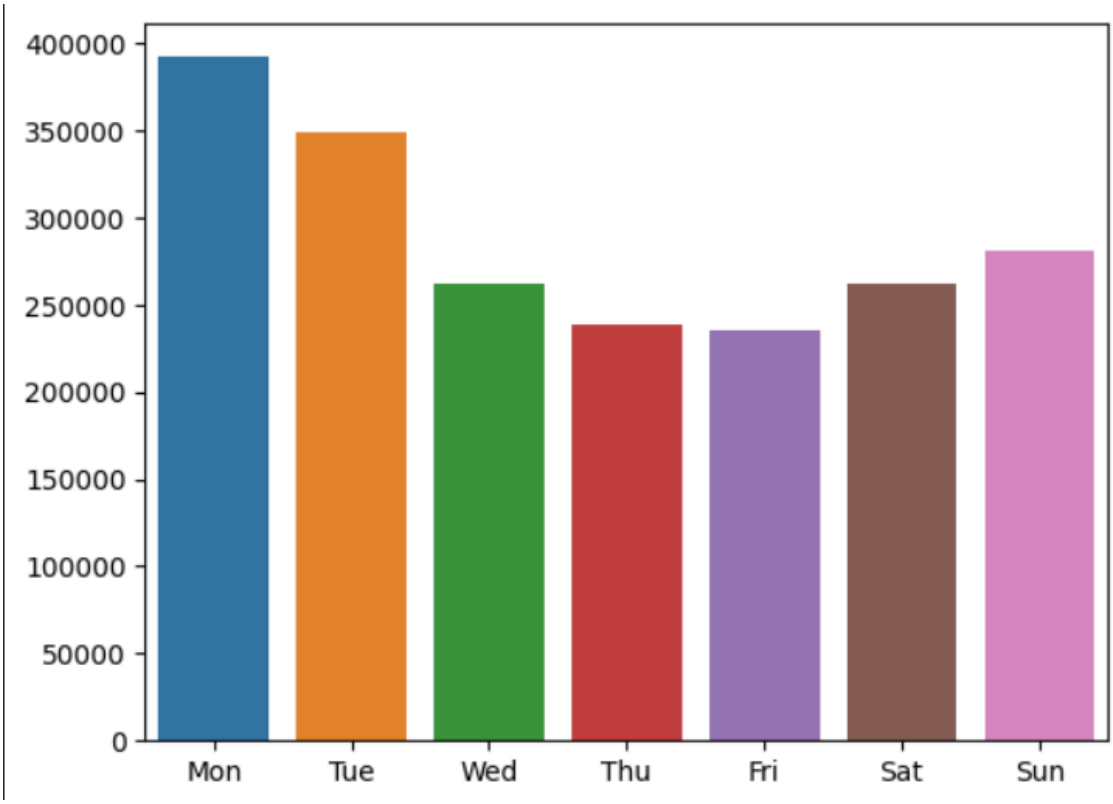
This graph tells us that most customers order less than 20 times, and the data isn't *significantly* varied, but there aren't any clear patterns either.

Time dependency Next, we'll check the order frequency by time of day and day of week to see if there are any clear correlations there.

First we'll check the hour of the day:

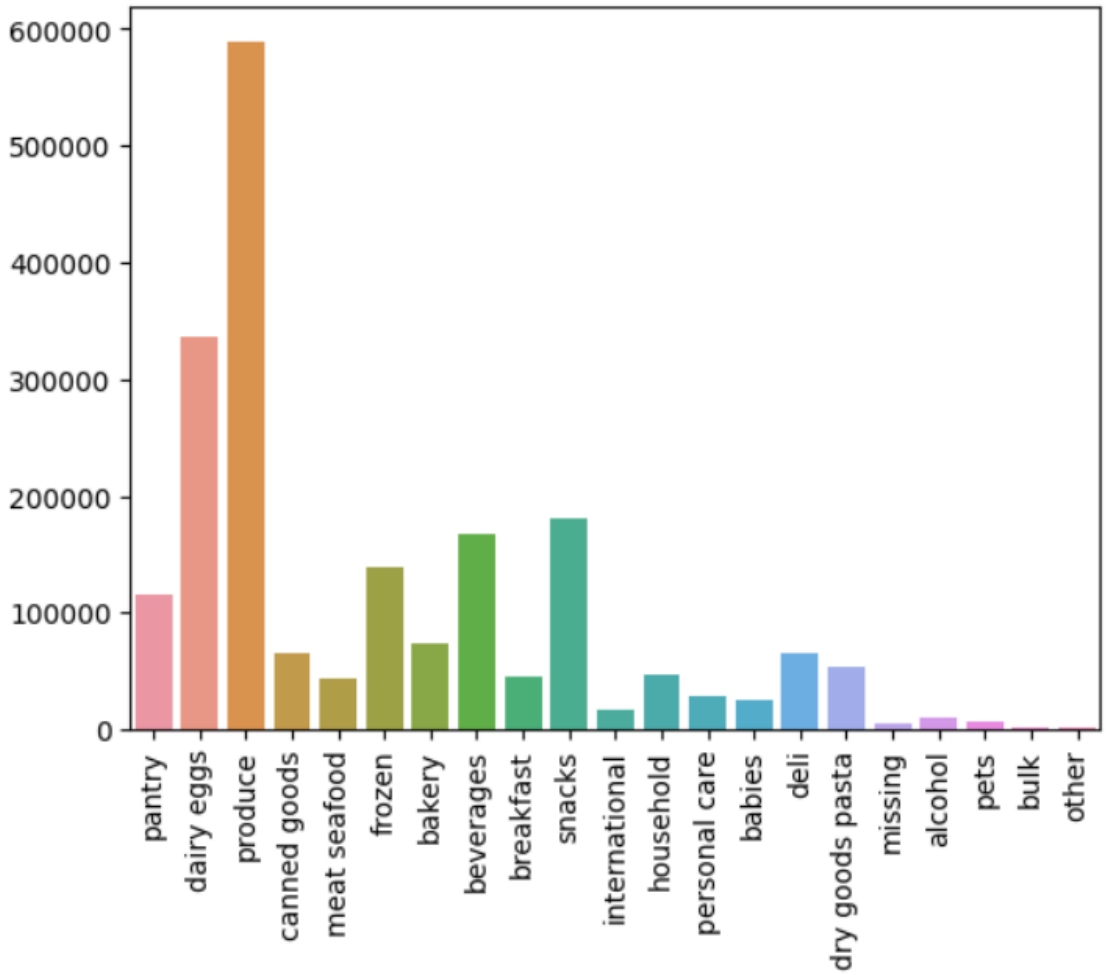


As you can see, most orders are in the middle of in the middle of the day, between 9 AM and 5 PM, with very few orders in the middle of the night, such as 3 or 4 AM. Next, we'll check the day of the week;



Again, similar to the other plot it's quite clear that there are some days where there tends to be more orders than others. For instance, in the graph above it's clear that most orders are on the first day, i.e. Monday.

Most Popular Department Similarly, we can check and see which department is the most popular;



And again, it's quite clear which department is the most popular.

Data Preprocessing Summary of data preprocessing steps made on original data (full code in other jupyter notebook (`Assignment_3_code.ipynb`)): - The column `days_since_prior_order` was the only column containing NaN values. Out of 2019501 rows, 124342 were `NaN`, meaning that the user had not previously ordered. For simplicity's sake, these values were filled with a value of -1. - Columns such as `order_id` and `user_id` that are mostly used for "identification" purposes are dropped, as it is thought that they play little role in our final machine learning model's performance. - Similarly, the `product_name` column was dropped, as the `product_id` column already exists. - In our dataset, we have two different categorical columns; `department` and `product_name`. Given that the latter column has been dropped, `department` will be converted into categorical codes.

The final dataframe used in the following sections looks as follows:

[17]:
```
import pandas as pd

df = pd.read_csv('./data/preprocessed_data.csv')
df.head()
```

[17]:

	order_number	order_dow	order_hour_of_day	days_since_prior_order	\
0	1	2	18	-1.0	
1	1	2	18	-1.0	
2	1	2	18	-1.0	
3	1	2	18	-1.0	
4	1	2	18	-1.0	

	product_id	add_to_cart_order	reordered	department_id	department
0	17	1	0	13	16
1	91	2	0	16	7
2	36	3	0	16	7
3	83	4	0	4	19
4	83	5	0	4	19

8.1.2 *Use K-means clustering to identify the optimal number of clusters. Experiment with different values of K and use metrics such as the elbow method and silhouette score to evaluate the performance of the clustering.*

For implementing the algorithm, Scikit-Learn's implementation of K-means clustering was used.

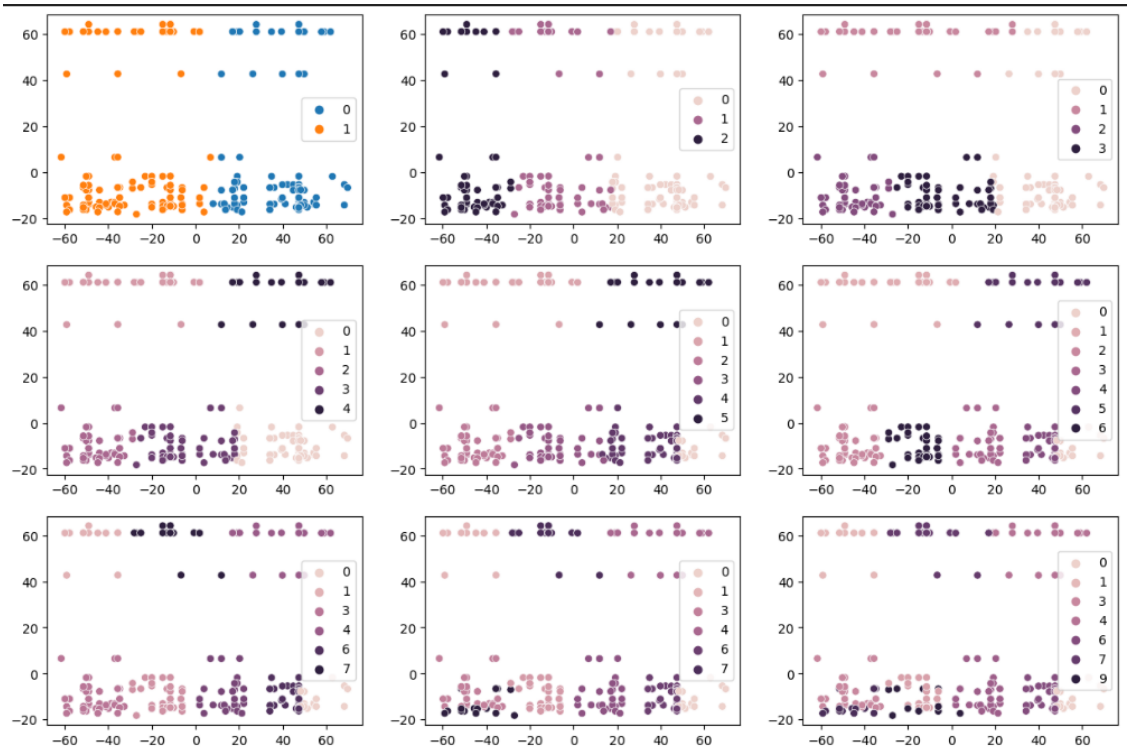
Results using the elbow method (plotting WCSS against the value of k) shows that `k = 8` would be the most optimal value. Using the elbow method again (plotting silhouette score against the value of k) shows that `k=` would be the most optimal value.

8.1.3 *Visualize the clusters and analyze their characteristics. This may involve plotting the clusters in 2D or 3D using PCA or t-SNE.*

For this, sklearn's implementation of PCA was used.

For each number of k, we can reduce the number of features down to 2 for easy visualization using PCA, and then iterating over different values of k (from 2 to 10) to find the best (i.e. most well separated clusters).

The results of which are shown below:



Where the number of colors in the legend corresponds to the number of clusters used. We're looking for the clusters that are best separated from one another, and it appears that when `k=5`, the clusters are best separated.

8.1.4 *Experiment with other clustering algorithms such as DBSCAN or hierarchical clustering, and compare their performance with K-means.*

The results are as follows:

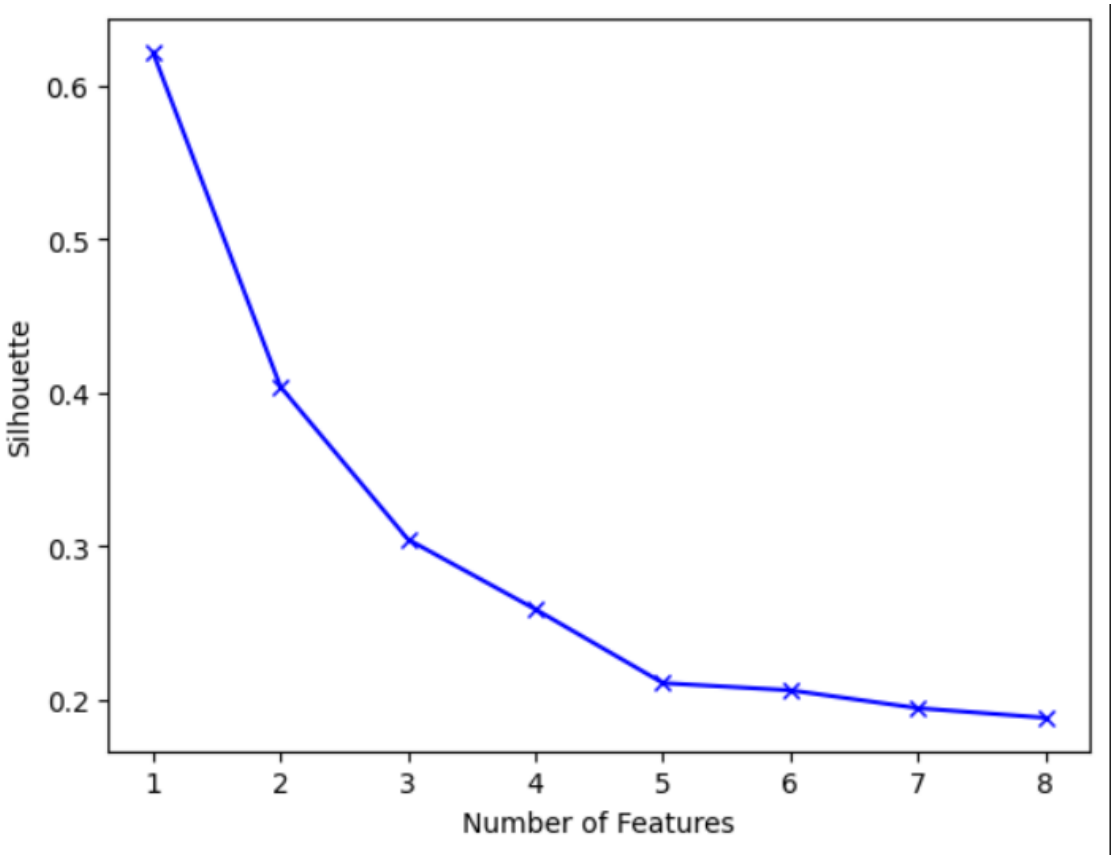
Model Name	Score
H Clustering	0.323507
KMeans	0.240621
DBScan	-0.25176

With hierarchical clustering having the best absolute score. However, with more fine tuning one might obtain better results.

8.1.5 **Try to reduce data dimensionality using PCA before training your model, use different numbers of components and report their effects.*

The preprocessed dataset used in training previous k-means models consisted of 9 features. Therefore,

The methodology I used for this was iterating over all possible number of features, calculating the principal components and transforming the data, and then finding the best number of clusters between 2 and 15, and taking the lowest silhouette score into account. The results are shown in the graph below;



The best results was when we had reduced our dataset down to 8 features with 14 clusters, however, due to computational complexity, opting for 5 features with 12 clusters might be the better choice.