Predicting Startup Success

Milestone 3: Model Implementation

Website:

https://wihi1131.github.io/Data-Mining-Project/Data%20Exploration

GitHub:

https://github.com/WiHi1131/Data-Mining-Project

Michael Van Vuuren, Melina Kopischkie, William Hinkley

Overview

This report is organized into six sections. The authors of each section are:

Will	Section 1 Section 2
Michael	Section 3 Section 4 Section 5
Melina	Section 6

The sections have different goals, preprocessing steps, and models. Some sections contain multiple models. In total, we created nine models. Each section also discusses our reasoning for the section and finishes off with a conclusion.

Overall, the objective of this milestone was to train machine learning models to recognize patterns in our datasets and then validate these models with test data. Using these models, we can make predictions on inference sets containing samples that our models have not seen before. This is powerful because it enables us to predict how successful a company is likely to be based on preliminary data, the results of which can then be used to modify their strategy if needed. All model files are in the `Model Building` folder in the GitHub repository.

Table of Contents

Section 1

File: Will-K-Means.ipynb

Goal: Generate clusters using K-Means for data in `primary.csv`.

Reason: After our Data Exploration phase and visualizing correlations (or the lack thereof) with different numerical variables revealed that it might be difficult to find any strong explanatory variables from our scraped dataset. For this reason, our first step was to generate clusters of data from a K-Means approach, to try and discern whether our dataset contained any identifiable groups with relevant features that would be worth predicting.

Preprocessing:

Before implementing the K-Means model, some processing had to be done on the dataset. Below is a snippet of what the dataset looked like before processing:

```
df.head()
```

	name	tagline	summary	description	year_founded	website	city	region	country	postal_cod
0	Valera Health	Your Path to Wellness, Just a Click Away	Valera Health, based in New York, is a mental ...	Valera Health operates as a tele-mental health...	2015.0	https://valerahealth.com	Brooklyn	New York	United States	1124
1	Bestow	Protecting Life, Simplified	Bestow is a Texas-based company that offers fa...	Bestow operates as an insurance technology com...	2017.0	https://bestow.com	Dallas	Texas	United States	7522
2	PlainID	Secure Your Identity, Empower Your Business	PlainID is a company based in Tel Aviv that sp...	PlainID is an Identity Security Posture Manage...	2014.0	https://plainid.com	Tel Aviv	NaN	Israel	678913
3	Snapcart	Innovating Connections,	Snapcart is a commercial	Snapcart specializes	2015.0	https://snapcart.global	Jakarta	NaN	Indonesia	1394

Note that this snippet does not contain all columns of the dataset.

Transformation

Notice the presence of many columns containing long strings or dictionaries of text information. K-Means models can only be conducted with numerical data. Our data contained a column called ‘sentiment’ with string values containing a numerical score along with explanations and other indicators of sentiment, such as numbers of positive/negative words within each article. It was decided that sentiment score might be a useful attribute to try and create clusters from, so the numerical part of this string needed to be separated. This numerical score was first extracted and put into a new column with the following code:

```
import ast

#extract 'sentimentScore' from the 'sentiment' string
def extract_sentiment_score(sentiment_str):
    try:
        sentiment_dict = ast.literal_eval(sentiment_str)
        return sentiment_dict.get('sentimentScore', None)
    except (ValueError, SyntaxError):
        return None

#create a new column 'sentimentScore'
df['sentimentScore'] = df['sentiment'].apply(extract_sentiment_score)
```

After creating this column, all numerical columns were defined and isolated:

```
#Defining numerical columns to work with
numerical_cols = [
    'year_founded',
    'investor_count',
    'mosaic_change',
    'funding_total_millions',
    'last_funding_millions',
    'funding_count',
    'sentimentScore'
]
```

Two significant outliers were also identified, and dropped from the dataset:

```
#Significant outliers - dropped from the dataset
df.drop([9,3313], inplace = True)
```

It is standard practice to implement scaling of values for K-Means clustering models, so a StandardScaler from Scikit-learn was also used. Below shows a snippet of the implementation of scaling, and how the scaled data looked before K-Means was performed:

```
#Using a StandardScaler - common good practice for K-means
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[numerical_cols])
```

```
X_scaled
```

```
array([[-0.50947838,  1.24346422, -0.92501958, ..., -0.35479021,
         0.73804362,  0.7131997 ],
       [ 0.1113885 , -0.47841181, -0.94445878, ...,  0.47999504,
        -0.1943735 , -0.22430704],
       [-0.81991182, -0.19143247, -0.95417839, ...,  0.54855494,
        -0.1943735 ,  0.7131997 ],
       ...,
       [ 0.73225538, -0.76539114, -1.48875655, ...,  0.08234766,
        -0.81598491, -3.50558062],
       [ 0.1113885 ,  0.95648488, -0.05025531, ...,  0.34287526,
        -0.81598491, -0.03680569],
       [ 0.73225538,  0.09554687, -1.51791536, ..., -0.39044136,
         0.73804362, -0.41180839]])
```

Modeling:

K-Means Clustering

First, we much implement K-Means and select a value for K. Many values of K were tested, and the clusters were visualized in two dimensions using principal component analysis. From visualizations that were only possible from reducing the dimensions to two, a value for K = 3 was chosen. Since the outcome of the k-means clustering was, for us, a somewhat exploratory measure, we believe this makes sense over other potential values of k which could have been chosen based off of other metrics, as we will discuss below.

A K-means model with 3 clusters divided the data into the following groups:

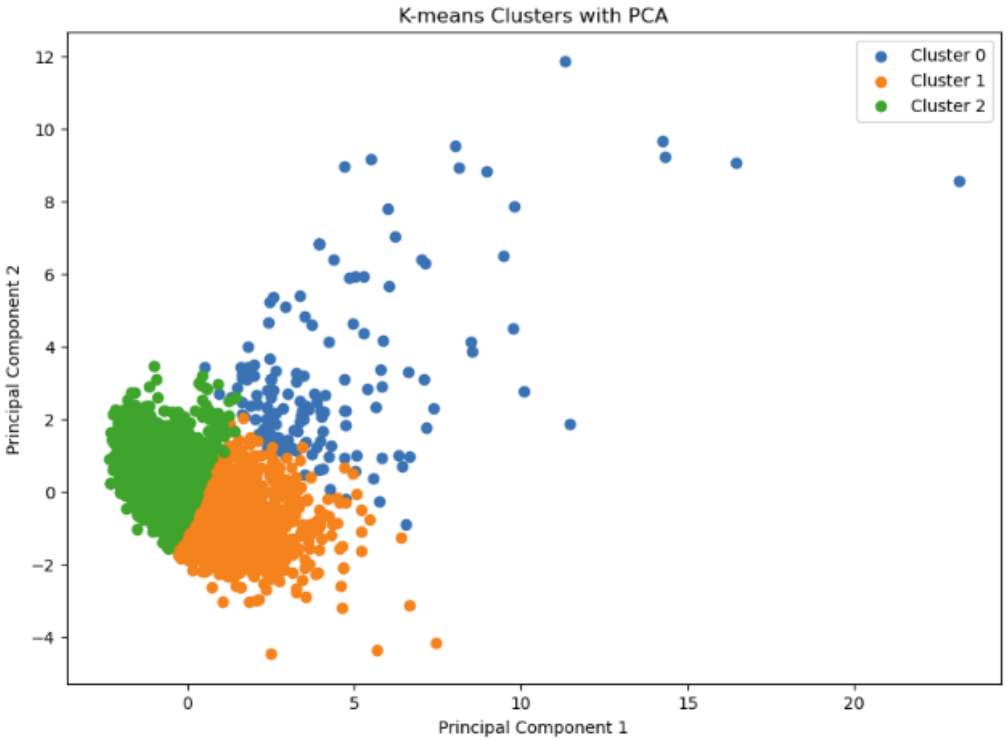
```
#Printing the number of values in each cluster
print(df['cluster'].value_counts())
```

```
2    3414
1    1940
0     153
Name: cluster, dtype: int64
```

Using Principal Component Analysis, the dimensions of each of these groups was reduced to two for visualization purposes:

```
from sklearn.decomposition import PCA

#reduce to 2 principal components for visualization purposes
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)

#create a DataFrame for the principal components
pc_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pc_df['cluster'] = kmeans.labels_

#plot the clusters in 2D
plt.figure(figsize=(10, 7))
for cluster in range(k):
    cluster_data = pc_df[pc_df['cluster'] == cluster]
    plt.scatter(cluster_data['PC1'], cluster_data['PC2'], label=f'Cluster {cluster}')

plt.title('K-means Clusters with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

The visualizations of these dimension-reduced clusters, with each cluster colored differently, is as follows:



We can see that most of the data belongs to either Cluster 1 or 2, and these clusters are close to each other when visualized. The most interesting cluster for our analysis is Cluster 0. Though this cluster contains the least amount of data points, and those data points have a large spread, they clearly are distinguished from Clusters 1 and 2, having the highest values of the two principal components.

We can understand this difference further by looking at means of the attributes of the data in each cluster:

```
#Visualizing means of values within each cluster
cluster_profiles = df.groupby('cluster')[numerical_cols].mean()
print(cluster_profiles)
```

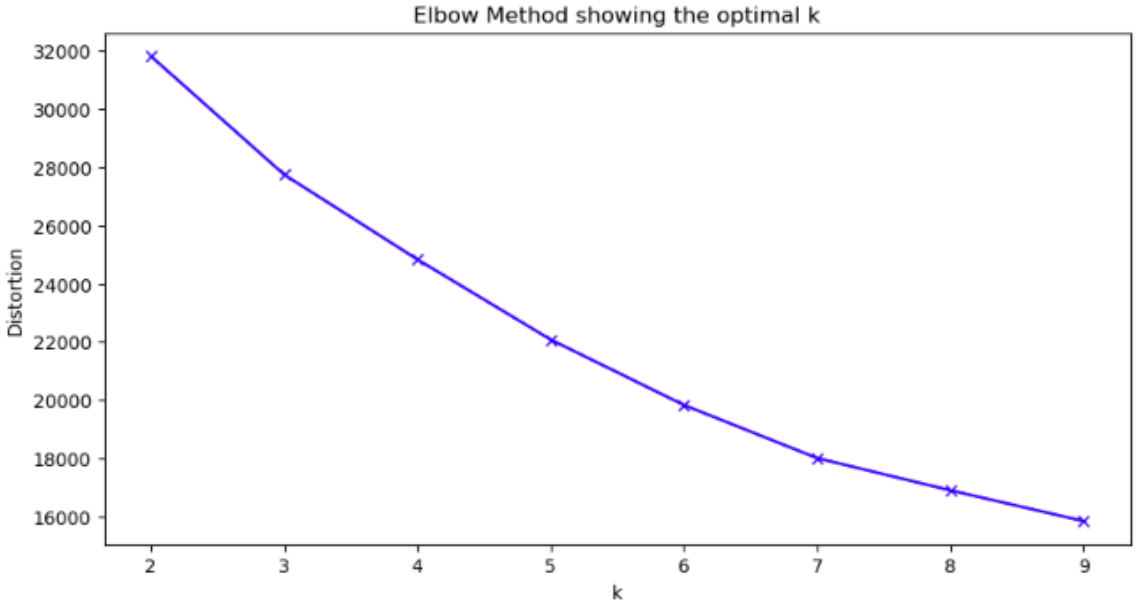
```
         year_founded  investor_count  mosaic_change  funding_total_millions  \
cluster
0         2015.209150       19.686275     -17.058824             1160.205882
1         2014.214433       15.649485     -37.505670              130.425923
2         2018.084359        8.507616     -13.135325               43.507496

         last_funding_millions  funding_count  sentimentScore
cluster
0                   342.752026       8.607843       85.032680
1                    33.632320       8.446392       92.418557
2                    21.976216       3.888694       92.707674
```

We can see that Cluster 0 is characterized by the highest means of investor counts, total funding, the last funding amount (the last two by far), and, interestingly, a lower sentiment score than the other two clusters. We also note the similarity with the year founded and funding count as Cluster 1, and a lower magnitude mosaic change than Cluster 1. This paints the following picture: Cluster 0 appears to contain startups that have been around for 9-10 years, but, in contrast to startups in Cluster 1, have more investors and vastly more amounts of funding (though they have had the same rounds of funding as those startups in Cluster 1), a smaller change in mosaic score than Cluster 1, and a noticeably lower sentiment score than either other cluster. We might expect Cluster 2 to have the lowest numbers of investors and funding numbers, since their mean founding year is much more recent. However, the founding year for startups in Cluster 0 is very close to that of Cluster 1, and yet financial metrics for startups in Cluster 0 are significantly better. The difference in mean sentiment score for Cluster 0 startups is particularly intriguing. It may be that the more successful startups in Cluster 0 simply have more press written about them as opposed to other startups, and more press may contain negative sentiment. This could be a direct implementation of the old maxim: “any press is good press”. In other words, more successful startups may have more negative things written about them, but at least this means they get more media coverage, and this translates to much more funding than other startups.

With all this in mind, we decided the prediction of many of our subsequent models would be to predict the cluster grouping that was generated from this K-means model.

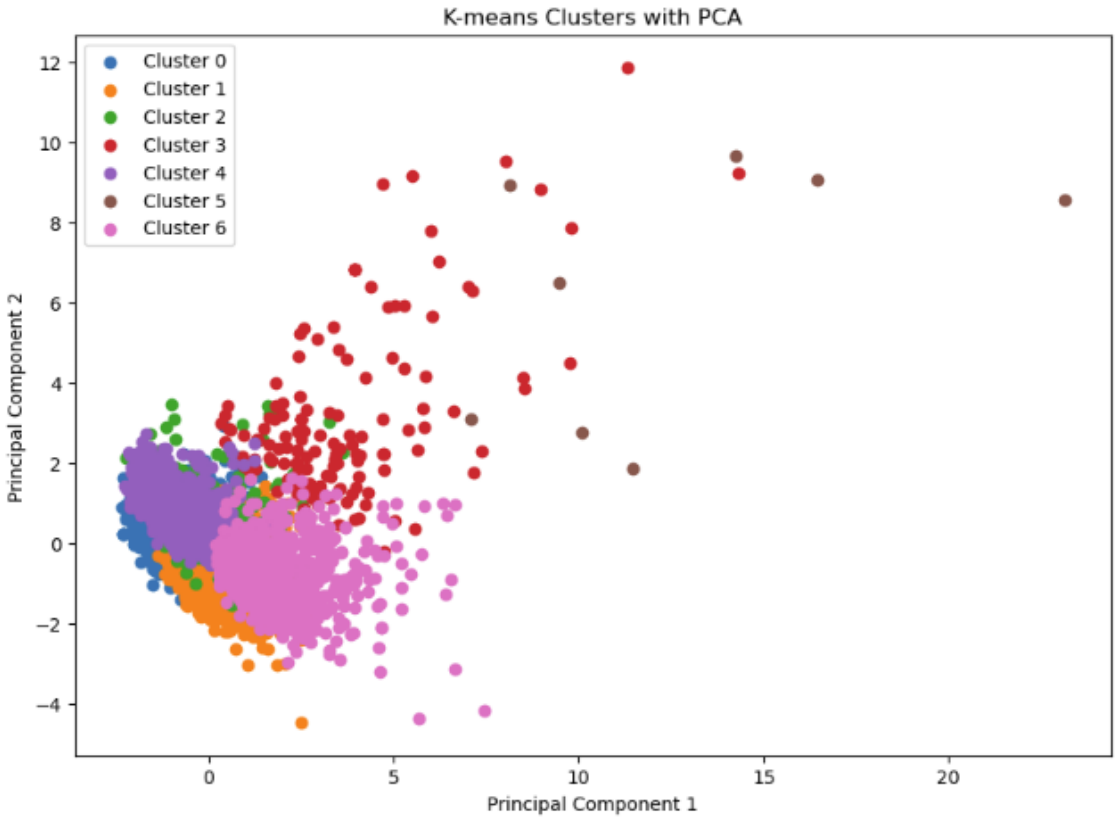
In terms of hyperparameter tuning, other values for K were considered but were not used due to being less interpretable for further model building and prediction.

A graph using the elbow method for finding an optimal value of K was generated for consideration:



We can see there is not any definitive value of K for which the slope of this curve drastically decreases from one value of K to the next – the decrease in distortion is gradual throughout. The largest decreases in the slope appear to occur between k = 2 and k = 3, and between k = 6 and k = 7.

A visualization of using k = 7 with PCA to reduce dimensions to 2 is as follows:



We can see a huge amount of variability within the large ball of data that was previously only divided into 2 clusters for K = 3, and the former cluster for Cluster 0 at K = 3 is also divided, with brown points of Cluster 5 being mixed in to the red points of Cluster 3 in the above graph.

Conclusion: Though it may be interesting to use such a clustering in a further analysis beyond the scope of this project, it was decided that it would be sufficient to use K = 3 and attempt to find ways to predict any data found to be in Cluster 0, as this appears to have enough interesting differences (from the examination of the means of attributes for data in that cluster) to make predictive models focusing mostly on finding startups that fit into this cluster.

Section 2

File: Will_DM_NN.ipynb

Goal: Use neural network to predict clusters found with K-Means for data in `primary.csv`.

Reason: After creating the clusters, we created a neural network to make predictions of which companies belong to which clusters. This network looks at numerical features when making a decision.

Preprocessing:

A new dataset was created after K-Means clustering was performed, including the cluster of each datapoint. The below image is a snippet of that data:

```
df.head()
```

ount	last_funding_type	last_funding_date	sentiment	articles	sentimentScore	cluster
8	Series B - II	2024-04-09	{'sentimentScore': 100, 'cntPositive': 15, 'cn...	[{'contentId': 'ab325015a7bbae7beefc4833b73696...	100	1
5	Series C	2020-12-16	{'sentimentScore': 90, 'cntPositive': 48, 'cnt...	[{'contentId': 'b633176d-6378-46d0-9b15-d2b611...	90	2
5	Series C	2021-12-21	{'sentimentScore': 100, 'cntPositive': 21, 'cn...	[{'contentId': '29c3c127-069d-4be7-aaf6-3a89bb...	100	1
4	Series A	2017-10-25	{'sentimentScore': 92, 'cntPositive': 7, 'cntN...	[{'contentId': '5ffd28ae-d4f3-46e2-a3b1-526c96...	92	2

Note the cluster column on the right-hand side, showing the cluster each data point belongs to.

Transformation

Like before, and to keep the model consistent, we used only numerical columns for the neural network:

```
#Defining numerical columns to work with
numerical_cols = [
    'year_founded',
    'investor_count',
    'mosaic_change',
    'funding_total_millions',
    'last_funding_millions',
    'funding_count',
    'sentimentScore'
]
```

We split the data into training and testing sets, and, like before, used standard scaling, as this is generally best practice for training neural networks:

```
#select 'cluster' as the target for prediction
X = df[numerical_cols[:-1]]  # All columns except 'cluster'
y = df['cluster']

#create training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

#feature scaling is best practice for neural networks
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Below is a snippet of how the training data looked before the neural network model was trained on it:

```
X_train_scaled
```

```
array([[ 0.73610856,  0.52188196, -0.29576101, ..., -0.30244367,
         -0.1984724 ,  0.71804358],
       [ 1.04515909, -1.19744994, -1.06993434, ..., -0.37226874,
        -0.81896277,  0.71804358],
       [-2.04534624, -0.48106165, -1.00219418, ..., -0.30244367,
         0.42201797, -2.43996214],
       ...,
       [ 0.11800749,  0.66515962,  0.01390832, ...,  2.19729374,
         0.11177278,  0.43095215],
       [ 0.73610856,  0.3786043 , -0.16028068, ..., -0.24825942,
        -0.1984724 ,  0.71804358],
       [-0.19104304,  1.52482557, -0.16028068, ...,  1.37307864,
         1.35275352,  0.52664929]])
```

Modeling:

Neural Network

A very basic and straightforward neural network model with two hidden layers and an output layer was created using the Keras library, which runs on top of TensorFlow, another python library. This was used as a classifier to predict the three classes corresponding to the three k-means clusters we found previously. We decided to use a standard ReLU activation function for hidden layers and a softmax function for the output layer, because these are suitable for basic classification models such as this.

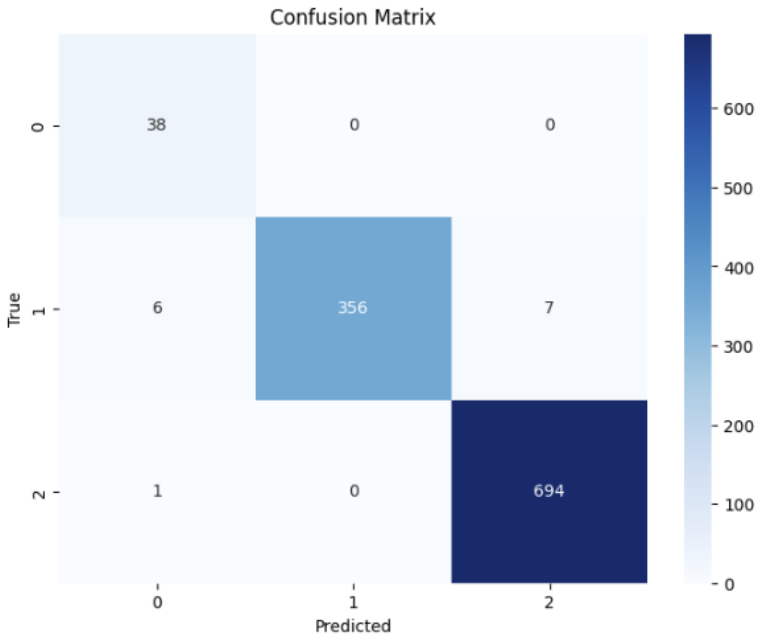
```
#neural network model using keras. Note the very basic model with only three layers and fairly standard
#activations for each
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    layers.Dense(32, activation='relu'),
    layers.Dense(3, activation='softmax')  #since there are 3 classes
])

#compiling the model with a typical optimizer and loss function
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

#training the model on the data
history = model.fit(
    X_train_scaled, y_train,
    epochs=9,
    batch_size=32,
    validation_split=0.1,          #Note a validation set is used
    verbose=1
)
```

We omitted more complex techniques such as regularization, dropout, or batch normalization, since this was meant to be a baseline model that could potentially be optimized later. The architecture of the model is simple, with only three layers: one hidden ReLU layer with 64 neurons, another hidden ReLU layer with 32 neurons, and an output layer with softmax activation with 3 neurons, suitable for classification. The optimizer (Adam) and loss function (sparse categorical cross-entropy) are both commonly used. The sparse categorical cross entropy function is able to be used here since the labels for the clusters are provided as integers.

```
#model evaluation for each epoch
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6785 - loss: 0.7923 - val_accuracy: 0.9524 - val_loss: 0.1884
Epoch 2/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9791 - loss: 0.1314 - val_accuracy: 0.9615 - val_loss: 0.1111
Epoch 3/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9824 - loss: 0.0793 - val_accuracy: 0.9728 - val_loss: 0.0761
Epoch 4/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9866 - loss: 0.0625 - val_accuracy: 0.9887 - val_loss: 0.0515
Epoch 5/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9881 - loss: 0.0497 - val_accuracy: 0.9909 - val_loss: 0.0479
Epoch 6/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9911 - loss: 0.0406 - val_accuracy: 0.9819 - val_loss: 0.0462
Epoch 7/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9904 - loss: 0.0425 - val_accuracy: 0.9887 - val_loss: 0.0385
Epoch 8/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9925 - loss: 0.0290 - val_accuracy: 0.9909 - val_loss: 0.0355
Epoch 9/9
124/124 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9929 - loss: 0.0294 - val_accuracy: 0.9955 - val_loss: 0.0333
Test Accuracy: 0.9873
```

We monitored accuracy during training and evaluation. Nine epochs were chosen after initially trying 30 and noticing an area where the loss function ceased to decrease around nine epochs. Note the training data was also split into a 0.1 size validation set to help prevent overfitting. Overall, this model was created to be extremely basic and as a baseline for further optimization if deemed necessary. Below is the code used, the output of each epoch, and a confusion matrix to evaluate performance:



Surprisingly, we saw an extremely high rate of accuracy of prediction even though no class balancing was performed on the data and despite the simplistic nature of the neural network. Note the final accuracy on the test set was 0.9873. We can also examine the confusion matrix above. We can see the only errors are 6 predictions that an item would be in class 0, but was actually in class 1, 7 items predicted to be in class 2, but were actually in class 1, and only 1 item predicted to be in class 0, but was actually in class 2. Since Class 0 is our interesting class and the one that would likely yield startups most likely to receive large amounts of funding, we likely don't care much about incorrect predictions of class 2 items that were actually in class 1. We do see that this neural network tends to place some items in class 0 that do not belong there – considering the rarity of a startup actually belonging to class 0, this may be a bit concerning should this model be introduced to new data (7 wrong predictions out of 45 predicted to be in Class 0 would give us a 15% likelihood that our model might tell us to invest in a Class 0 company that is not actually classified as one). Of course, since we are predicting clusters and not actual performance of a company, it is entirely possible that a Class 1 or 2 company might contain some metrics like funding amount that would be favorable, even if they are not within our 'interesting' cluster. And, regardless, an 85% accurate prediction for Class 0 is still very good considering how imbalanced the classes are in this case.

Conclusion: Our model performs extremely well without the need for class balancing or adjustment of a very simple neural network. With such an excellent accuracy score, we should be wary of possible overfitting. The best way to uncover overfitting would be to find out how well this model performs on brand-new, unseen data.

Section 3

File: Michael-ML-Clusters.ipynb

Goal: Predict clusters found with K-Means using classification on `primary.csv`.

Reason: Given an unseen sample, we want to be able to predict its cluster based on its features. This section is not necessarily applicable to real world problems, because 1) we could have just added the point to the dataset then used K-Means to predict its cluster, and 2) there would be few unseen samples with all the features used to train our model. Nonetheless, this section helps us test our processing pipeline and model outputs, so we feel it is necessary to showcase.

Preprocessing:

Preprocessing is performed on both the training set and test set. Typically, encodings and scalings are first fit on the training set, then the data in both the training set and test set are transformed. By doing this, we ensure there is no data leakage, and that the test set can be processed by the model.

1. Loading

First, we load the dataset and split it into a feature matrix and a target vector. Then we split those into training and test sets.

Incomplete snippet of initial training dataset features:

```
X_train.head()
```

	name	tagline	summary	description	year_founded	website	city	region	country	postal_code	concepts	keywords	investor_count
3463	Kinetic	Protecting Your ...	Kinetic is a com...	Kinetic speciali...	2014	https://wearkine...	New York	New York	United States	10001	['Specializes in...	['Workforce Safe...	14
4561	Grow Credit	Elevate Your Cre...	Grow Credit is a...	Grow Credit offe...	2018	https://growcred...	Santa Monica	California	United States	90401	['Offers a free ...	['FinTech', 'Fin...	26
3968	Retirable	Design Your Drea...	Retirable is a c...	Retirable is a c...	2019	https://retirabl...	New York	New York	United States	10003	['Personalized r...	['Financial Plan...	8
5350	Flow Neuroscience	Elevate Your Min...	Flow Neuroscienc...	Flow Neuroscienc...	2016	https://flowneur...	Malmo	NaN	Sweden	211 40	['Medication-fre...	['MedTech', 'Men...	8
2052	thirdweb	Unleashing the F...	Thirdweb is a Ca...	Thirdweb is a fu...	2020	https://thirdweb...	San Francisco	California	United States	94123	['include:']	['Blockchain', '...	17

Initial training dataset target counts:

```
y_train.value_counts()
```

```
2    2379
1    1355
0     120
Name: cluster, dtype: int64
```

2. Encoding

Next, we encode the categorical features, which transforms them into numerical features, which is needed for training certain models. To do this, we look at the cardinality of each categorical feature (number of unique values).

```
X_train.select_dtypes(exclude=['number']).nunique()
```

```
name                3799
tagline             3757
summary             3799
description         3799
website             3799
city                 659
region               153
country               70
postal_code         1856
concepts            3732
keywords            3744
last_funding_type    131
last_funding_date   1168
sentiment           3796
articles            3799
dtype: int64
```

We encode the features accordingly. Here are the encodings:

Feature	Encoding	Reasoning
`name`, `tagline`, `summary`, `description`, `website`, `concepts`, `keywords`, `sentiment`, `articles`	Drop	Not useful for model
`city`, `postal_code`	Frequency encoding	High cardinality
`region`, `country`, `last_funding_type`	Label encoding	Moderate cardinality
`last_funding_date`	Split into day, month, year	Easier to deal with

3. Augmenting

Next, we can optionally augment our feature matrix. However, after training the models, we found that in this case, augmentations did not significantly affect model performance.

```
X_train_augmented = X_train_encoded.copy()
X_test_augmented = X_test_encoded.copy()

augment = False

if augment:
    for df in [X_train_augmented, X_test_augmented]:
        df['funding_per_investor'] = df['funding_total_millions'] / df['investor_count']
        df['funding_per_round'] = df['funding_total_millions'] / df['funding_count']
        df['last_funding_since_founded'] = df['last_funding_year'] - df['year_founded']
```

4. Balancing

Now, we can balance our data such that each class in the target vector is represented by roughly the same number of samples. To do this, we use SMOTE oversampling. Visit https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ for more details.

The classes are balanced using SMOTE oversampling such that for each minority class:

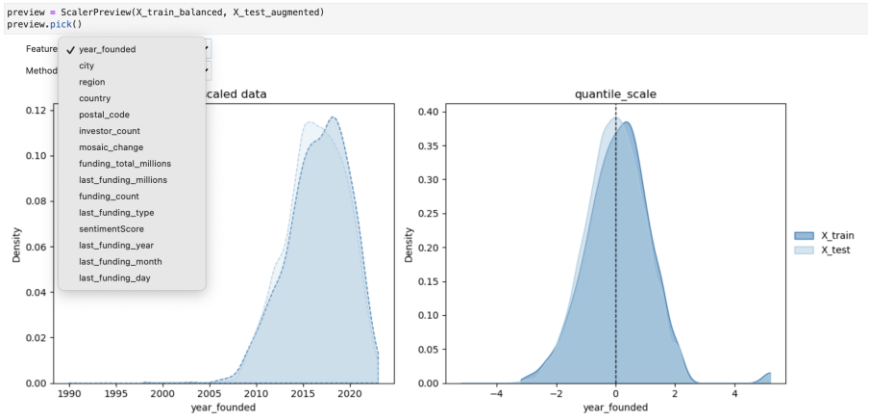
$$\text{minority class samples} \geq \text{ratio} \cdot \text{majority class samples}$$

```
X_train_balanced, y_train_balanced = preprocessor.balance_data(X_train_augmented, y_train, X_test_augmented, ratio=1/3)
```

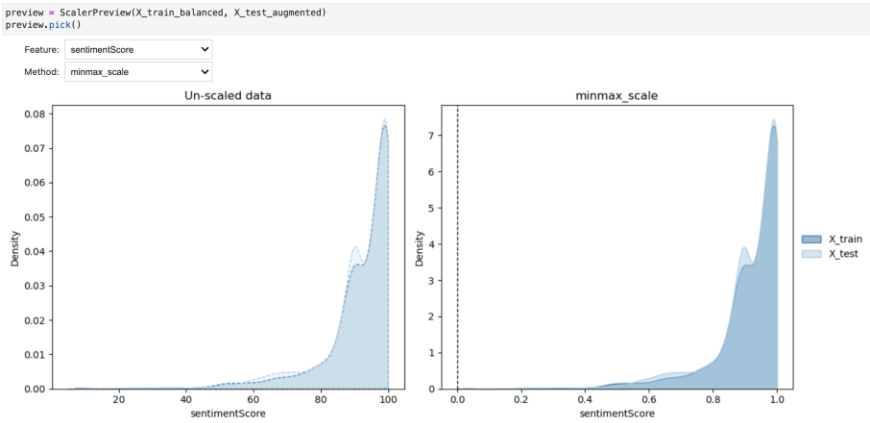
```
Class distribution before SMOTE:
Class 2: 2379
Class 1: 1355
Class 0: 120

Class distribution after SMOTE:
Class 2: 2379
Class 1: 1355
Class 0: 793
Balanced data!
```

5. Scaling

Finally, we scale our feature columns (all of which are now numerical). To help with this, we created a scale previewer. From a dropdown, we can select a feature and a scaling method, and the previewer will show how the data distributions for both the training and test set look before and after the feature is scaled. For example, `year_founded` can be scaled to be more normal using quantile scaling.



Another example is that `sentimentScore` can be scaled between 0 and 1 using min-max scaling.



After we previewed different scaling methods for each feature, we scaled the features accordingly. Here are the scalings:

Feature	Scaling	Reasoning
`funding_total_millions`, `last_funding_millions`, `funding_count`	Log Scale	Heavy right skew
`year_founded`	Standard Scale	Nearly normal
`mosaic_change`	Robust Scale	Nearly normal with outliers
`region`, `postal_code`, `investor_count`, `last_funding_type`, `sentimentScore`	Yeo-Johnson Scale	Brings skewed and multimodal distributions closer to normal
`city`, `country`, `last_funding_year`, `last_funding_month`, `last_funding_day`	Min-Max Scale	Brings distributions between 0 and 1

Now, our features are near 0 (helps distance models like K-Means and SVM).

```
X_train.head()
```

	year_founded	city	region	country	postal_code	investor_count	mosaic_change	funding_total_millions	last_funding_millions	funding_count
0	-0.738319	0.868263	0.507421	0.959459	1.706064	0.414914	-0.634146	2.943913	1.386294	2.564949
1	0.508137	0.035928	-1.151447	0.959459	0.476535	1.529194	0.000000	4.781641	2.397895	1.609438
2	0.819751	0.868263	0.507421	0.959459	1.893561	-0.479262	1.146341	2.858193	1.605430	1.945910
3	-0.115091	0.005988	1.193066	0.837838	-0.627624	-0.479262	0.000000	2.557227	2.639057	2.197225
4	1.131365	1.000000	-1.151447	0.959459	0.476535	0.750441	-0.341463	3.401197	3.218876	1.098612

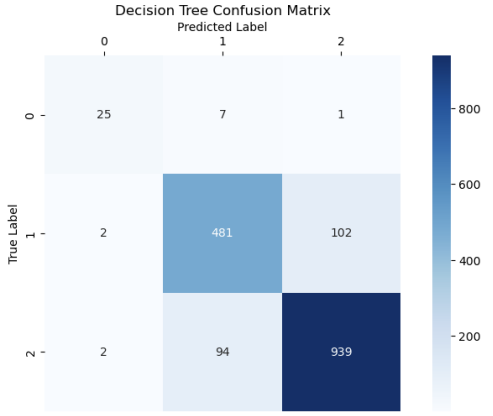
Modeling:

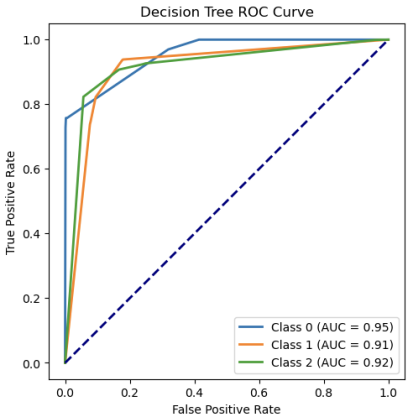
1. Decision Tree

We start off with a decision tree; it is interpretable and tests our data viability. We start with an initial maximum depth of 3. We have three classes, so we use One-vs-Rest when making the ROC Curves.

Decision Tree Performance

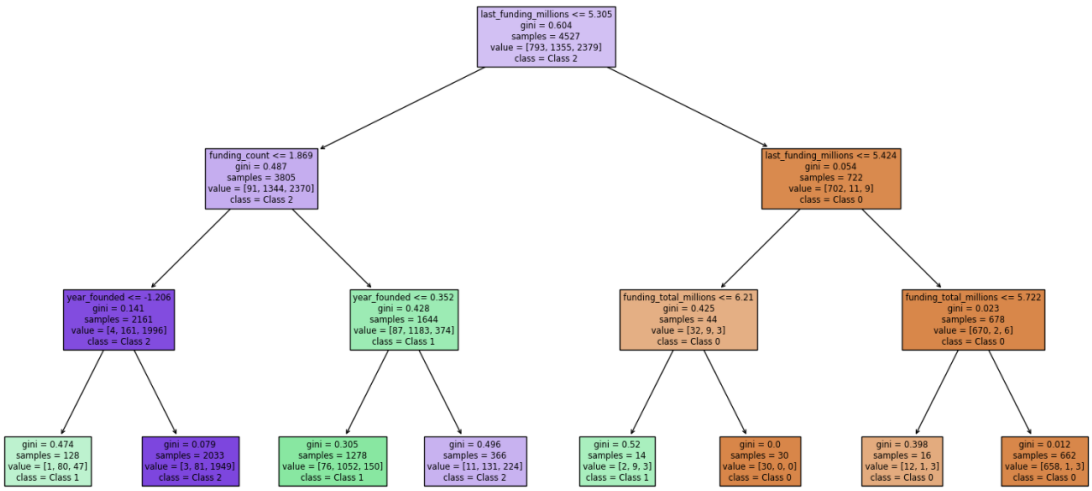
Metric	Score
Accuracy	0.874
Precision	0.863
Recall	0.829
F1 Score	0.845





Decision Tree Parameters

```
Depth: 3
```



We can see that our F1 Score is already quite high. We can also see that the feature that results in the greatest information gain when split on is `last_funding_millions`. It seems like funding information and founding information are the features the decision tree uses to determine which cluster a company belongs to. Using hyperparameter tuning, we can push our F1 Score even higher.

Hyperparameter tuning

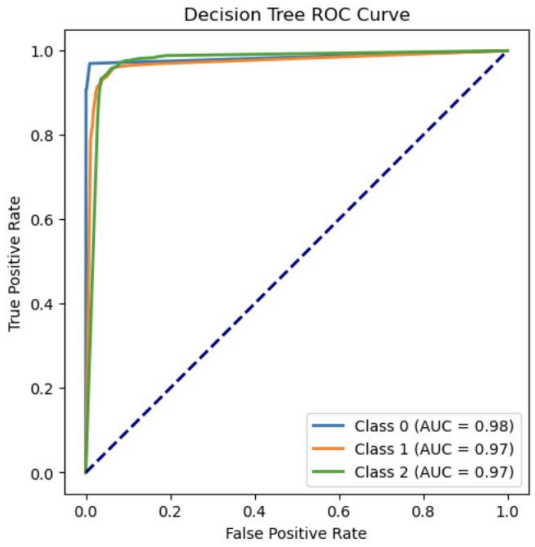
```
dt_param_grid = {
    'max_depth': [3, 5, 10, 15, None],
    'min_samples_split': [2, 50, 100],
    'min_samples_leaf': [1, 5, 10]
}

dt_best_params = dt_trainer.tune_hyperparameters(dt_param_grid, cv=5)
```

```
Best parameters found: {'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

We can see that with hyperparameter tuning, which uses 5-fold cross validation, our new maximum depth is 10. An F1 Score of 0.941 is excellent for a multiclass classifier, and an AUC near 1 for all the ROC curves shows that this model is well suited to predicting clusters.

Decision Tree Performance

Metric	Score
Accuracy	0.947
Precision	0.952
Recall	0.931
F1 Score	0.941



Decision Tree Parameters

```
Depth: 10
Tree with depth of 10 too big to display.
```

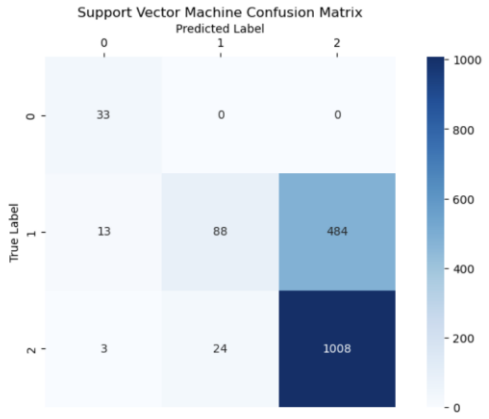
2. Support Vector Machine (SVM)

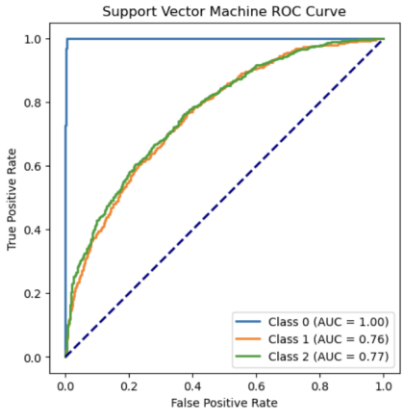
We follow the same approach with an SVM. By using an SVM, we test if our scaling is sufficient, since SVM is affected by distance, given that it tries to find a hyperplane dividing the data.

Here are the results without any scaling:

Support Vector Machine Performance

Metric	Score
Accuracy	0.683
Precision	0.712
Recall	0.708
F1 Score	0.618





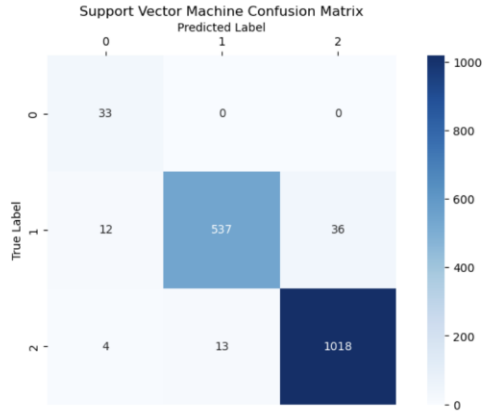
Support Vector Machine Parameters

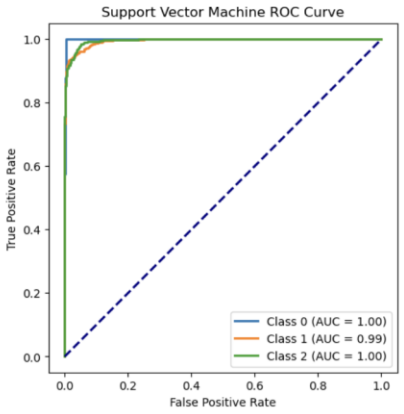
```
Kernel: rbf
Number of support vectors: 2775
```

Here are the results with scaling:

Support Vector Machine Performance

Metric	Score
Accuracy	0.961
Precision	0.872
Recall	0.967
F1 Score	0.909





Support Vector Machine Parameters

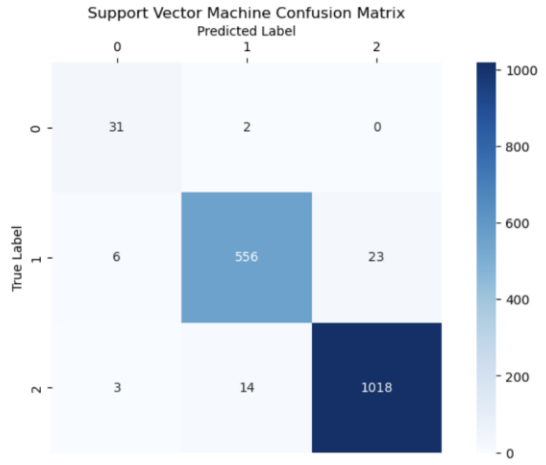
```
Kernel: rbf
Number of support vectors: 1054
```

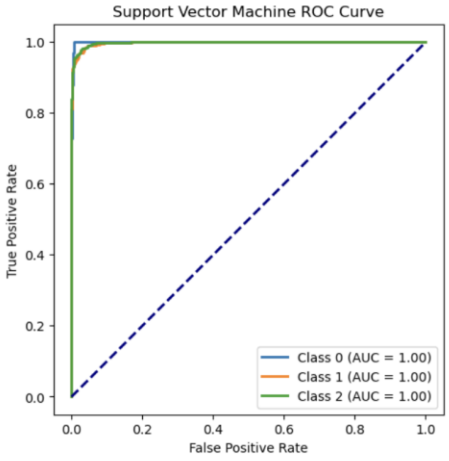
In addition to performing better, the SVM on the scaled data runs much faster.

With hyperparameter tuning, we can achieve an even greater F1 Score for the SVM on the scaled data.

Support Vector Machine Performance

Metric	Score
Accuracy	0.971
Precision	0.908
Recall	0.958
F1 Score	0.930





Support Vector Machine Parameters

```
Kernel: rbf
Number of support vectors: 531
```

Conclusion: Using a decision tree and support vector machine on our primary dataset to predict which cluster each company belongs to worked well. The tuned decision tree had an F1 Score of 0.941, and the tuned SVM had an F1 Score of 0.930. We also tested the viability of creating models on our datasets and developed a preprocessing pipeline. Finally, we saw firsthand the effect that scaling has on distance-based model accuracy. The next two sections use the companies from our primary dataset that had valuation information. The models created are more applicable for real-world inference, because we drop circular features, such as total funding when trying to make predictions. Instead, we use information that companies without much funding information would have, such as their industry, location, year founded, and latest funding.

Section 4

File: Michael-ML-Valuation-2Bins.ipynb

Goal: Using classification on `primary-with-valuation.csv`, predict whether a company will be in the lower half or upper half of valuations.

Reason: While the decision tree and SVM for predicting clusters were successful, they are not applicable to inferring company success on unseen, real-world data. This is because they use features that are either unavailable or circular in nature for most companies. For this section, we first divide the valuation amounts into two bins, low and high, and then use realistically obtainable features to predict whether a company will end up with a low or high valuation.

Preprocessing:

The preprocessing for this section follows a similar process to the preprocessing in section 1. The differences occur in the encoding and augmentation of the data.

1. Encoding

We drop features such as total funding and investor count; these are circular when predicting valuation.

```
encodings = {
    'ohe': [],
    'drop': ['name','tagline','summary','description','website','concepts','sentiment','articles','funding_total_millions','funding_count','investor_count'],
    'freq': ['city','postal_code'],
    'label': ['region','country','last_funding_type']
}

X_train_encoded, X_test_encoded = preprocessor.encode_data(encodings)
```

```
Encoded data!
```

2. Augmenting

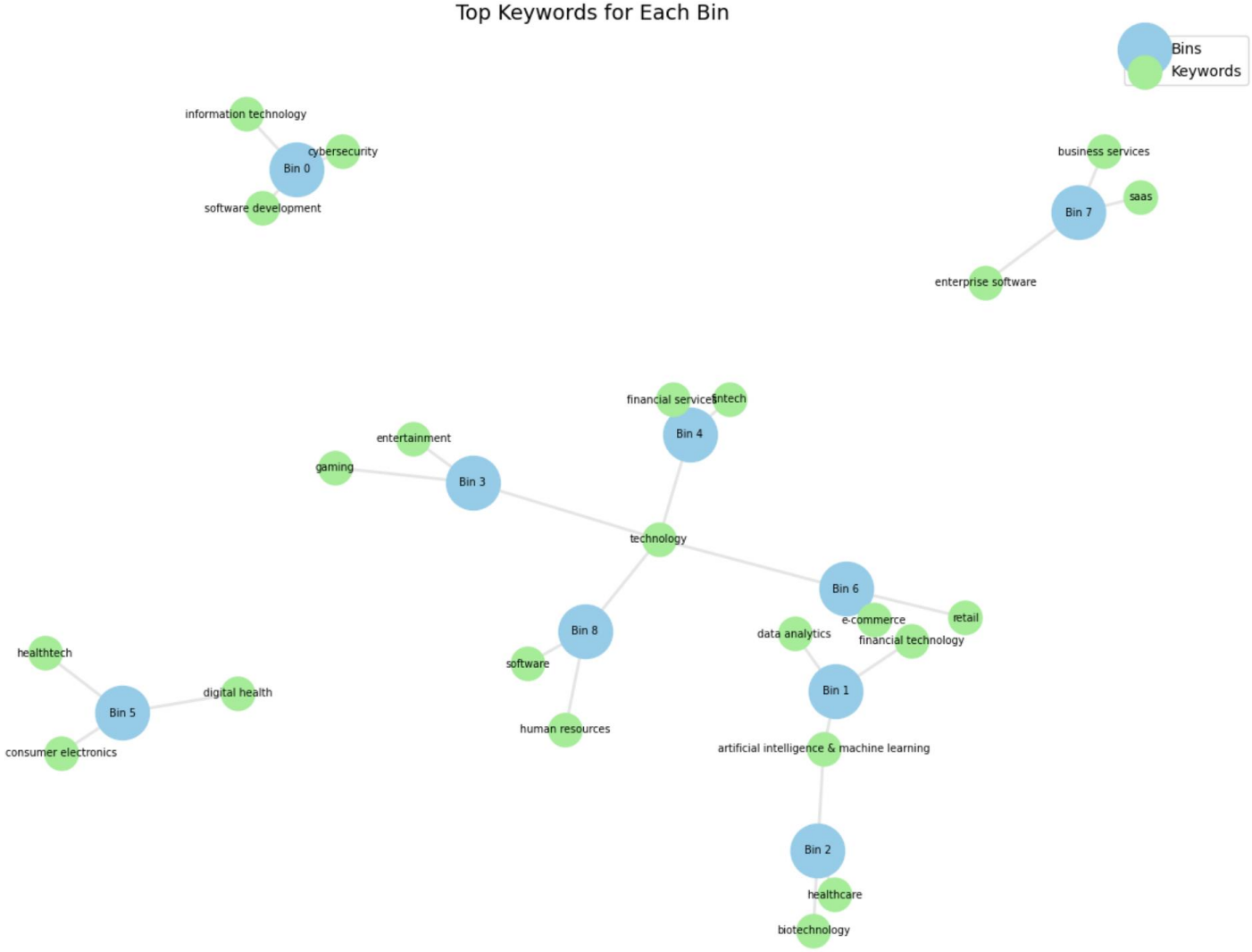
Here is the interesting part. Since we dropped so many features that were used by past models to determine clusters, we need to augment our dataset with new features that give it more information. Our approach to this was to encode the keywords column, which contains industry information for each company, so that the model can use industry area when classifying.

We start by one-hot encoding all the keywords in each list for each company using Scikit-learn's `MultiLabelBinarizer`. Then we use Scikit-learn's `TfidfTransformer` to reduce the effects of our sparse one-hot encoded matrix. Then, we use `KMeans` to cluster similar points in our one-hot encoded vector space. Finally, we one-hot encode the keyword bins formed by K-Means.

	keywords	keyword_bins
432	[your list of company industry categories to c...	1
582	[edtech, fintech, application software, ai as ...	1
442	[fintech, platform as a service (paas), e-comm...	6
675	[software, procurement, saas management, it se...	4
334	[data analytics, business intelligence, saas, ...	7

With the top 3 most common keywords in each bin, we use `networkx` to create a graph to connect keyword bins to keywords. Using our clustering algorithm, we have created clusters with keywords that are conceptually tangential, ie. Bin 0 has Cybersecurity, Software Development, and Information Technology, while Bin 6 has E-commerce, Retail, and Technology.

```
Top 3 most common keywords in each bin:
Bin 0:
        cybersecurity: 39
        software development: 25
        information technology: 21
Bin 1:
        artificial intelligence & machine learning: 53
        financial technology: 20
        data analytics: 19
Bin 2:
        healthcare: 22
        biotechnology: 17
        artificial intelligence & machine learning: 16
Bin 3:
        entertainment: 16
        gaming: 7
        technology: 7
Bin 4:
        financial services: 63
        fintech: 45
        technology: 14
Bin 5:
        digital health: 13
        consumer electronics: 11
        healthtech: 6
Bin 6:
        e-commerce: 63
        retail: 16
        technology: 10
Bin 7:
        saas: 35
        business services: 19
        enterprise software: 17
Bin 8:
        software: 26
        technology: 15
        human resources: 9
```



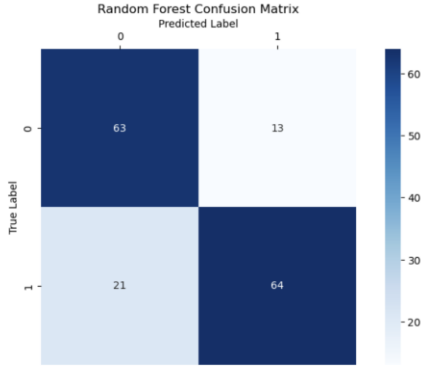
Modeling:

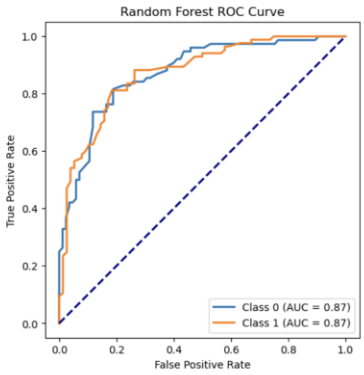
Random Forest

We use a random forest model because it avoids overfitting and had the best performance.

Random Forest Performance

Metric	Score
Accuracy	0.789
Precision	0.791
Recall	0.791
F1 Score	0.789





Random Forest Parameters

```
Depths: [17, 16, 20, 17, 15, 17, 15, 18, 14, 21, 19, 16, 15, 15, 16, 17, 13, 15, 17, 16, 14, 16, 14, 16, 15, 19, 15, 16, 17, 14, 16, 16, 13, 15, 14, 17, 2
0, 18, 16, 14, 17, 13, 16, 19, 15, 15, 15, 16, 16, 23, 16, 18, 18, 13, 16, 17, 15, 14, 17, 16, 20, 15, 15, 15, 14, 16, 14, 18, 14, 16, 14, 16, 15, 15, 15,
12, 15, 14, 16, 18, 16, 16, 13, 18, 15, 16, 16, 13, 20, 15, 16, 14, 15, 15, 17, 16, 15, 13, 17, 17]
Average depth: 15.89
```

With 100 trees, the F1 Score is decent, and with hyperparameter tuning becomes slightly better.

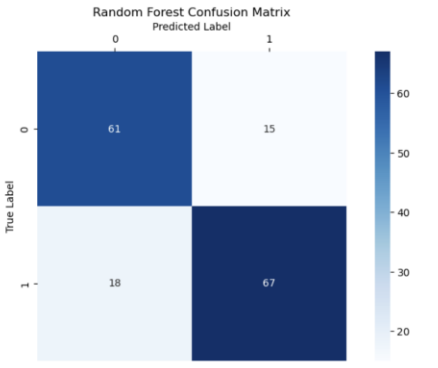
Hyperparameter tuning

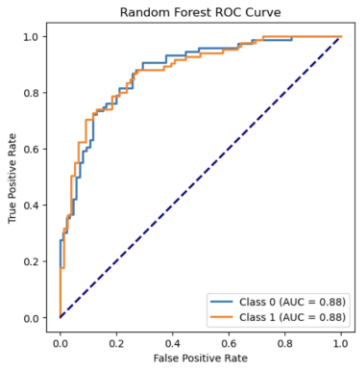
```
rf_param_grid = {
    'n_estimators': [100, 200, 400],
    'max_depth': [5, 10, 15, None]
}

rf_best_params = rf_trainer.tune_hyperparameters(rf_param_grid, cv=3)
```

```
Best parameters found: {'max_depth': 15, 'n_estimators': 200}
```

Random Forest Performance

Metric	Score
Accuracy	0.795
Precision	0.795
Recall	0.795
F1 Score	0.795





Random Forest Parameters

```
Depths: [15, 15, 15, 15, 15, 15, 15, 15, 14, 15, 15, 15, 15, 15, 15, 13, 15, 15, 15, 14, 15, 14, 15, 15, 15, 15, 15, 15, 14, 15, 15, 13, 15, 14, 15, 1
5, 15, 15, 14, 15, 13, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 13, 15, 15, 15, 14, 15, 15, 15, 15, 15, 14, 15, 14, 15, 14, 15, 14, 15, 15, 15, 15,
12, 15, 14, 15, 15, 15, 15, 13, 15, 15, 15, 15, 13, 15, 15, 15, 14, 15, 15, 15, 15, 15, 13, 15, 15, 14, 15, 14, 15, 15, 15, 15, 15, 15, 14, 14, 15, 15, 1
4, 15, 14, 15, 15, 15, 15, 15, 13, 14, 15, 15, 15, 14, 15, 14, 15, 15, 14, 15, 15, 15, 15, 15, 15, 13, 12, 15, 15, 15, 15, 14, 15, 15, 14, 14, 15, 15,
15, 14, 13, 15, 15, 15, 15, 15, 15, 14, 15, 13, 15, 15, 13, 15, 14, 14, 15, 13, 15, 15, 13, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 14, 1
5, 15, 14, 15, 15, 14, 15, 15, 15, 15]
Average depth: 14.66
```

Conclusion: Our random forest model did a good job of deciding whether a company would have low or high valuation based on our modified feature set. Using this model, given an unseen company, we could decide with moderate confidence whether they would end up with a low or high valuation.

Section 5

File: Michael-ML-Valuation-3Bins.ipynb

Goal: Using classification on `primary-with-valuation.csv`, predict whether a company will have a low, medium, or high valuation.

Reason: While the previous section created a model that predicts whether a company will end up with a low or high valuation, it would be even more helpful to predict whether a company will end up with a low, medium, or high valuation. In this section, we create three bins for valuation and train a classifier to predict these bins.

Preprocessing:

The preprocessing for this section is identical to that of the previous section.

Modeling:

Logistic Regression

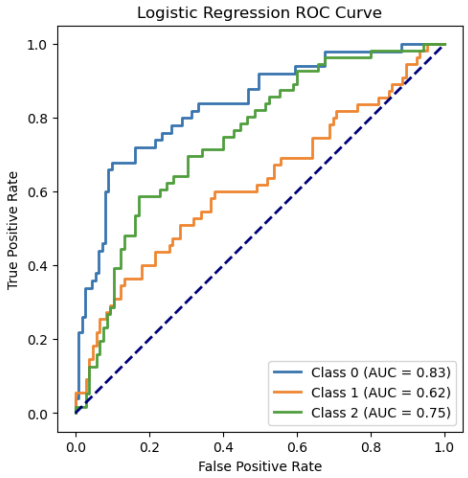
We use a logistic model because it is interpretable, which is useful for determining how our keyword bins influence the model. Here are the bins again:

```
Top 3 most common keywords in each bin:
Bin 0:
        cybersecurity: 39
        software development: 25
        information technology: 21
Bin 1:
        artificial intelligence & machine learning: 53
        financial technology: 20
        data analytics: 19
Bin 2:
        healthcare: 22
        biotechnology: 17
        artificial intelligence & machine learning: 16
Bin 3:
        entertainment: 16
        gaming: 7
        technology: 7
Bin 4:
        financial services: 63
        fintech: 45
        technology: 14
Bin 5:
        digital health: 13
        consumer electronics: 11
        healthtech: 6
Bin 6:
        e-commerce: 63
        retail: 16
        technology: 10
Bin 7:
        saas: 35
        business services: 19
        enterprise software: 17
Bin 8:
        software: 26
        technology: 15
        human resources: 9
```

Class 0, Class 1, and Class 2 correspond to low, medium, and high valuations. Below, in our tuned logistic regression model, we see that the model is more likely to classify companies in Bin 8 as Class 0 (lower valuation). Bin 8 includes software, technology, and human resources. Compared to other bins, it makes sense that human resources would have lower valuations. Looking at Bin 5, we see that the model is more likely to classify these companies as either Class 0 or Class 2. It is much less likely to classify them as Class 1. In other words, since this bin includes digital health, consumer electronics, and healthtech, the model sees healthcare companies as more likely to either have low or high valuations, with fewer in the middle.

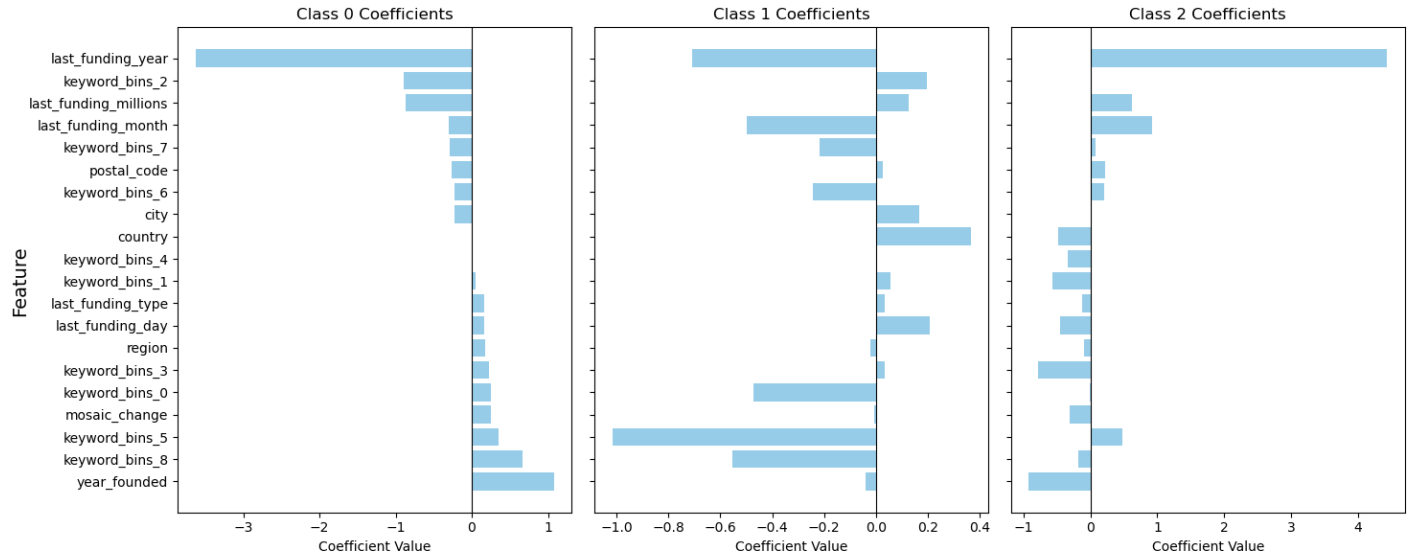
Logistic Regression Performance

Metric	Score
Accuracy	0.571
Precision	0.579
Recall	0.578
F1 Score	0.566



Logistic Regression Parameters

```
Class 0 Intercept: 5.185509107785031
Class 1 Intercept: -0.4333137637021516
Class 2 Intercept: -6.679878459347155
```



Conclusion: Our random forest model did an okay job of deciding whether a company would have low, medium, or high valuation. Using this model, given an unseen company, we could decide with some confidence whether they would end up having a low, medium, or high valuation.

Section 6

File: Mel ML Project Models.ipynb

Goal: Use regressions to predict the valuation of a company using data in `primary-with-valuation.csv`.

Reason: We used classifiers to predict valuation bins, but we did not yet have a way to predict actual valuation amounts given a feature set. In this section, we use regressors to accomplish this task.

Preprocessing:

Here is an incomplete snippet of the dataset before preprocessing:

me	tagline	summary	description	year_founded	website	city	region	country	postal_code	concepts	keywords
nID	Secure Your Identity, Empower Your Business	PlainID is a company based in Tel Aviv that sp...	PlainID is an Identity Security Posture Manage...	2014.0	https://plainid.com	Tel Aviv	NaN	Israel	6789139	['PlainID offers the Identity Security Posture...	['Cybersecurity', 'Identity Management', 'Data...
lice	Experience Money, Mastered.	slice is a financial technology company based ...	Slice operates as a financial technology compa...	2016.0	https://sliceit.com	NaN	Assam	India	781028	['Trusted by over 17 million Indians', 'Offers...	['FinTech', 'ConsumerTech', 'E-Commerce', 'Dig...
lice	Experience Money, Mastered.	slice is a financial technology company based ...	Slice operates as a financial technology compa...	2016.0	https://sliceit.com	NaN	Assam	India	781028	['Trusted by over 17 million Indians', 'Offers...	['FinTech', 'ConsumerTech', 'E-Commerce', 'Dig...
alth	Healing at Home, Comfort in Care	DispatchHealth is a Colorado-based company tha...	DispatchHealth is a healthcare company that fo...	2013.0	https://dispatchhealth.com	Denver	Colorado	United States	80907	["DispatchHealth offers on-demand urgent healt...	['Healthcare', 'Telemedicine', 'Home Health Ca...
ios	Step Into Adventure, Unleash Your Imagination	Survios is a California-based company that spe...	Survios is a leading VR games studio known for...	2013.0	https://survios.com	Marina del Rey	California	United States	90292	['Specializes in creating immersive VR experie...	['Augmented Reality', 'Gaming', 'Virtual Reali...

<u>Transformation</u>

As we were plotting regression models, we wanted to focus on the numerical attributes, so we dropped the columns that couldn't contribute, including the name, keywords, concepts, articles, and website.

With the remaining columns, we looked at the attribute types to decide if any encoding needed to be done if the columns weren't float or integer types. The city, valuation date, and sentiment score were object types, so we decided to use target encoding. We chose this over any other type because we assumed that there would be relationships between those attributes and the target feature, valuation in millions.

Before training our SVR model, we scaled the features for better performance and hypertuned the parameters using GridSearchCV, which printed out the best epsilon and C values. The SVR had a great R-squared score of around 99%, an improvement from before finetuning parameters. The KNN model performed much better without scaled features, so we used the unscaled training and test features. We also set the weights equal to distance because it gives more importance to closer neighbors to make

predictions. This also reduces the impact of farther neighbors, or outliers in this case that we did not want to drop. If we had dropped the outliers for the KNN, the R-squared score would have been better, but we would lose what we're looking for in the data. As more data is collected and trained, this data point may no longer be an outlier and improve the overall predictive power of the model.

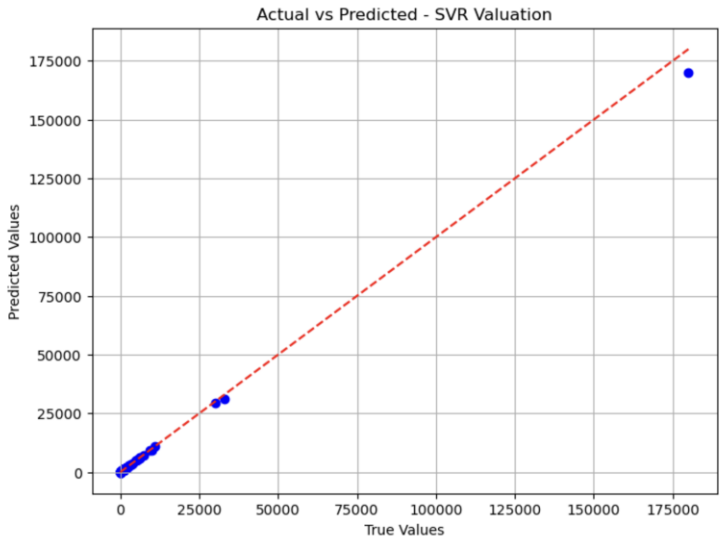
Here is the dataset after preprocessing:

year_founded	investor_count	mosaic_change	funding_total_millions	last_funding_millions	funding_count	city_target	valuation_date_target	sentiment_target
2014.0	10	-120.0	99.00	75.00	5	326.500000	36777.644000	48.0
2016.0	36	-76.0	390.50	7.77	21	0.000000	714.938696	1800.0
2016.0	36	-76.0	390.50	7.77	21	0.000000	714.938696	1800.0
2013.0	15	-47.0	740.96	259.00	8	705.444444	1624.958065	1700.0
2013.0	14	28.0	70.95	16.70	5	1276.666667	56.000000	30.0

Modeling:

1. Support Vector Regression (SVR)

We used a support vector regression to predict the valuation in millions of companies. First, we scaled the parameters then hyper-tuned them using GridSearchCV to find the best C and epsilon values. Epsilon defines a margin of tolerance for the model and C determines the trade-off between the bias and variance.

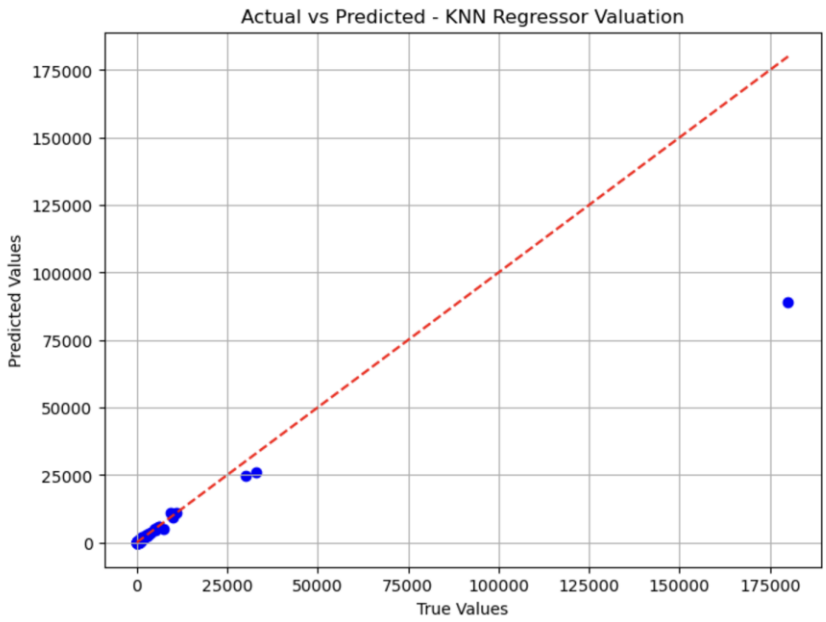



```
Mean Absolute Error (MAE): 111.03196638107684
Mean Squared Error (MSE): 637862.359483656
Root Mean Squared Error (RMSE): 798.6628572080061
R-squared (R^2): 0.9969844521450806
```

2. K-Nearest Neighbors (KNN) Regression

We made a plot and used the elbow method to determine the optimal number of clusters for high accuracy. We found the best clusters to be three and obtained decent r-squared values. The reason it is

not higher is due to an outlier in the data, but we wanted to keep the outlier to visualize the companies that performed better than expected.




```
Mean Absolute Error (MAE): 751.0012829743388
Mean Squared Error (MSE): 51766817.745064884
Root Mean Squared Error (RMSE): 7194.9161041019015
R-squared (R^2): 0.7552680231304131
```

Conclusion: Using SVR and KNN regressors, we were able to predict valuations with quite high accuracy. For the SVR regressor, we had an r-squared value of 0.997, which is very high. For the KNN regressor, we had an r-squared value of 0.755, which is moderately high.