Company Revenue Prediction

Chirobocea Mihail-Bogdan (407), Costea Razvan George (407), Ionescu Ariana-Georgiana (411), Marin Mircea-Mihai (407), Mihalcea Dragos Stefan (407), Nacu Andrei-Emilian(406), Popa Mihai-Cristian (407)

ABSTRACT

The aim of this project is to analyze company information from the lens of revenue prediction. The dataset consists of a number of company attributes, and we employ clustering algorithms in order to group them on the basis of their estimated revenues. Consequently, the clusters are used as labels for classification on the one hand, and regression for direct revenue prediction, on the other hand. Both the former and the latter approach allow further insight into company characteristics.

1. INTRODUCTION

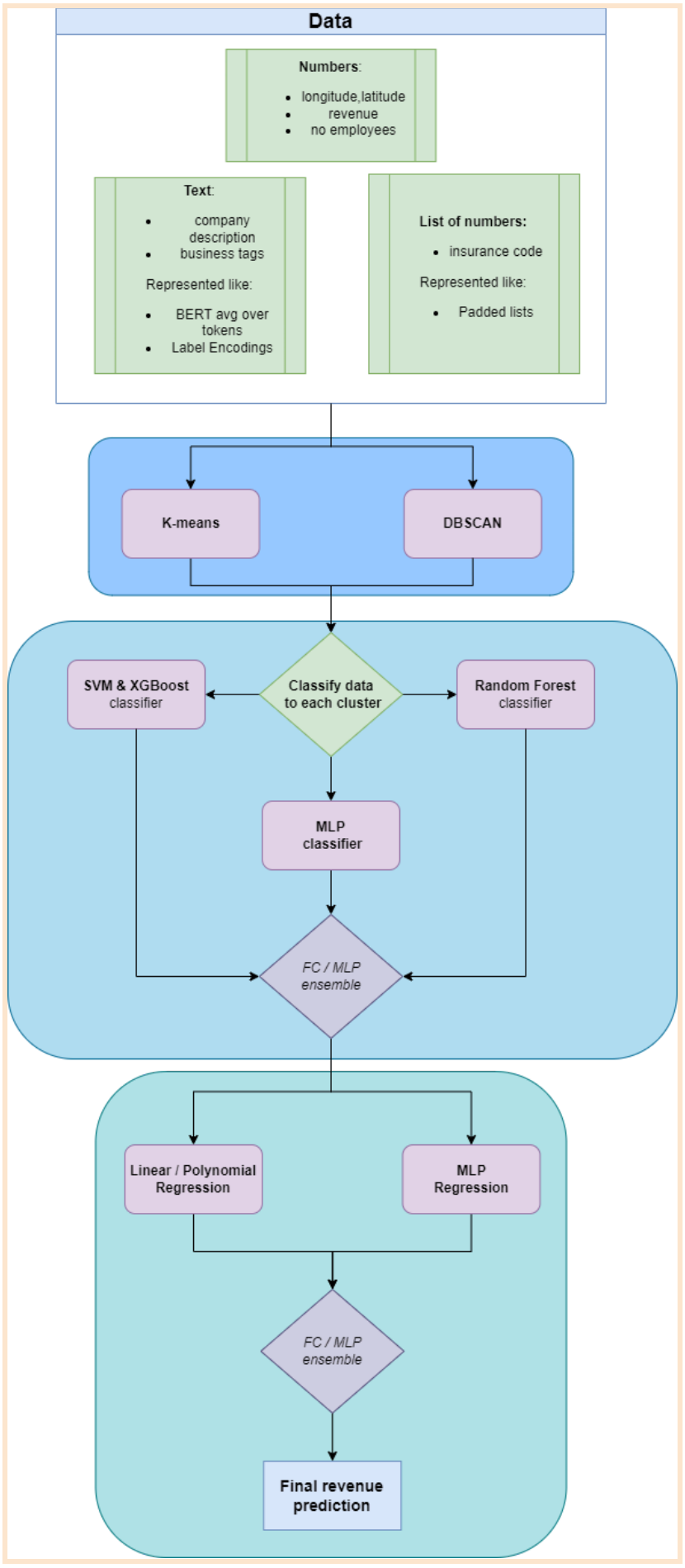
Our team project focuses on utilizing the Veridion API to analyze companies and implement a straightforward revenue prediction system. At the core of our approach is a systematic data processing pipeline.

Beginning with data collection, we extract real-time and historical information from the Veridion API. Subsequently, a carefully designed processing pipeline involves filtering and preprocessing to refine the data for meaningful analysis.

Recognizing the importance of revenue, we employ clustering techniques to categorize companies based on their revenue profiles. This clustering process organizes companies into ranges, facilitating structured analysis and anomaly detection for companies with exceptionally high revenues.

Building on the clustered revenue data, our project integrates a classification system that efficiently assigns companies to predefined revenue ranges. This classification step enhances the precision of subsequent regression analyses, allowing for nuanced predictions tailored to each range.

In the final phase, precise regression models are applied to each revenue range, providing detailed and accurate revenue predictions. This report outlines our modest yet effective approach, emphasizing the systematic nature of our pipeline.


Data
Numbers:
longitude,latitude
revenue
no employees
Text:
company description
business tags
Represented like:
BERT avg over tokens
Label Encodings
List of numbers:
insurance code
Represented like:
Padded lists
K-means
DBSCAN
SVM & XGBoost classifier
Classify data to each cluster
Random Forest classifier
MLP classifier
FC / MLP ensemble
Linear / Polynomial Regression
MLP Regression
FC / MLP ensemble
Final revenue prediction

2. DATA EXPLORATION

2.1 Feature Selection & Preprocessing

Our dataset of ~50,000 companies was derived using the Veridion API. Initially, a pool of ~500,000 companies was extracted. We employed a not-null filtering for certain attributes that might be relevant for the company revenue, such as:

- no of employees
- no of headquarters
- business tags
- company type
- technologies
- country
- year founded
- insurances

We removed irrelevant features like: name, email, phone, social media links and so on.

The dataset encompasses various attribute types, including:

- Numerical - integer, floats
- List of numericals - we retain the maximum dimension, with padding applied using zeros for the remaining values, ensuring data integrity.
- Text -
    - we used label encoding where text possibility was discrete, like the case of country code;
    - for text of indefinite length, like company description, we employed a BERT-like transformer to encode the text, averaging all tokens' encoding to obtain a whole sentence encoding.

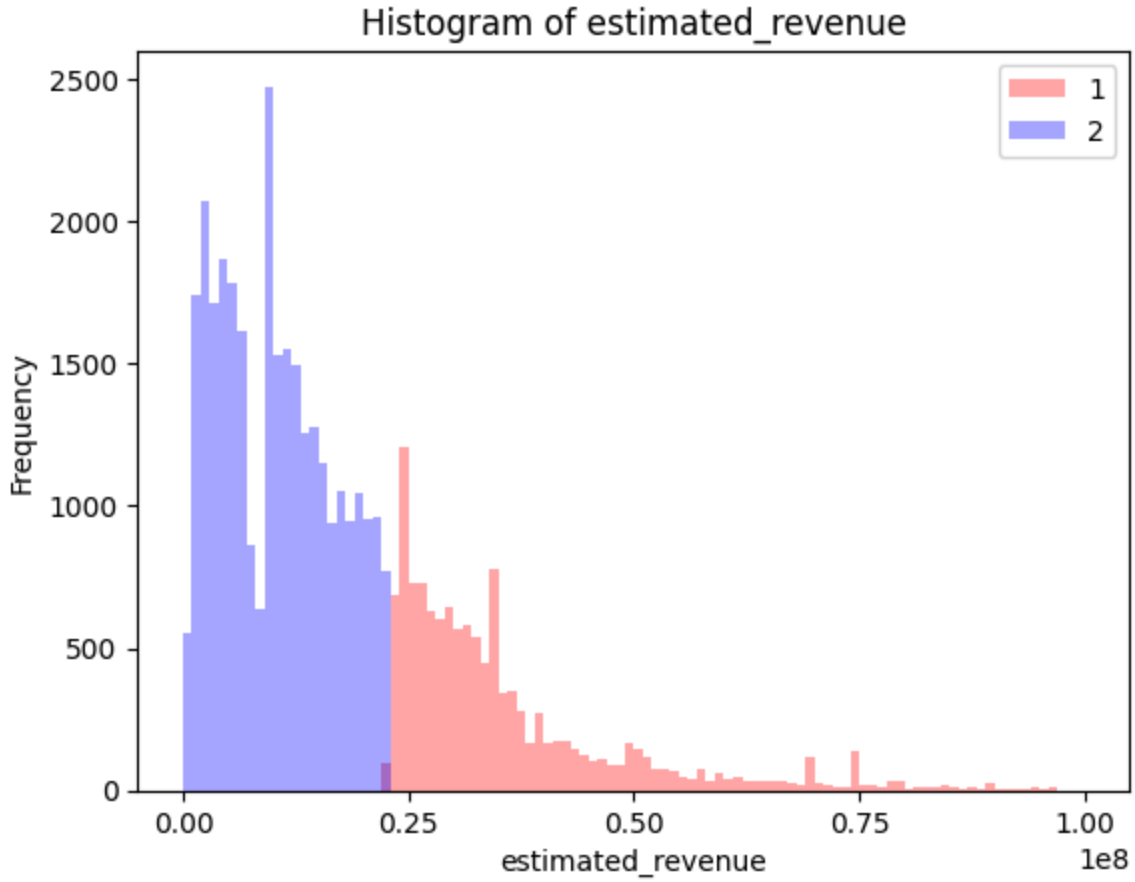
Numerical values, excluding those encoded with the transformer, underwent normalization using

min-max scaling, confined within the range of 0 to 1.

2.2 Statistics

The graphs here have been remade, in order to show the original values, before normalization took place

2.2.1 Revenue

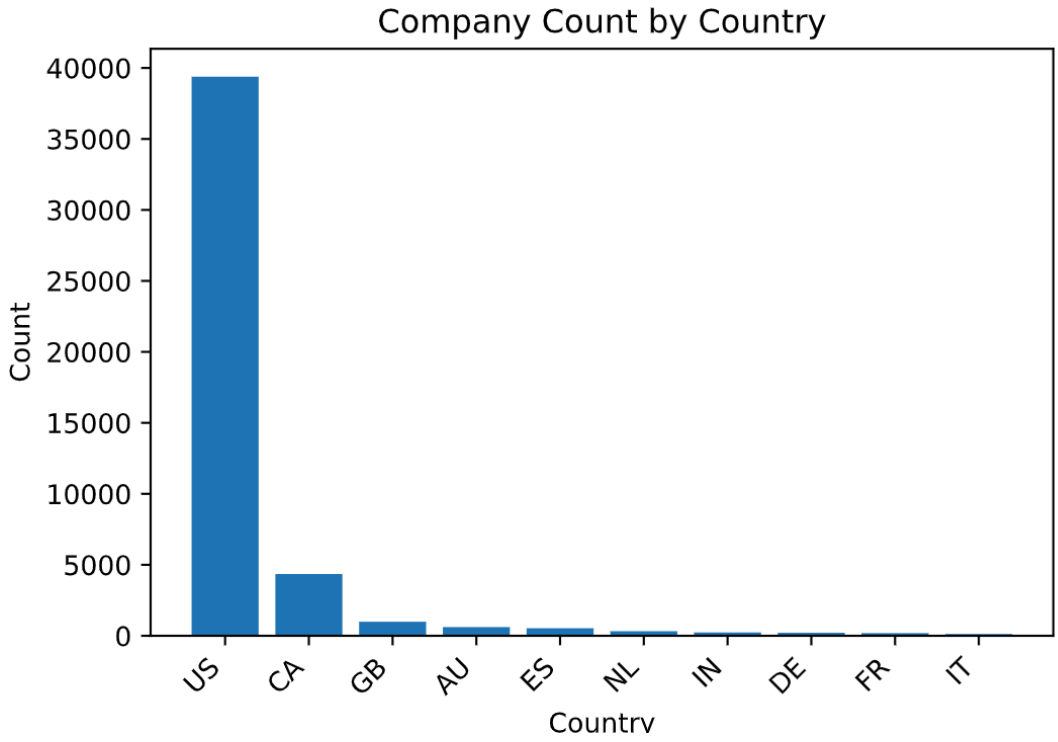



```
Statistics for Label 1:
Count: 12554
Mean: 35737299.665445
Standard Deviation: 13096872.463209
Minimum: 23000000.000000
25th Percentile: 26600000.000000
Median: 31800000.000000
75th Percentile: 39300000.000000
Maximum: 97000000.000000
```

```
Statistics for Label 2:
Count: 30274
Mean: 10464238.604050
Standard Deviation: 6242699.560970
Minimum: 7000.000000
25th Percentile: 4900000.000000
Median: 10000000.000000
75th Percentile: 15300000.000000
Maximum: 22900000.000000
```

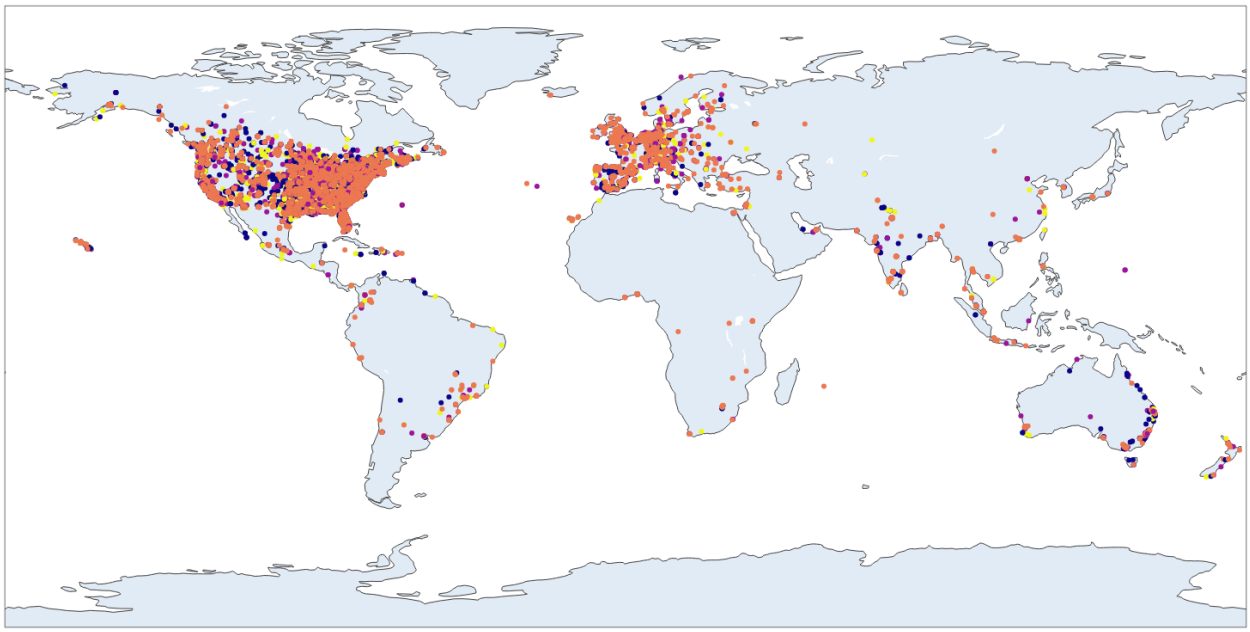
A majority of companies have orders of magnitude less revenue than the company with the highest revenue.

2.2.2 *Country code*



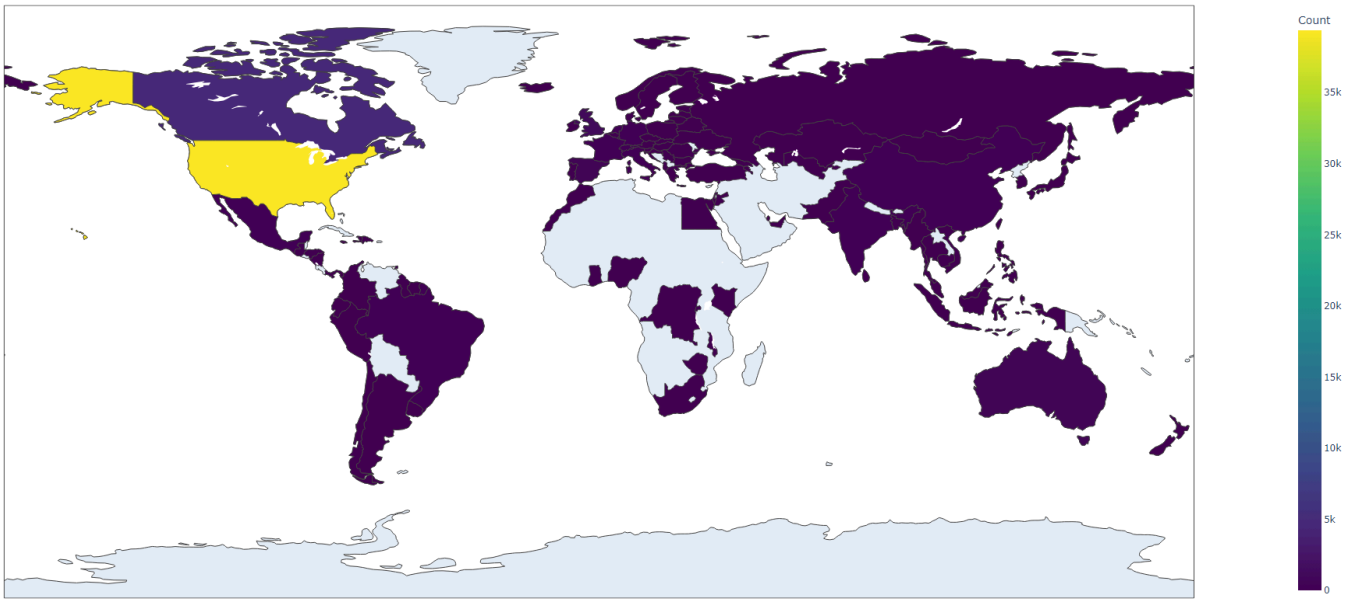
Most companies from the dataset were founded in the United States, though there are also many from Canada

2.2.3 *Maps*

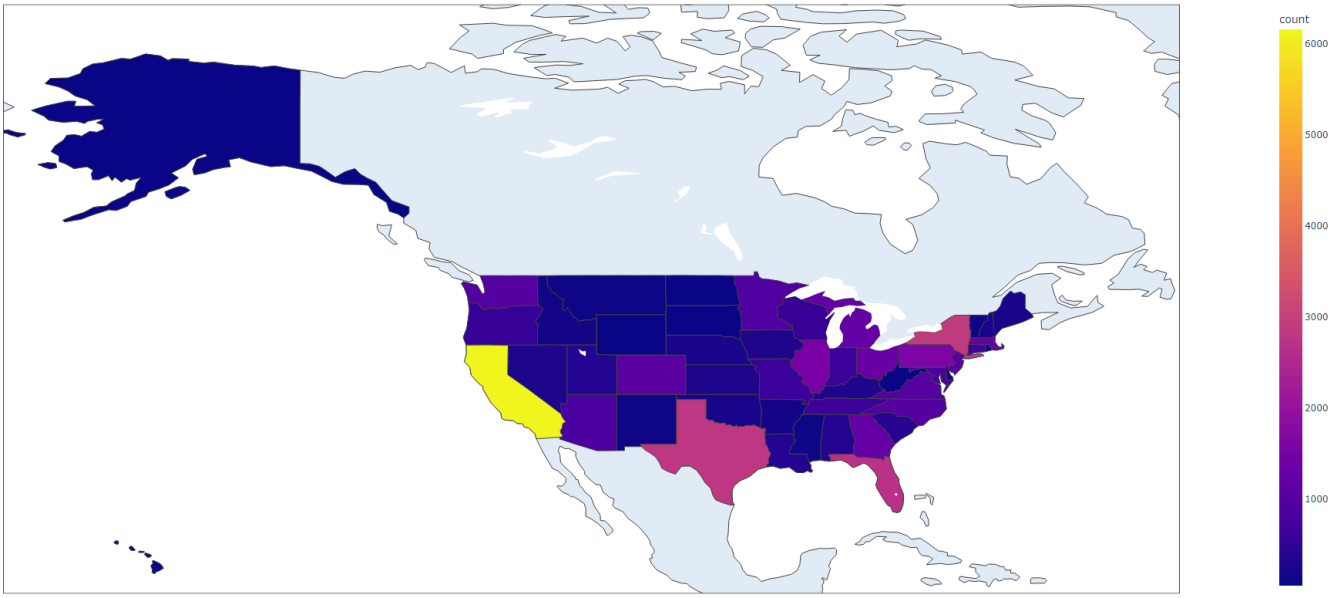


Map of Company locations by main latitude and longitude, colored according to the four revenue-based

labels. A high density is visible in the EU and the USA, with smaller clusters around large cities.

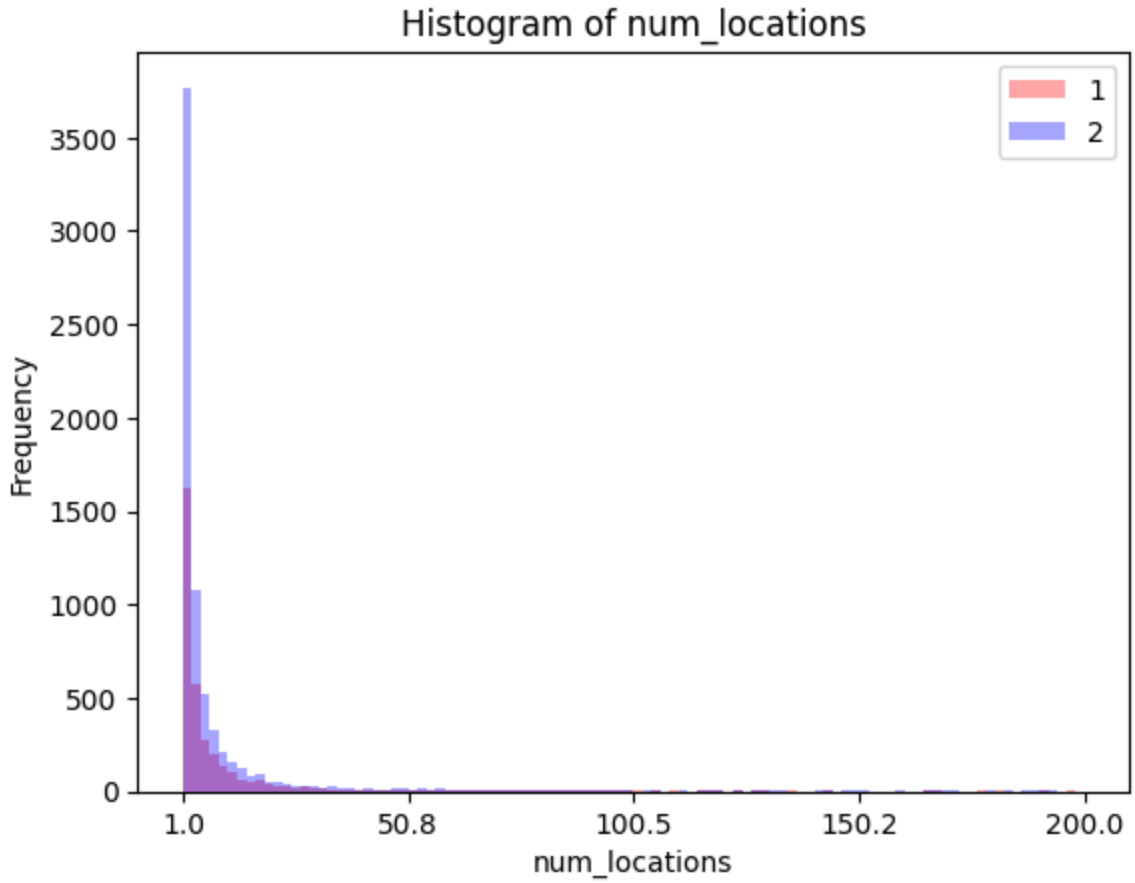


Map of countries by number of companies in the dataset, per their main_country. The United States is certainly the country with the most companies in this dataset.



States in the USA and their number of companies. California, Texas, New York and Florida account for a large number of companies.

2.2.4 *Number of locations*

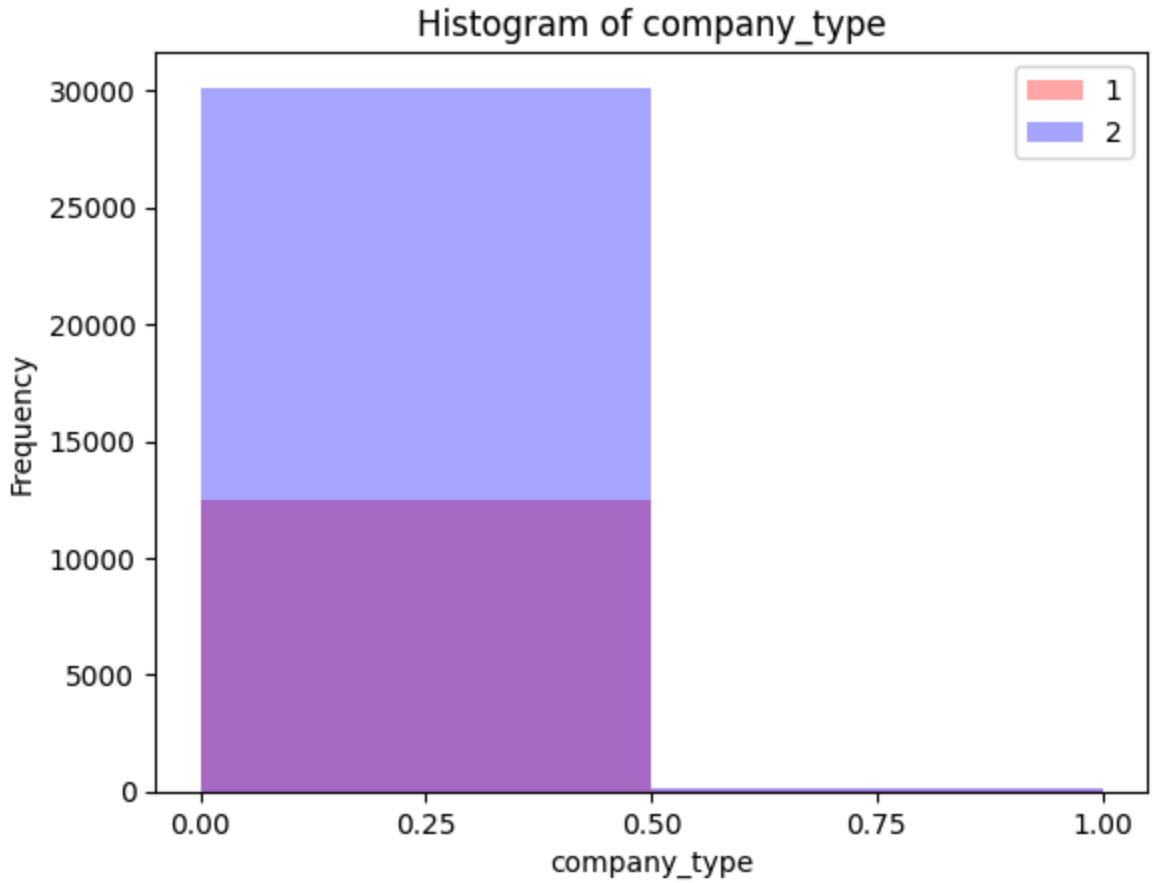



```
Statistics for Label 1:
Count: 12554
Mean: 5.052414
Standard Deviation: 56.347005
Minimum: 1.000000
25th Percentile: 1.000000
Median: 2.000000
75th Percentile: 3.000000
Maximum: 5327.000000
```

```
Statistics for Label 2:
Count: 30274
Mean: 3.805543
Standard Deviation: 25.518418
Minimum: 1.000000
25th Percentile: 1.000000
Median: 1.000000
75th Percentile: 2.000000
Maximum: 2500.000000
```

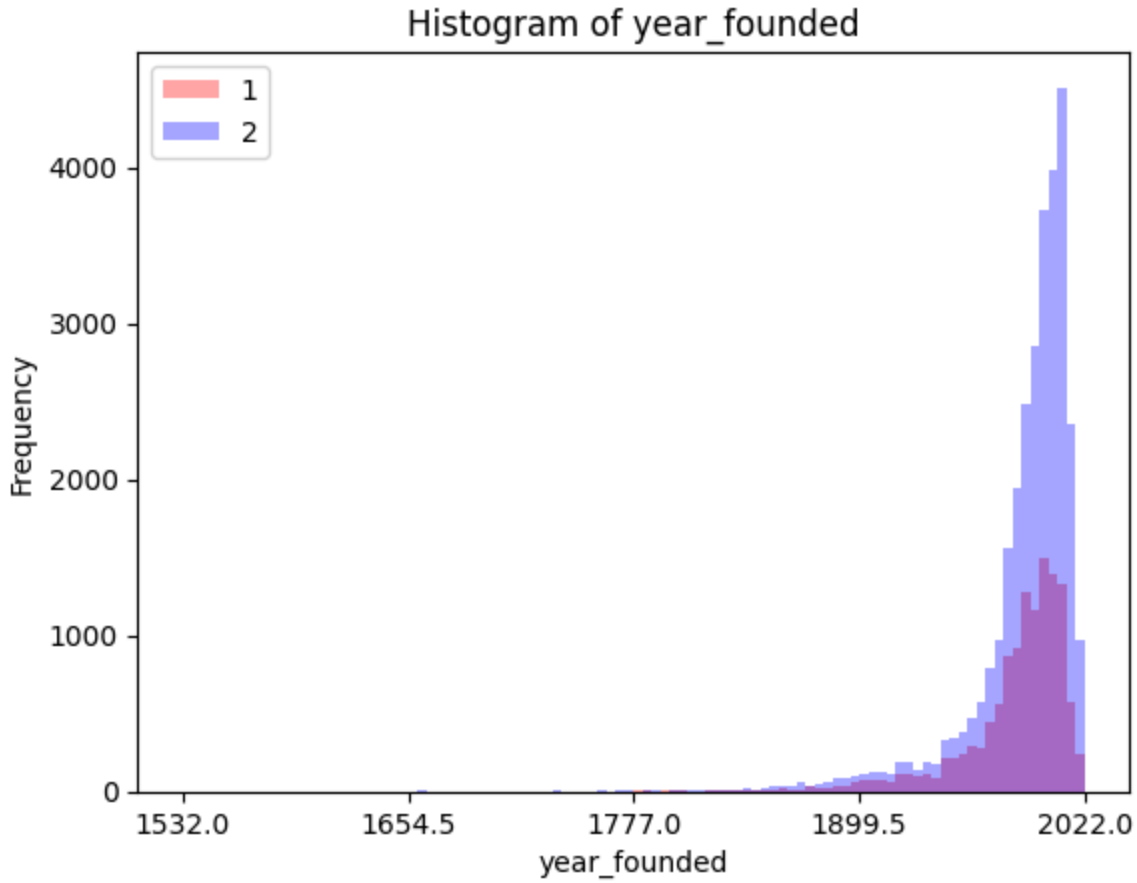
As can be seen from the non-normalized data above, the vast majority of companies have a small number of locations, only very few of them going over 10 locations. There is no significant difference between the two labels in this regard.

2.2.5 *Type*



The companies from the dataset are mostly private.

2.2.6 *Year founded*

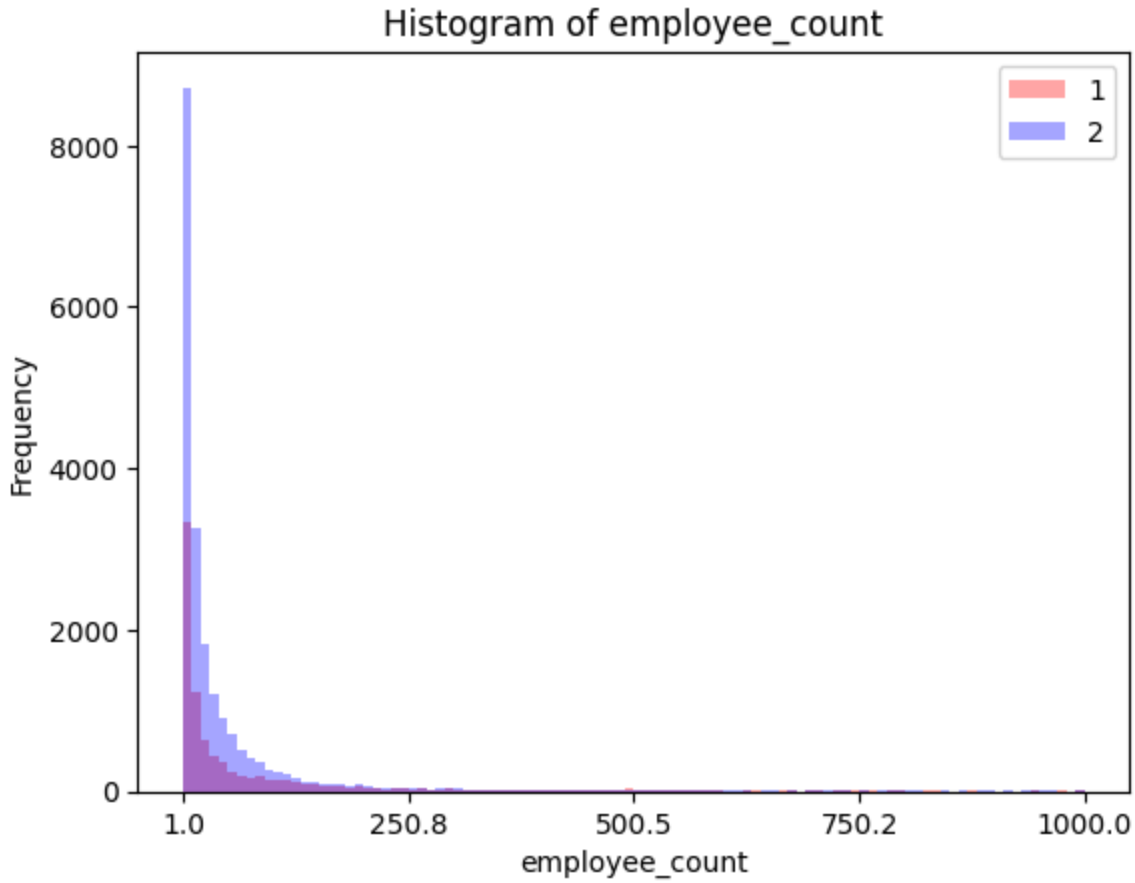



```
Statistics for Label 1:
Count: 12554
Mean: 1984.506930
Standard Deviation: 30.325773
Minimum: 1704.000000
25th Percentile: 1976.000000
Median: 1992.000000
75th Percentile: 2004.000000
Maximum: 2022.000000
```

```
Statistics for Label 2:
Count: 30274
Mean: 1989.775088
Standard Deviation: 29.633310
Minimum: 1532.000000
25th Percentile: 1983.000000
Median: 1998.000000
75th Percentile: 2008.000000
Maximum: 2022.000000
```

Most companies are relatively new, founded close to the 2000s. This is true for both labels.

2.2.7 *Number of employees*



Statistics for Label 1:

Count: 12554

Mean: 145.441533

Standard Deviation: 3262.846814

Minimum: 1.000000

25th Percentile: 3.000000

Median: 9.000000

75th Percentile: 41.000000

Maximum: 250000.000000

Statistics for Label 2:

Count: 30274

Mean: 95.551265

Standard Deviation: 1607.009321

Minimum: 1.000000

25th Percentile: 3.000000

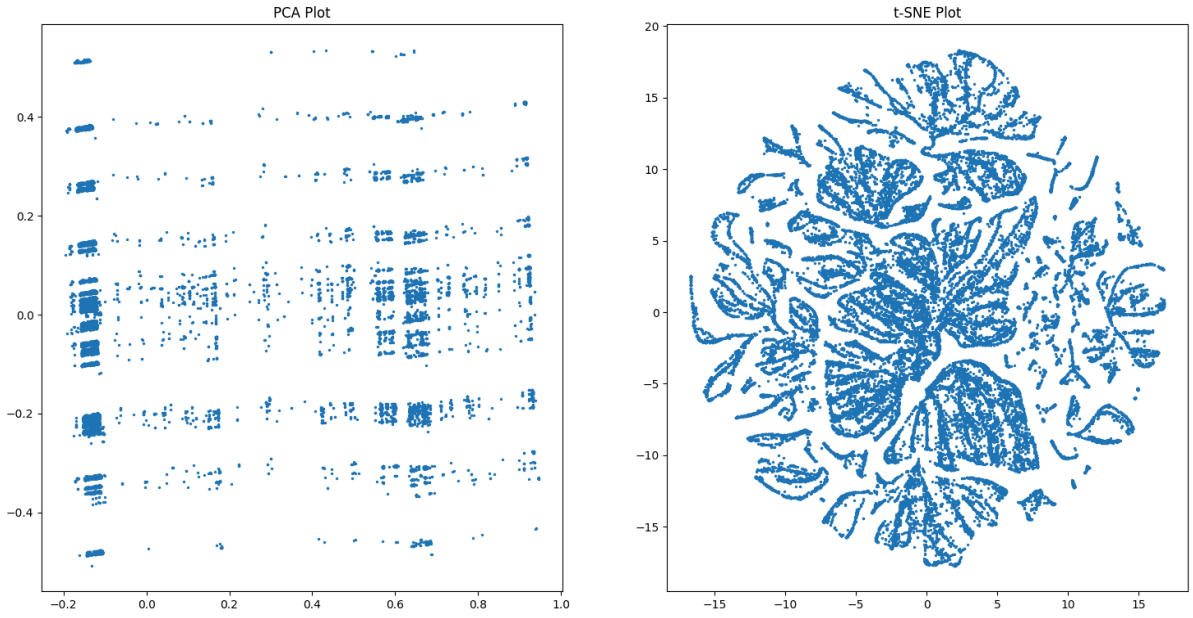
Median: 7.000000

75th Percentile: 28.000000

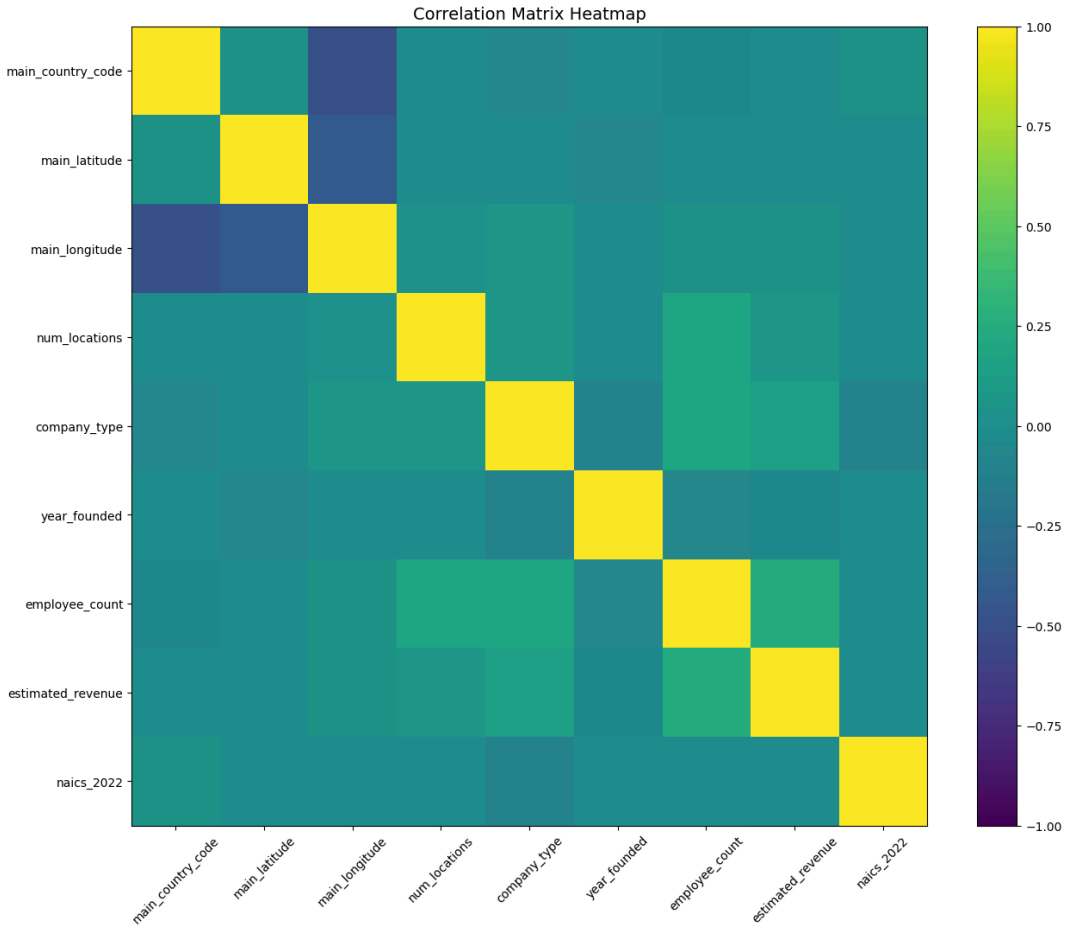
Maximum: 142931.0000

Most companies have very few employees, less than 10. Both labels exhibit a similar distribution.

2.2.8 *Dimensionality reduction: All features projected into 2 dimensions*



2.2.9 *Feature correlation and relative importance*



Coordinates are very important for revenue prediction. Also, company type and estimated revenue are very relevant features.

3. CHOSEN MODELS

3.1 Unsupervised Models: Clustering

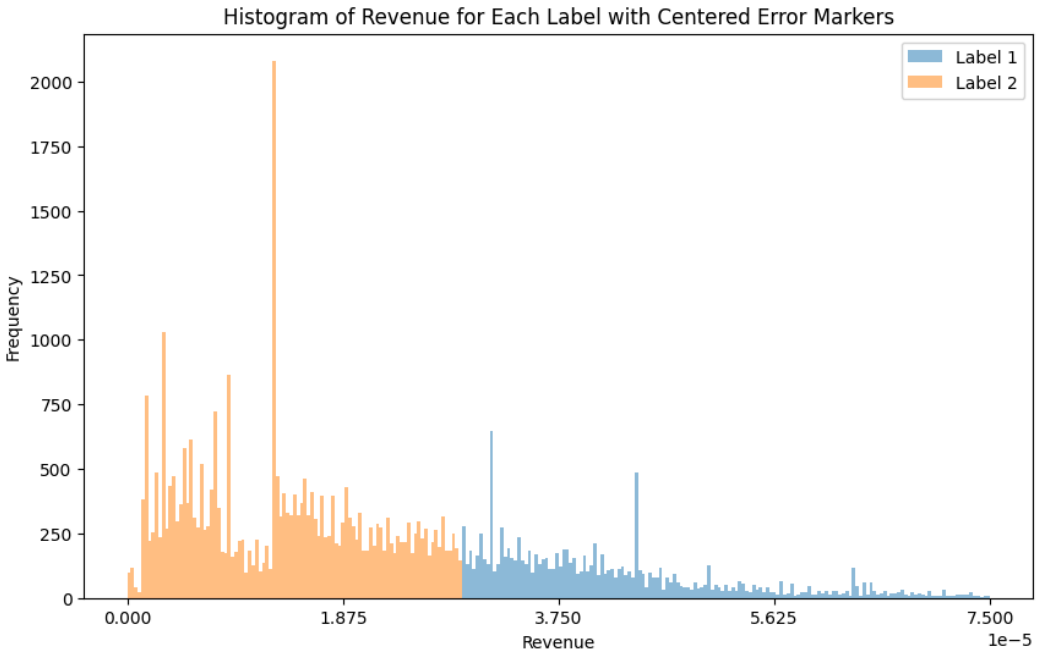
3.1.1 K-means

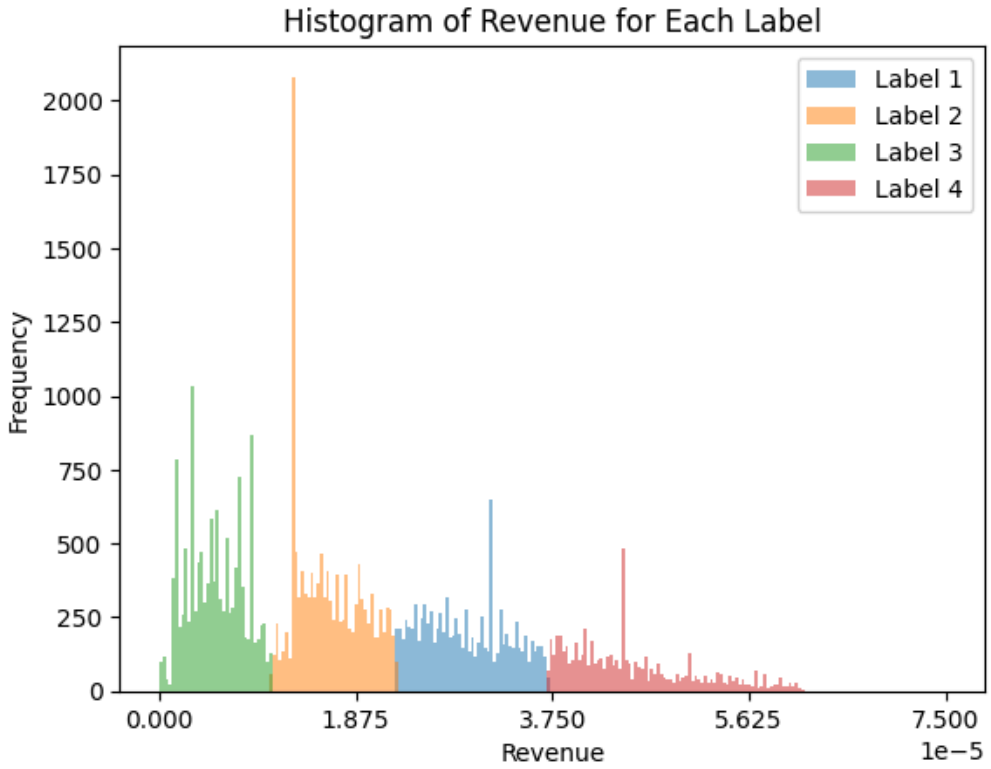
Clustering or Cluster Analysis is the task of grouping a set of objects such that objects in the same group (cluster) are more similar to each other than to objects in other groups.

K-means is a centroid-based clustering method, meaning that each cluster is represented by a prototype object (a center). This method partitions data points into a fixed number of clusters.

This unsupervised learning method has been used to obtain different classes of revenue, such that all data points could be then classified in one of the classes, and based on these labels to make regression for each class of revenue at a time. For the clusterization, the only feature used was the revenue.

The approach was to use the K-means model with GridSearchCV from scikit-learn, to find the number of clusters that maximized the silhouette score. Since there was a need for the clusters obtained to contain a viable number of samples, such that the classification models be helpful, we decided to use thresholds for the lowest amount of samples that a cluster may contain in order to be used for labeling. We ended up with two variants: two and four significant (after the elimination of the very small ones) clusters. The silhouette score for each of the variants is: 0.618 for the two cluster option and 0.590 for the four cluster option.



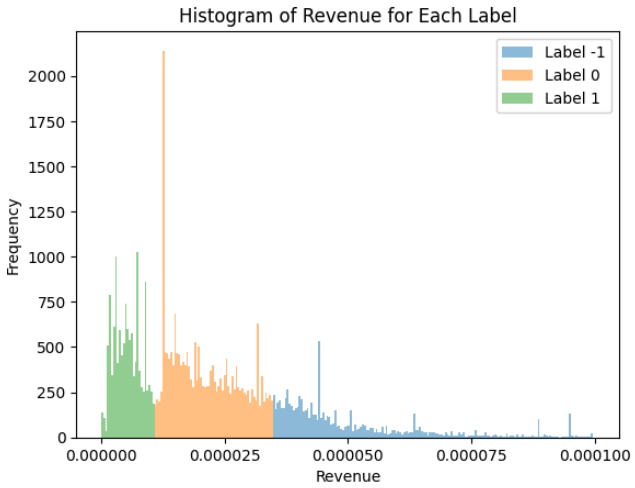


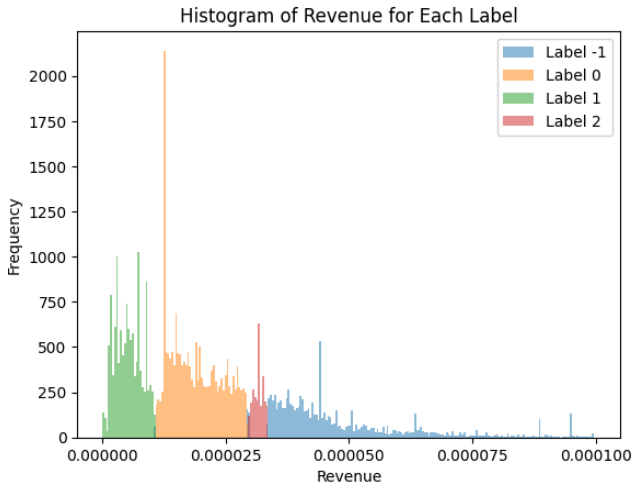
3.1.2 *DBSCAN*

Density-Based Spatial Clustering of Applications with Noise, or shorter DBSCAN, is a clustering technique that finds some main clusters with high density and expands them to encapsulate other data points until all samples are verified. It is very efficient when dealing with noisy data. The main parameters are *eps* and *min_samples*, which represent the maximum distance between two samples to be grouped in the same cluster, and the minimum samples for a group of points to be considered a cluster, respectively. The distance is euclidean by default, but other distances can be used.

Compared to K-means, DBSCAN does not take the number of clusters as a parameter. Our method was to find some values for *eps* and *min_samples* that give balanced clusters. Because the distances between data points is very small, we found that *eps* should be in the interval [1e-6, 1e-7], otherwise the algorithm only detects one cluster with the majority of samples, and a very small one plus the noisy points (e.g. 1e-3). Next we observed that *min_samples* should be in the interval [1100, 1400] to obtain two or three main clusters plus the noise. If this parameter is lower (e.g for 10, 50, 100, 500) we obtain only one cluster plus anomalies (from 3 to 50 different classes of anomalies) and noise.

We decided to go for two sets of parameters for (*eps, min_samples*), one with (1e-6, 1100) and one with (1e-6, 1350). For these two sets we obtain two and three clusters plus the noisy points (label -1 is the noise).





It can be observed that the noisy data points (label -1) can be considered a cluster in this case. They are labeled as noisy because the minimum number of samples for a cluster is set very high and the distance is set very low, therefore samples above a certain threshold of revenue are considered noisy. In the classification part we can consider that "cluster" as a class among the detected ones.

The number of samples for each cluster is:

```
Label -1 count: 11348
Label 0 count: 21486
Label 1 count: 12507
```

```
Label -1 count: 12432
Label 0 count: 17942
Label 1 count: 12449
Label 2 count: 2518
```

And the silhouette score for each set of parameters is:

```
Silhouette score:  0.194
```

```
Silhouette score:  0.111
```

3.2 Supervised Models: Classification

3.2.1 Data Preparation

The aim of these models was to predict the classes for the data after applying k-means clusterization.

Firstly, the 'label' feature from k-means was added to train and validation sets by using a DataFrame because the labels were in different files, where data had only the id, estimated_revenue and label features. We decided to drop the outliers (data which was not a part of any cluster) after seeing that they did not represent a significant part of the data. Only 2000-4000 samples in the training data out of ~50k had no label.

Some features consisted in lists that had text encoded in numerical data. We dropped some of those columns which had no effect on the performance of the model, features such as "long_description" or "business_tags". Features such as "ibc_insurance", "isic_v4", "nace_rev2" and "ncci_codes_28_1"

were kept because their lists were of small length and their data was relevant for the different models.

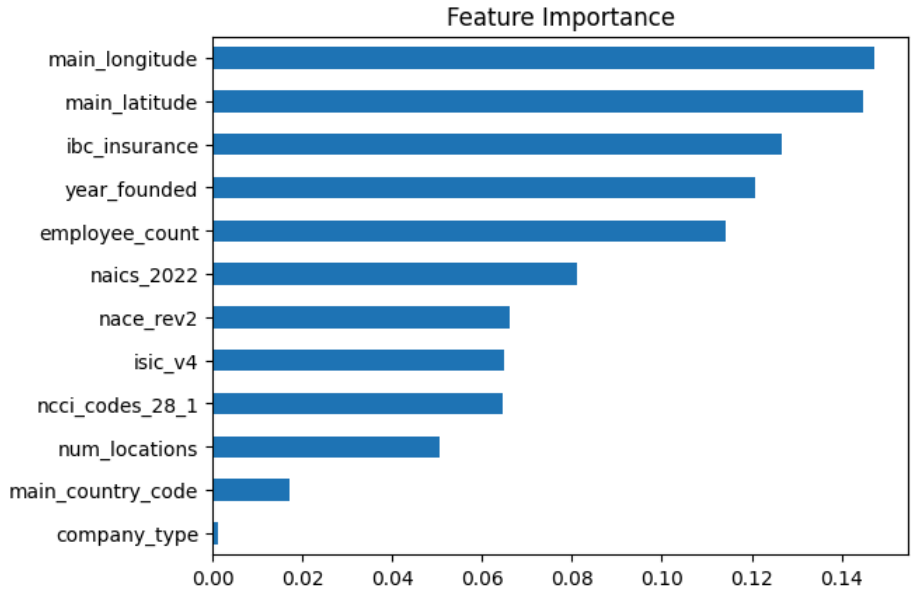
3.2.2 *Random Forest*

About random forest

A decision tree is a structure that splits data into categories by making use of its features. Apart from the leaves, each node represents a splitting criteria and the branches are regarded as paths which mark the next course of action for splitting or, eventually, the outcome. Decision trees, especially the ones with a significant height value, have a tendency to overfit. By adding up a lot of those decision trees, random forest solves this problem.

Random forest is a supervised learning algorithm that combines the predictions of multiple decision trees that can be used for both regression and classification tasks. The output of the algorithm applied on a classification task represents the class selected by the majority of the trees. For regression, the output is the average values between the results of all the trees.

The image below shows feature importance for 2-label clusterization.



As expected, the coordinates matter a lot because a lot of the companies are from the US. Also, ‘year_founded’ and ‘employee_count’ usually tells us something about the revenue of the company. ‘ibc_insurance’ seems to be correlated to the number of insurances for a company and the price that they

might pay for it (the bigger those numbers the more revenue the company has). We might get more insight for this by denormalizing the values.

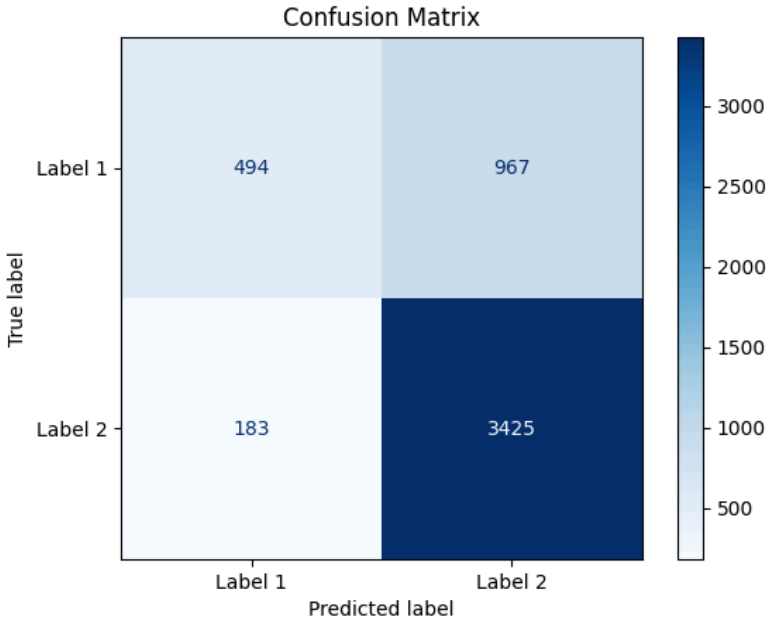
Results

We used RandomForestClassifier with 100 estimators from sklearn.ensemble. We also tested the model with 256 trees, which did not improve the model's accuracy. The relevant metrics provided are accuracy, precision,recall,f1 score(from classification_report provided by sklearn.metrics) and the confusion matrix, from the same library.

The results for the 2-label classification on the validation set are:

Accuracy: 0.77 Macro F1 score: 0.66

Label	Precision	Recall	F1 score
1.0	0.73	0.34	0.46
2.0	0.78	0.95	0.86

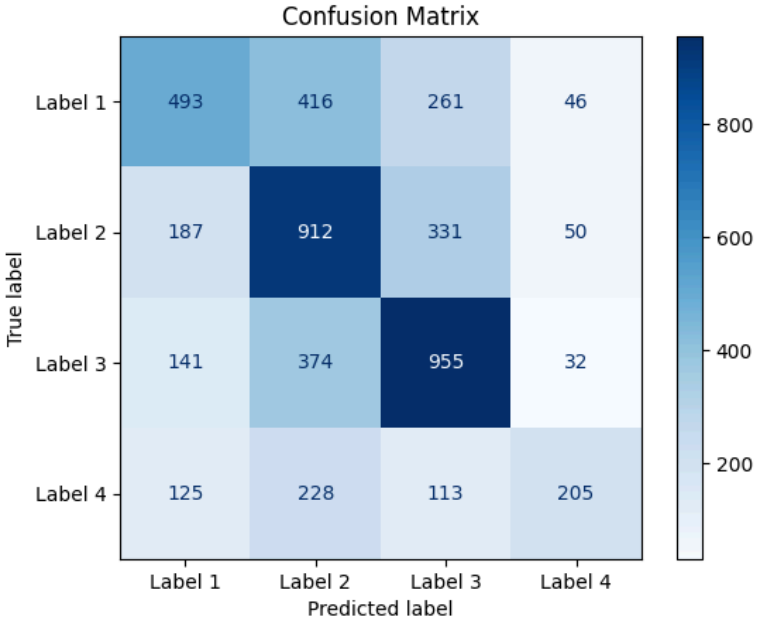


It can be observed that the model tends to focus better on the second class. Also, the first class is underrepresented and this might be a reason for the fact that Label 1 is usually predicted as Label 2, which can also be seen by the low recall value.

The results for the 4-label classification on the validation set are:

Accuracy: 0.53 Macro F1 score: 0.5

Label	Precision	Recall	F1 score
1.0	0.52	0.41	0.46
2.0	0.47	0.62	0.53
3.0	0.58	0.64	0.60
4.0	0.62	0.31	0.41



The model usually gets a lot of the data from classes 2 and 3 right. Label 1 is sometimes misinterpreted as Label 2 and Label 4 is too difficult for the algorithm to predict, which can also be seen from the fact that it has the lowest recall.

3.2.3 *Support Vector Machine (SVM)*

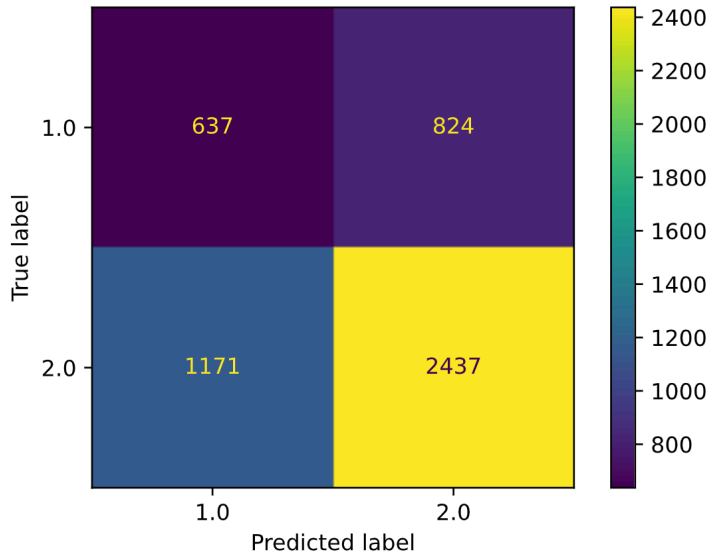
Support Vector Machine (SVM) is a supervised machine learning algorithm designed for classification tasks, aiming to find the optimal hyperplane that maximally separates different classes in a high-dimensional feature space.

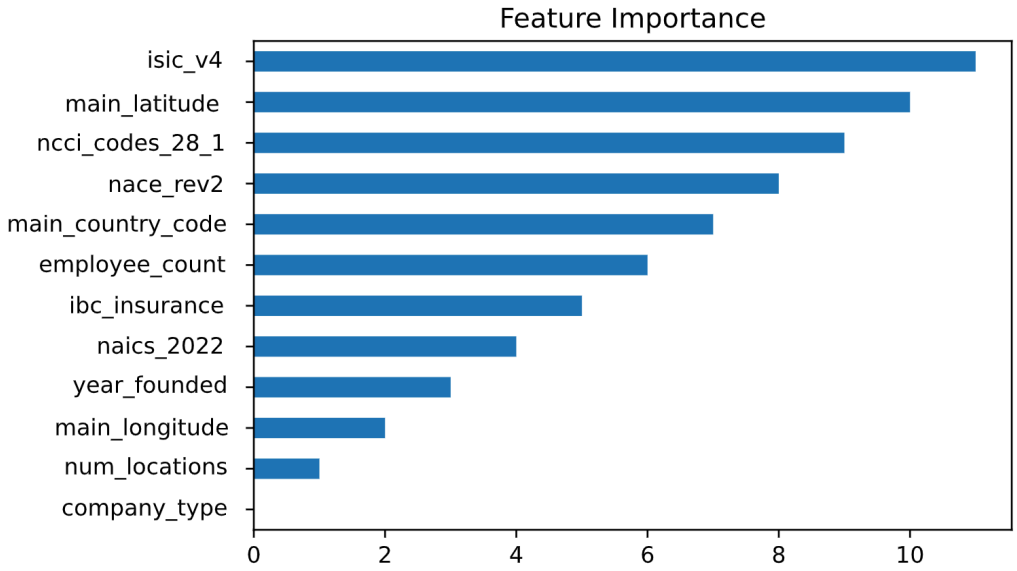
Two different *scikit-learn* models were used to perform classification using SVM.

First, *LinearSVC* was used, as it is better optimized for large amounts of data. However, its performance was not satisfactory, even after using Grid Search to find an optimal value of the *C* parameter. Accordingly, we later used the *SVC* model from the same library. Its execution time being indeed more limiting, we restricted the grid search to finding the best kernel, rather than modifying the value of *C*. Thus, we found that the (default) *rbf* kernel, along with a *gamma* value set to *auto* led to the best performance for SVM, shown in the table below. It bears mentioning that for both versions, using a *class_weight = balanced* greatly decreased the tendency to over-predict the second, more numerous class, providing a better F1 macro score almost every time.

Class	Precision	Recall	F1 score
1.0	0.35	0.44	0.39
2.0	0.75	0.68	0.71
Accuracy			0.61
Macro Avg	0.55	0.56	0.55
Weighted Avg	0.63	0.61	0.62

For better visualization, the figures below show the associated confusion matrix, and the relative feature importance, respectively. While normally a non-linear model such as a SVM with the rbf kernel cannot show feature importance using its coefficients, we calculated this using permutation importance.





For SVM, the Industrial Classification of All Economic Activities index appears to be most important, along with the main_latitude. Conversely, company_type and the number of locations are not quite relied upon by the model.

3.2.4 XGBoost

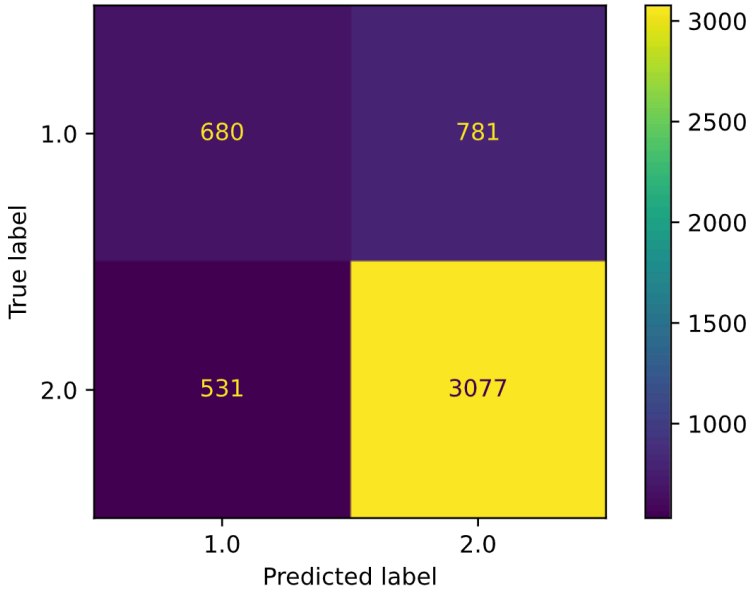
XGBoost, an abbreviation for eXtreme Gradient Boosting, is an efficient and scalable machine learning algorithm that belongs to the ensemble learning family, specifically designed for classification and regression tasks by sequentially combining the outputs of multiple weak learners to create a robust and accurate predictive model. Out of the gradient boosting methods, XGBoost is certainly among the most used and arguably among the most effective.

For this task, we used the *xgboost* library, more specifically the *XGBClassifier*. Owing to its large number of parameters, a 'complete' Grid Search would be very time consuming. Since the task at hand involves a binary classification, setting the *objective='binary:logistic'* is a necessity. The parameters used in the Grid Search are the learning rate, also called *eta*, as well as *scale_pos_weight*, since the

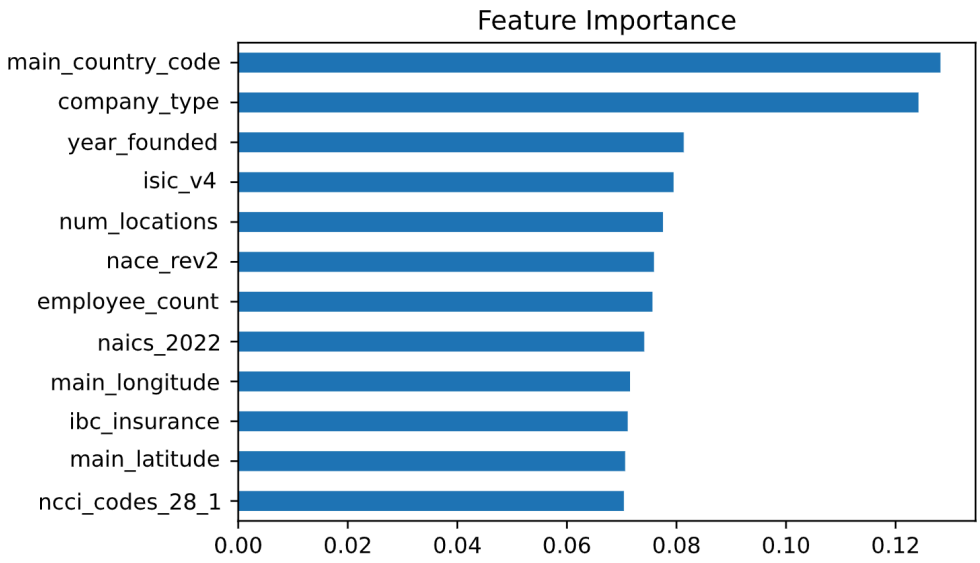
dataset is rather unbalanced, as previously mentioned.

In the end, we obtained the best performance for *scale_pos_weight = 0.7*, and *eta=0.8*. Interestingly, the addition of more estimators becomes futile after around 5000, when no performance increase occurs anymore. The table below shows the classification report for these parameter choices, while the figures present the confusion matrix and the feature importance.

Class	Precision	Recall	F1 score
1.0	0.56	0.47	0.51
2.0	0.80	0.85	0.82
Accuracy			0.74
Macro Avg	0.68	0.66	0.67
Weighted Avg	0.73	0.74	0.73



The report and confusion matrix shows that, as the other models, XGBoost also struggles with the first class, but performs comparatively better, overall.



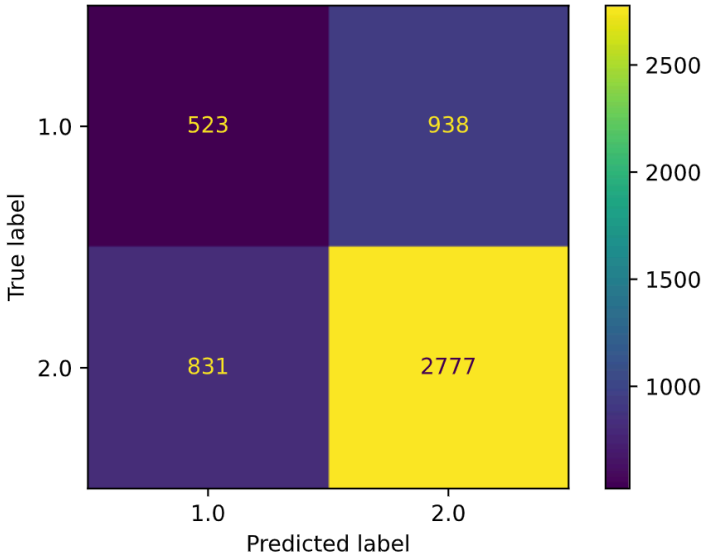
It can be seen that the main_country_code and company_type features seem to play a large role for XGBoost in determining the correct label.

3.2.5 K Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple supervised machine learning algorithm used for classification tasks. In KNN, the classification of a new data point is determined by the class of its k-nearest neighbors in the feature space. The algorithm relies on the assumption that similar data points share similar class labels. To classify a new instance, KNN calculates the distance between the point and its neighbors using a chosen distance metric, commonly Euclidean distance.

The table below was obtained for a n_neighbors value of 6. It can be seen that KNN is not well suited for this type of classification task due to the nature of the data. Additionally, feature importance cannot be calculated for this model.

Class	Precision	Recall	F1 score
1.0	0.39	0.36	0.37
2.0	0.75	0.77	0.76
Accuracy			0.65
Macro Avg	0.57	0.56	0.57
Weighted Avg	0.64	0.65	0.65

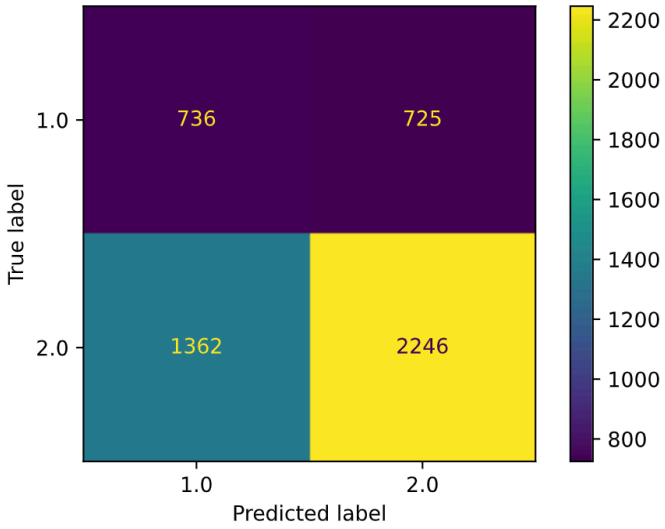


3.2.6 *Ridge Classifier*

This scikit-learn model is a variant of Ridge regression. The classifier first converts binary targets to {-1, 1} and then treats the problem as a regression task, optimizing the same objective as in regression. The predicted class corresponds to the sign of the regressor's prediction.

For these results, the class_weight = 'balanced' parameter was used. The value of alpha does not seem to impact the accuracy or F1 in a significant manner, so the default value was left. The Ridge Classifier tends to predict the label 1 much more aggressively than the other models, but in the process, it also mislabels the actual label 1 samples, and its performance is not ideal.

Class	Precision	Recall	F1 score
1.0	0.35	0.50	0.41
2.0	0.76	0.62	0.68
Accuracy			0.59
Macro Avg	0.55	0.56	0.55
Weighted Avg	0.64	0.59	0.61

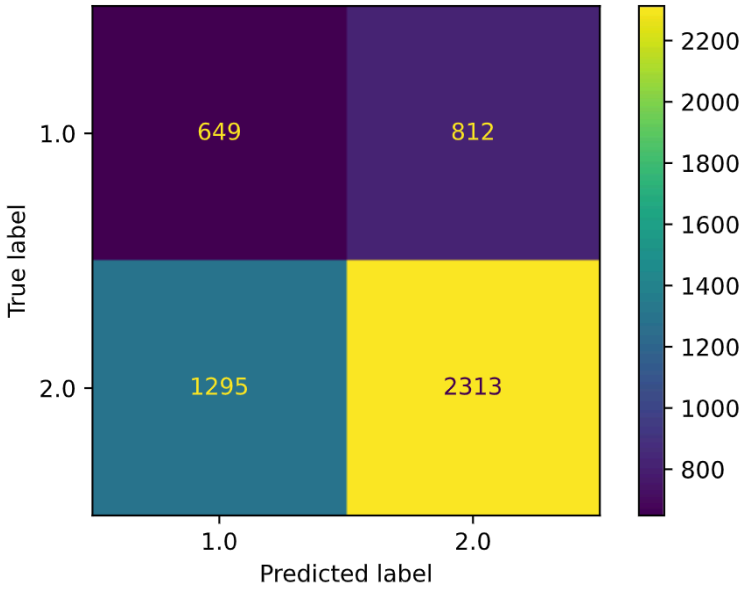


3.2.7 *Complement Naive Bayes (CNB)*

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the model's weights.

Since CNB is specifically tailored for imbalance data, this approach proves fruitful and the results are comparable with the other models. By contrast, Gaussian, Bernoulli and Multinomial Naive Bayes all only predict the second label.

Class	Precision	Recall	F1 score
1.0	0.33	0.44	0.38
2.0	0.74	0.64	0.69
Accuracy			0.58
Macro Avg	0.54	0.54	0.53
Weighted Avg	0.62	0.58	0.60



3.2.8 *Multi-layer Perceptron (MLP) for Classification*

Multi-layer Perceptron classifier (MLP) is an artificial neural network used for classification tasks. It is a type of feedforward neural network, meaning that information flows in one direction, from the input layer through the hidden layers to the output layer.

When using this model with the same data processing as before it would overfit on the second label, because of this we flattened the data.

To perform this model we used the *MLPClassifier* model from *sklearn.neural_network*. For parameter optimization, we used Grid Search from *sklearn.model_selection*, and the best parameters for the data set used were: for the activation function '*tanh*', the solver for weight optimization '*adam*' (which works best since we used a larger dataset), early stopping set to false, three hidden layers and the default values for *alpha* and *learning_rate_init*.

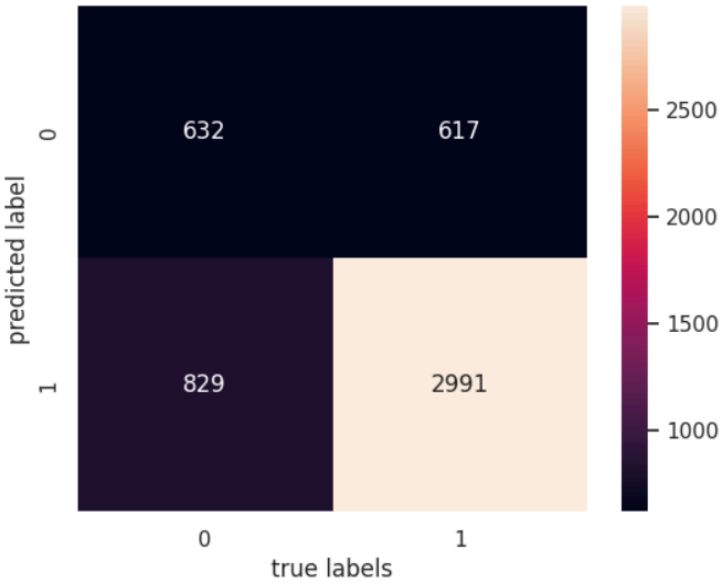
Accuracy on Validation Set: 0.714

Balanced Classification Score: 0.63

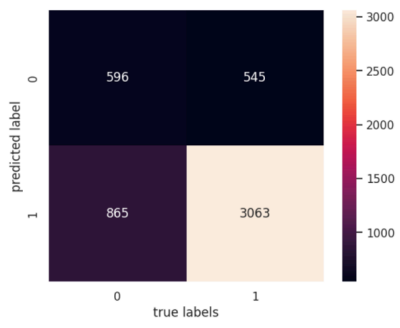
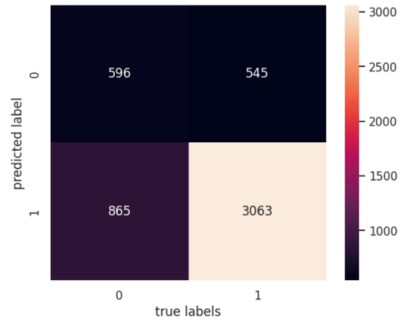
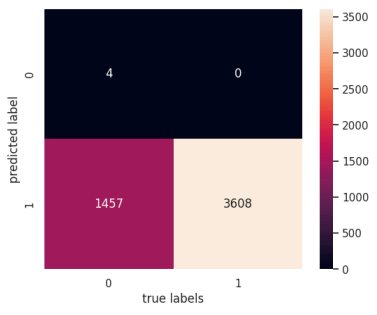
	precision	recall	f1-score	support
1.0	0.51	0.43	0.47	1461
2.0	0.78	0.83	0.81	3608
accuracy			0.71	5069
macro avg	0.64	0.63	0.64	5069

	precision	recall	f1-score	support
weighted avg	0.70	0.71	0.71	5069

To enhance the visualization of the data, we present the following graphical depiction illustrating the outcomes of the classification using a confusion matrix:



Other models we have tried have yielded the following results:

Model with 2 hidden layers:	Model with 4 hidden layers:	Model with early_stopping=True:
Accuracy on Validation Set: 0.721	Accuracy on Validation Set: 0.717	Accuracy on Validation Set: 0.71
Balanced Classification Score: 0.628	Balanced Classification Score: 0.622	Balanced Classification Score: 0.5
		

We can observe from the previous results (on the model with 3 layers which we have chosen as the most accurate, based on the balanced accuracy) and the first two from the table above that the results

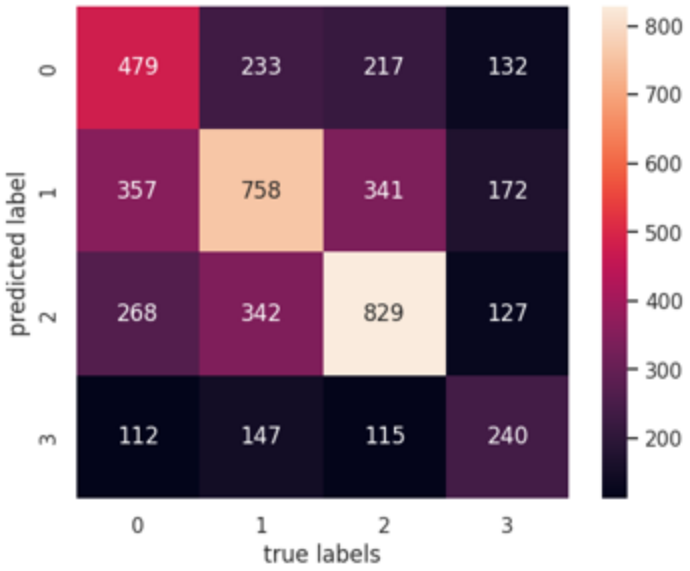
are similar. We chose the most accurate model based on the balanced accuracy (we have done this because the accuracy on the validation set is around 0.7 even if the model overfits on the second class like the last result from the table).

The results for the 4-label classification on the validation set are:

Accuracy on Validation Set: 0.47

Balanced Classification Score: 0.45

	precision	recall	f1-score	support
1.0	0.45	0.39	0.42	1216
2.0	0.47	0.51	0.49	1480
3.0	0.53	0.55	0.54	1502
4.0	0.39	0.36	0.37	671
accuracy			0.47	4869
macro avg	0.46	0.45	0.46	4869
weighted avg	0.47	0.47	0.47	4869

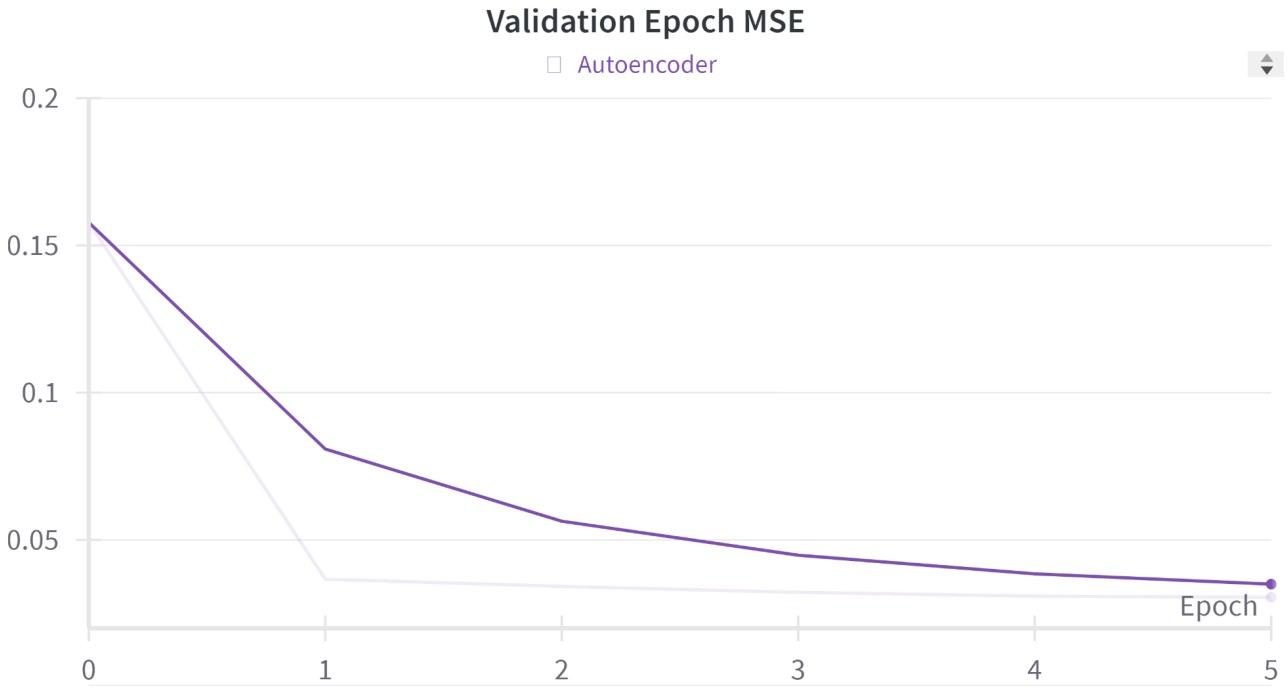


3.3 Supervised Models: Regression

3.3.1 Multi-layer Perceptron (MLP) for Regression

This method focused on a neural network pipeline. Due to the high capabilities of neural networks to handle high dimensional data and use them in a nonlinear manner, I used all selected features, including the high dimensional transformer-encoded text, without reducing it in any other way.

In order to better prepare the features for a final one number prediction, I firstly employed an autoencoder with a simple reconstruction objective - MSE. I also tried out a VQ-VAE that makes use of a finite learned codebook that truncates the bottleneck values in order to make the infinity space more manageable for reconstruction. This approach did not gain better results in this case. Furthermore, using 4 downscale layers - with a downscale factor of 2 - 3 layers in the bottleneck and 4 upscale layers - same factor of 2 - gives the best results for the autoencoder objective.

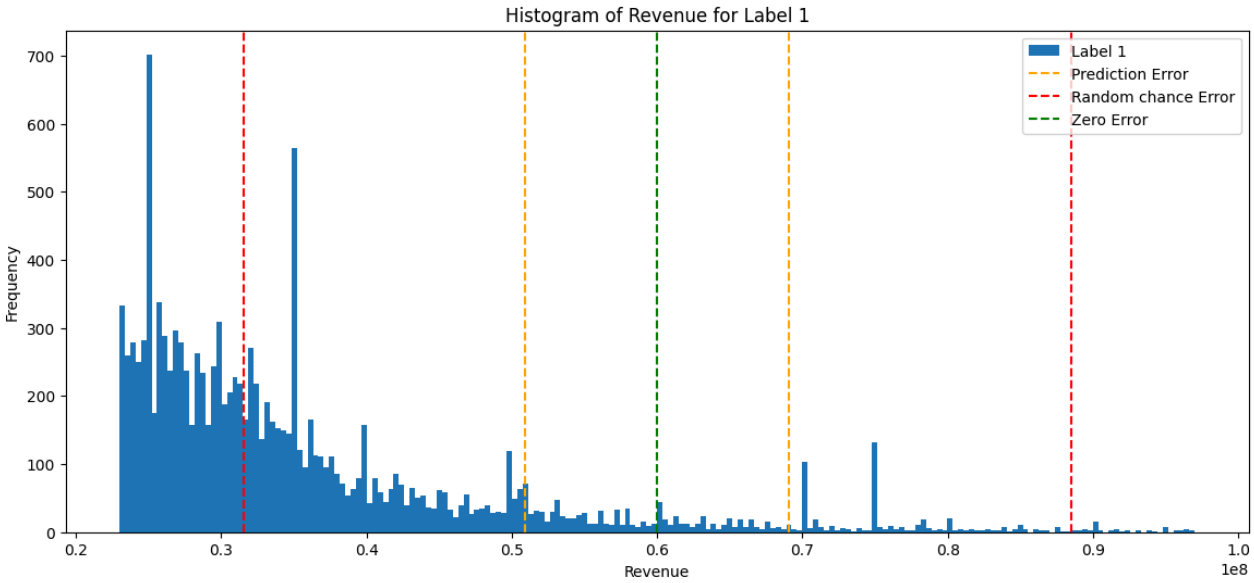


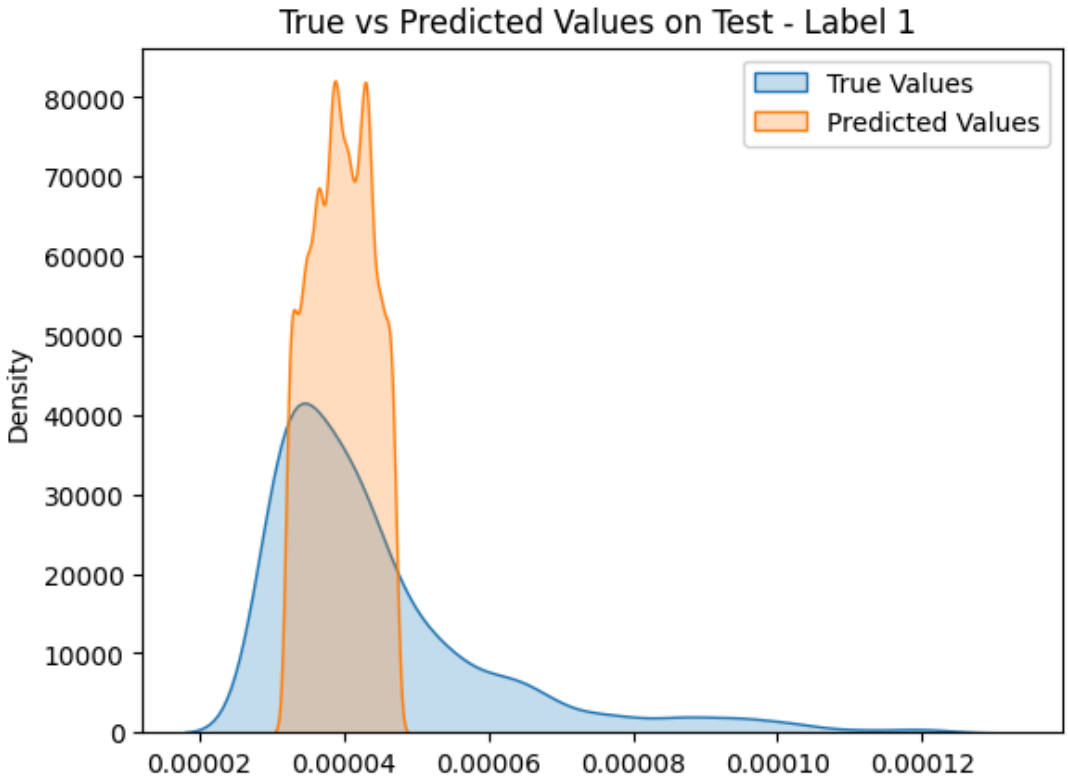
The best model gained a MSE of 0.03 on validation and had a bottleneck dimension of 256.

Furthermore, I used the trained encoder and added up to 3 new layers in order to downscale up to one dimension - the dimension of revenue value. I fine-tuned all layers on the new objective that involve only revenue prediction, and not a whole feature space reconstruction. We used MSE as loss and *denormalized Euclidean distance* as a metric.

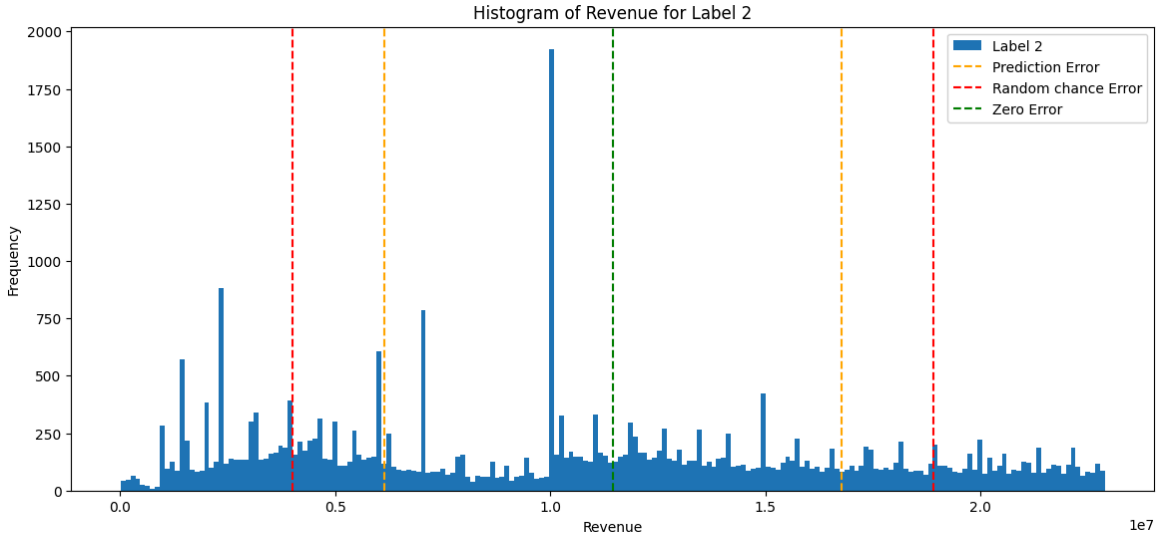
I fine-tuned two models, one only on the first label class of the data and on for only the second class. The models with best metrics on validation were chosen. I got the following **metrics** on test data:

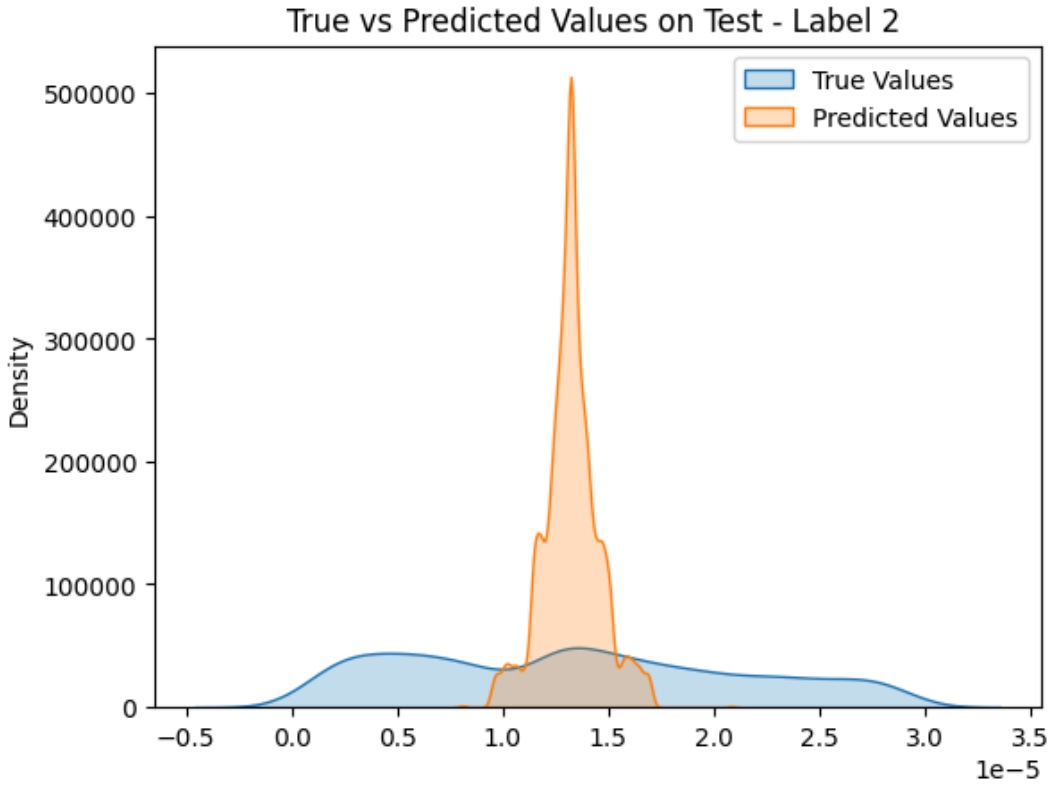
- Label 1: ***9 052 716*** *[L1 Loss - denormalized]*


Histogram of Revenue for Label 1
Label 1
Prediction Error
Random chance Error
Zero Error
Frequency
Revenue
1e8


True vs Predicted Values on Test - Label 1
True Values
Predicted Values
Density

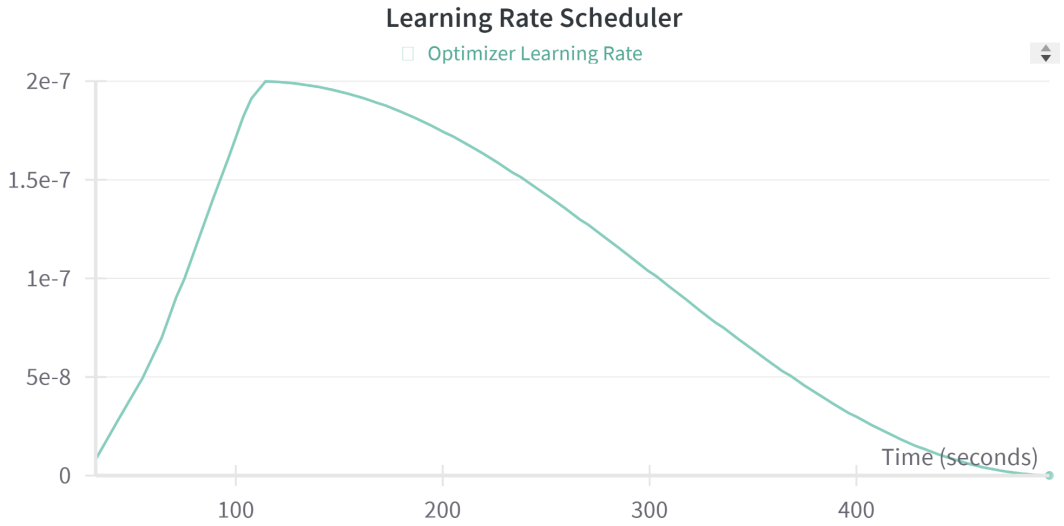
- Label 2: ***5 312 476*** *[L1 Loss - denormalized]*





The random chance was calculated by sampling randomly from the class range of revenue and calculating the average distance.

For all models, LayerNormalization, SiLU activation and a cosine learning rate scheduler were used.



Different values for learning rate and batch size were tested. It is worth noting that there is a big range of learning rates that did not work at all, resulting in either only zero prediction or in exploding gradients. This fact underlines the very difficult dataset, also because of the huge range of values.

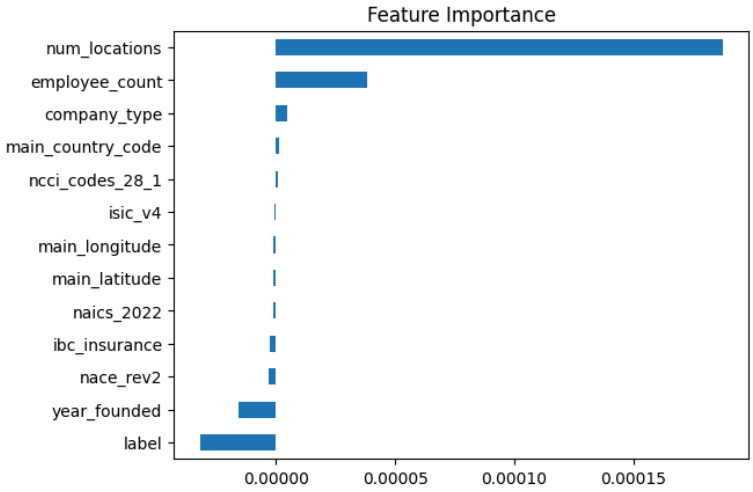
3.3.2 *Linear and Polynomial Regression*

Linear regression is a simple and widely used method for modeling the relationship between a dependent variable and one or more independent variables. The goal of linear regression is to find the best-fitting line that minimizes the sum of the squared differences between the observed and predicted values of the dependent variable.

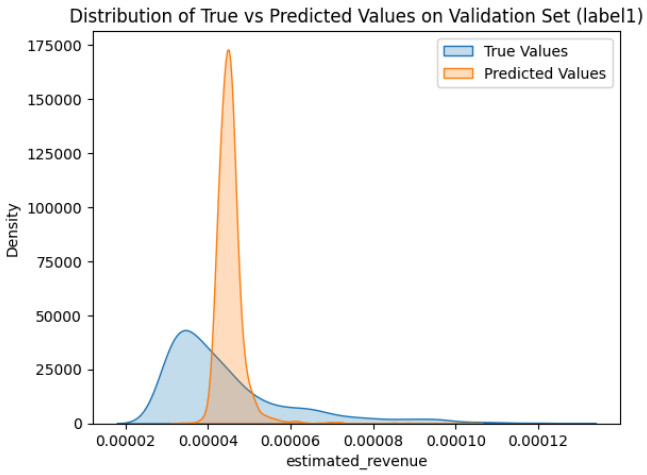
Polynomial regression is an extension of linear regression where the relationship between the variables is modeled as an nth-degree polynomial. Polynomial regression allows for more flexibility in capturing non-linear relationships in the data. By increasing the degree of the polynomial, the model can better fit curves and complex patterns.

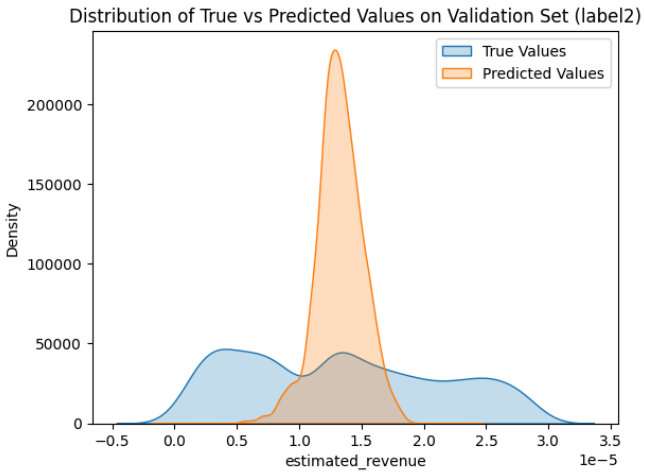
Our goal was to predict the estimated revenue of different companies, so we tried 4 different regression approaches that we will describe in the following paragraphs.

For this task we used all of the 13 features and the binary label clustering we generated earlier using K-means. This enabled us to train both Linear and Polynomial Regression models on the whole dataset as well as on individual classes and then compute their Mean Squared Error. For the Polynomial Regression we only used *grade=2* as bigger values lead to an exponential increase in the MSE.



We can observe that the results get slightly better when only trained on data labeled with “2” (which has double the samples of label 1) but that doesn’t necessarily mean it learns better.





Model	Mean Squared Eror
Linear Regression	1.189228*10^(-10)
Linear Regression (label 1)	2.563686*10^(-10)
Linear Regression (label 2)	6.196271*10^(-11)
Polynomial Regression	1.170916*10^(-10)
Polynomial Regression (label 1)	2.559355*10^(-10)
Polynomial Regression (label 2)	5.970276*10^(-11)

In the charts above, we can clearly see that the model tends to predict only a specific range of the estimated revenue.

4. RESULTS

4.1.1 Clustering Model Results

Based on the silhouette scores obtained with K-means and DBSCAN, which can be seen in the table below, for the classification task we decided to move forward with the labels resulting from K-means clustering, mainly the version with two labels.

Clustering Model	Silhouette Score
K-means two labels	0.618
K-means four labels	0.590
DBSCAN two labels	0.194
DBSCAN three labels	0.111

4.1.2 *Classification Model Results*

In the following table, we have added the version that maximizes Macro F1-score for each classification model, on the two labels classification task, based on the K-Means clustering. We completed the comparison table with the Precision and Recall scores obtained for each of the two classes.

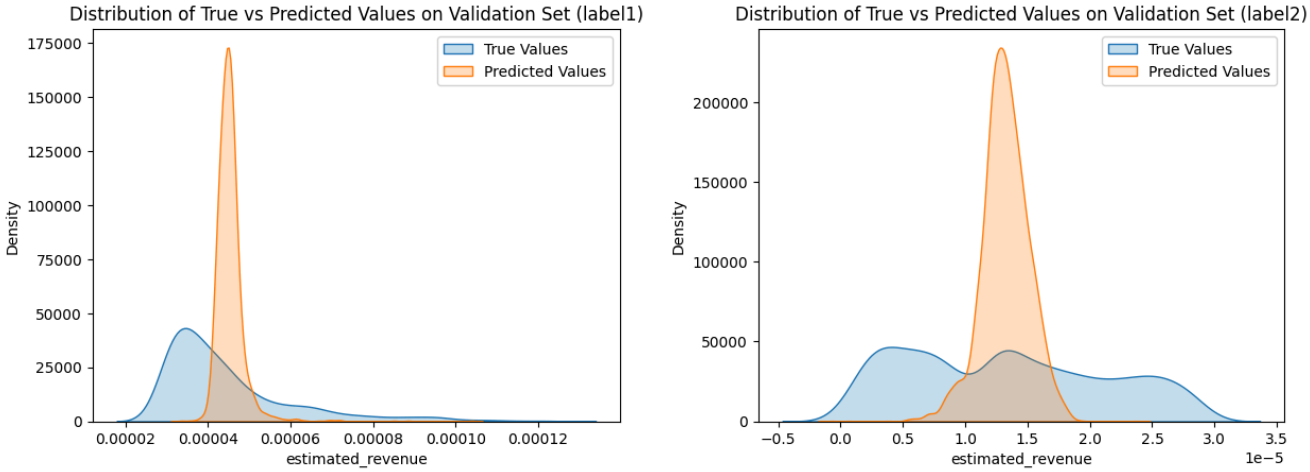
Classification Model	Macro F1 score	Precision class 1	Recall class 1	Precision class 2	Recall class 2
Random Forest	0.66	0.73	0.34	0.78	0.95
SVM	0.56	0.35	0.44	0.75	0.68
XGBoost	0.66	0.56	0.47	0.80	0.85
KNN	0.57	0.39	0.36	0.75	0.77
Ridge Classifier	0.55	0.35	0.5	0.76	0.62
CNB	0.53	0.33	0.44	0.74	0.64
MLP	0.64	0.51	0.43	0.78	0.83

Random Forest, XGBoost and MLP have the best performance out of the classification models, judging by Macro F1 score. Ridge Classifier, KNN, CNB and SVM have similar results amongst themselves.

4.1.3 Regression Model Results

MLP Regression

Linear regression



From the histograms above we can see that the MLP performs better on the first cluster - label 1 - where the range is smaller, better fitting the distribution. While, on the second cluster - label 2 - the MLP struggles a lot with the numerical instability given by a bigger range, and the Linear regression gives as so, better results.

5. FUTURE WORK

We will consider the option of combining the best outputs as part of our roadmap. One facile way to do that is to use fully connected layers in order to find optimal weights for each model, to perform a weighted average over their respective predictions.

6. CONCLUSION

We have seen in the Results section that the models chosen tend to perform better in certain situations, but overall, K-Means gave us the best clustering, Random Forest, XGBoost and MLP had the highest F1 score, for classification, and in terms of regression, the two models' performance is different, for each cluster.

After analyzing the features in this dataset, we noticed that most of them have a rather low relevance to the revenue. This low correlation might also suggest that the data could be misleading or noisy.

Furthermore, we found out that working with a very large range of numbers, in this case from $7000 up to $780,000,000,000, is very unstable for most models, resulting in poor results. One solution to counteract this issue seems to be to split this range into smaller dense clusters and also to remove anomalies.