Probabilistic Modeling Project Report

Abstract

The purpose of this project is to leverage probabilistic modeling techniques to classify data into distinct categories, as indicated by the target variable (Y). The data used for this project consists of multiple features, including categorical and numerical variables, which required preprocessing and feature engineering for effective modeling. This report details the methods, analyses, and outcomes of the project.

Keywords: Probabilistic Modeling, Market Segmentation, Market Classification.

Data Description

First, we have to understand the dataset we have to work on.

Data Overview:

The dataset comprises 8,523 rows and 11 columns, including the target variable `Y`. The features can be categorized as follows:

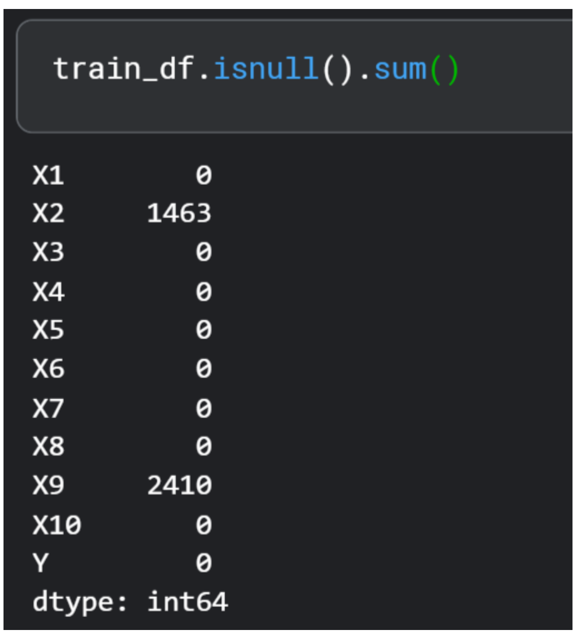
Numerical Features: X2, X4, X6, X8

Categorical Features: X1, X3, X5, X7, X9, X10

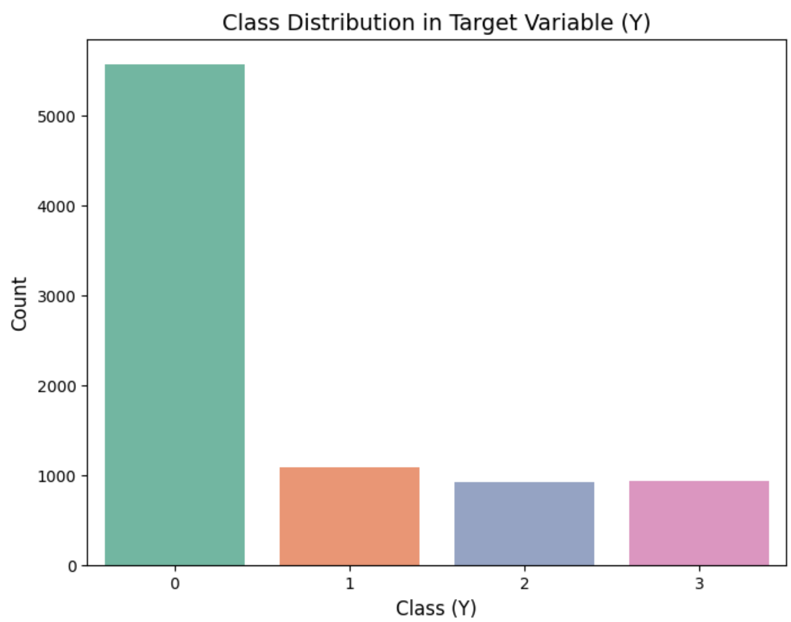
Target Variable: Y (4 Integer classes: 0, 1,2 3)

Initial Observations:

- Missing values were present in columns X2 and X9.
- Categorical columns required encoding to be used in probabilistic models.
- The target variable exhibited an imbalanced class distribution, as visualized using a bar plot.



```
train_df.isnull().sum()

X1        0
X2     1463
X3        0
X4        0
X5        0
X6        0
X7        0
X8        0
X9     2410
X10       0
Y         0
dtype: int64
```



Preprocessing Steps:

Categorical Data:

We tried three different methods for filling Nulls in X9:

- KNN Imputer.
- Replacing with the most frequent value which was 'Medium'.
- Accepting NaN as a fourth category encoded as "Unknown".

We manually encoded the three X9 columns using .map():

```
# We will manually encode so that we know which categ is which and have it be the same acros the 3 columns
mapping_X9 = {'Unknown': 0, 'Small': 1, 'Medium': 2, 'High': 3}
train_df['Med_X9'] = train_df['Med_X9'].map(mapping_X9)
train_df['unk_X9'] = train_df['unk_X9'].map(mapping_X9)
train_df['X9'] = train_df['X9'].map(mapping_X9)
train_df
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y	unk_X9	Med_X9	imputed_X9
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	2.0	Tier 1	0	2	2	2
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	2.0	Tier 3	2	2	2	2
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	2.0	Tier 1	0	2	2	2
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	1	0	2	2
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	3.0	Tier 3	0	3	3	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987	3.0	Tier 3	0	3	3	3
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	2002	NaN	Tier 2	0	0	2	2
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004	1.0	Tier 2	0	1	1	1
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009	2.0	Tier 3	2	2	2	2
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997	1.0	Tier 1	0	1	1	1

We then observed that the KNN imputer replaced NAN with 'Medium' which left us with two identical columns:

```
Med_X9                            imputed_X9
2    5203                         2    5203
1    2388                         1    2388
3     932                         3     932
Name: count, dtype: int64         Name: count, dtype: int64
```

The final step was encoding all other categorical columns which were achieved by:

- Frequency encoding X1 which had 1559 unique values:

```
frequency_encoding = train_df['X1'].value_counts().to_dict()
train_df['X1'] = train_df['X1'].map(frequency_encoding)
train_df['X1'].value_counts()

X1
6     2298
5     1975
7     1771
4      936
8      880
3      339
9      225
2       70
10      20
1        9
```

- Ordinal Encoding for X10:

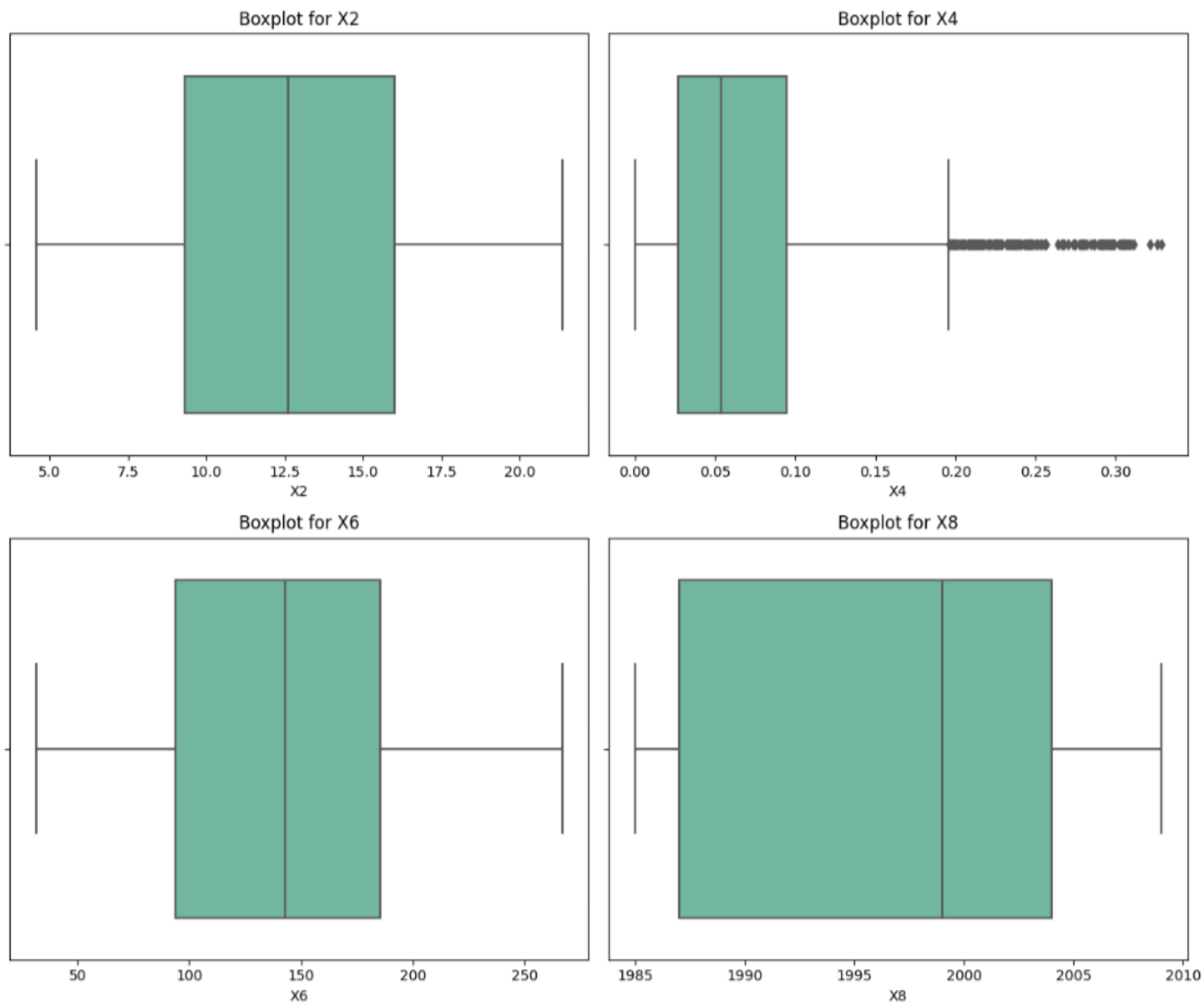
```
oe_X10 = OrdinalEncoder(categories=[['Tier 1', 'Tier 2', 'Tier 3']])
train_df['X10'] = oe_X10.fit_transform(train_df[['X10']])
```

- Label Encoding for all other categorical (after unifying format in X3):

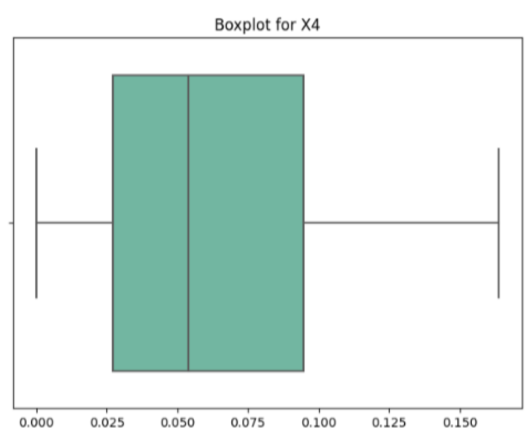
```
le_X3 = LabelEncoder()
train_df['X3'] = le_X3.fit_transform(train_df['X3'])

le_X5 = LabelEncoder()
train_df['X5'] = le_X5.fit_transform(train_df['X5'])

le_X7 = LabelEncoder()
train_df['X7'] = le_X7.fit_transform(train_df['X7'])
```

Numerical Data:

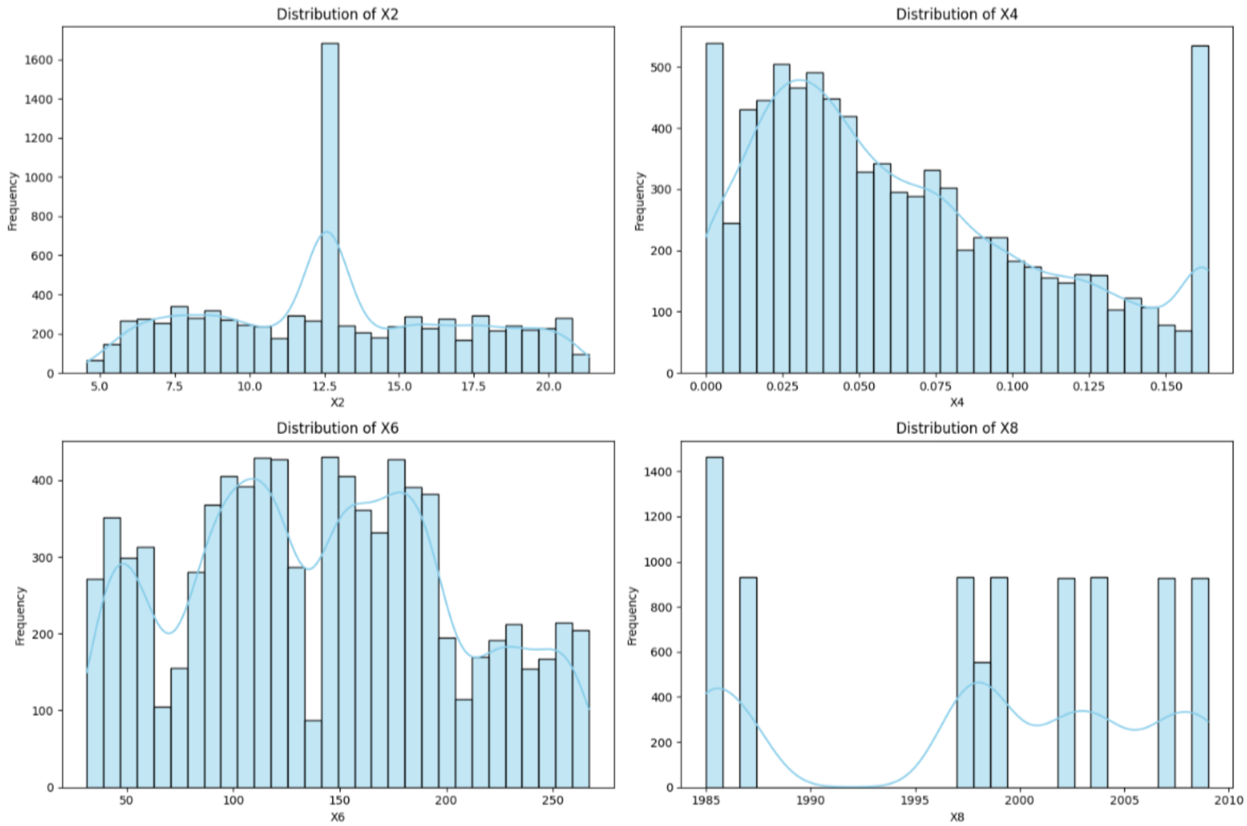
We detected outliers and found some in X4:



We clipped them at the upper and lower bound:

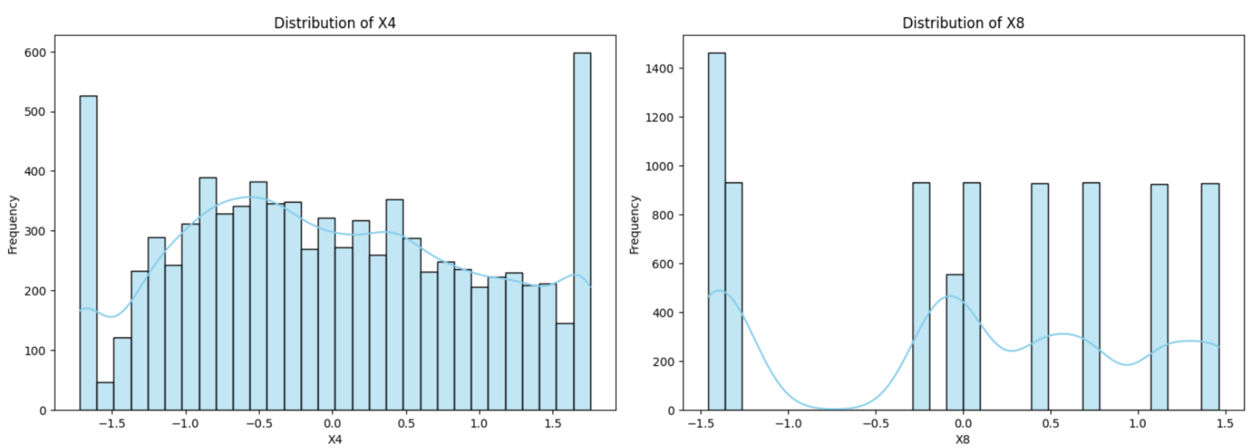


We then checked the skew of each column:



Through this we found that using the median for Null imputation is more feasible especially for columns like X4.

We replaced the nulls in X2 using Median and used Yeo-Johnson Transformation to deal with skew of X4 and X8 as it is a transformation that works for both the negatively and positively skewed columns and this is the distribution of X4 and X8 after:



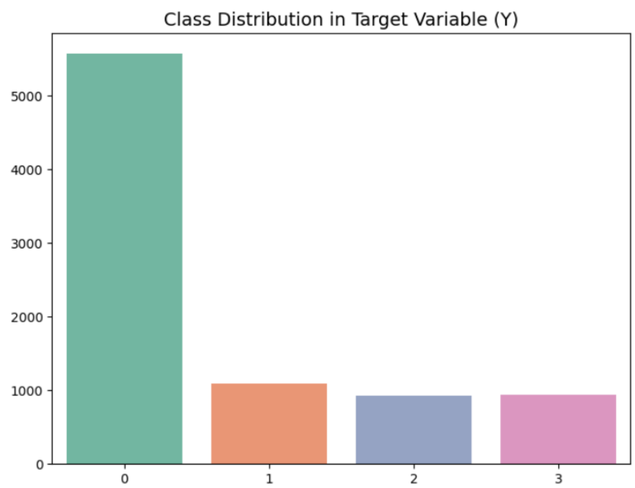
Lastly, we scaled the data into the [0, 1] Range using MinMax Scaler.

EDA

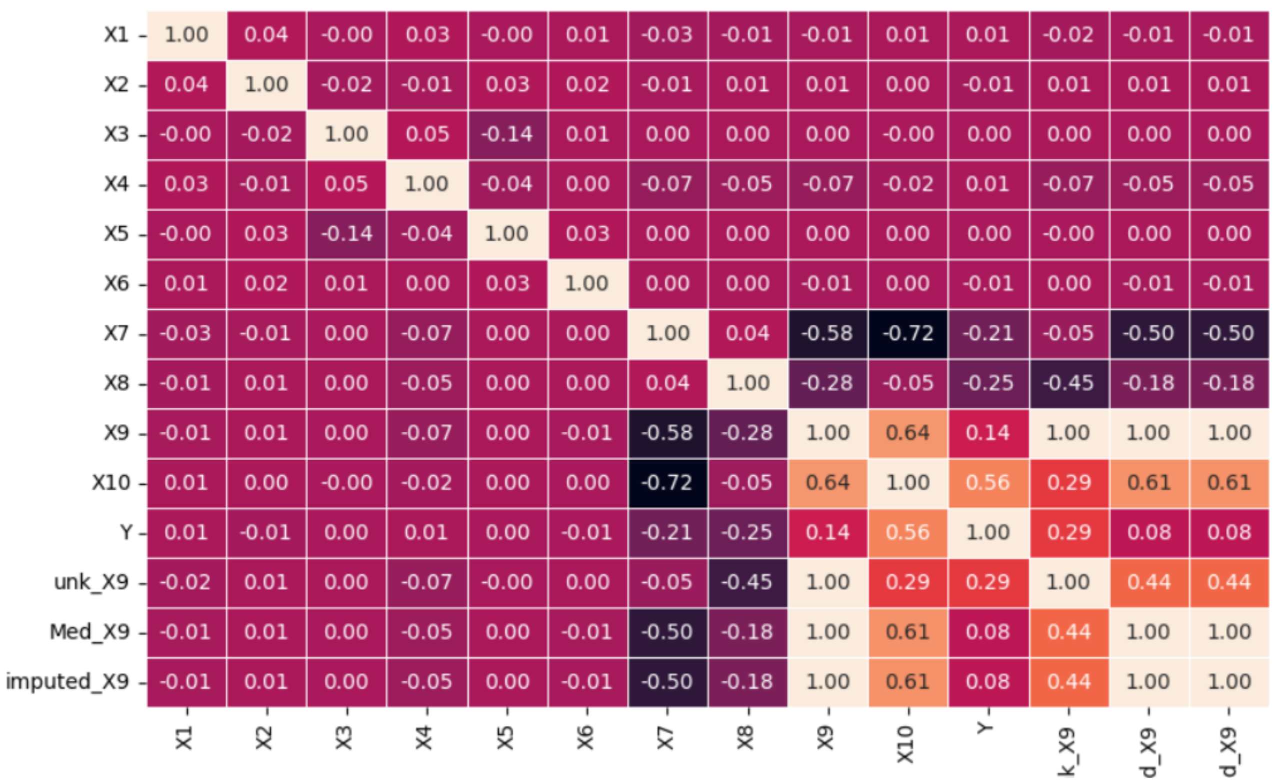
Secondly, we try to understand the relations between our features and target and what to use for training.

Class Distribution:

We found that class distribution is severely imbalanced:

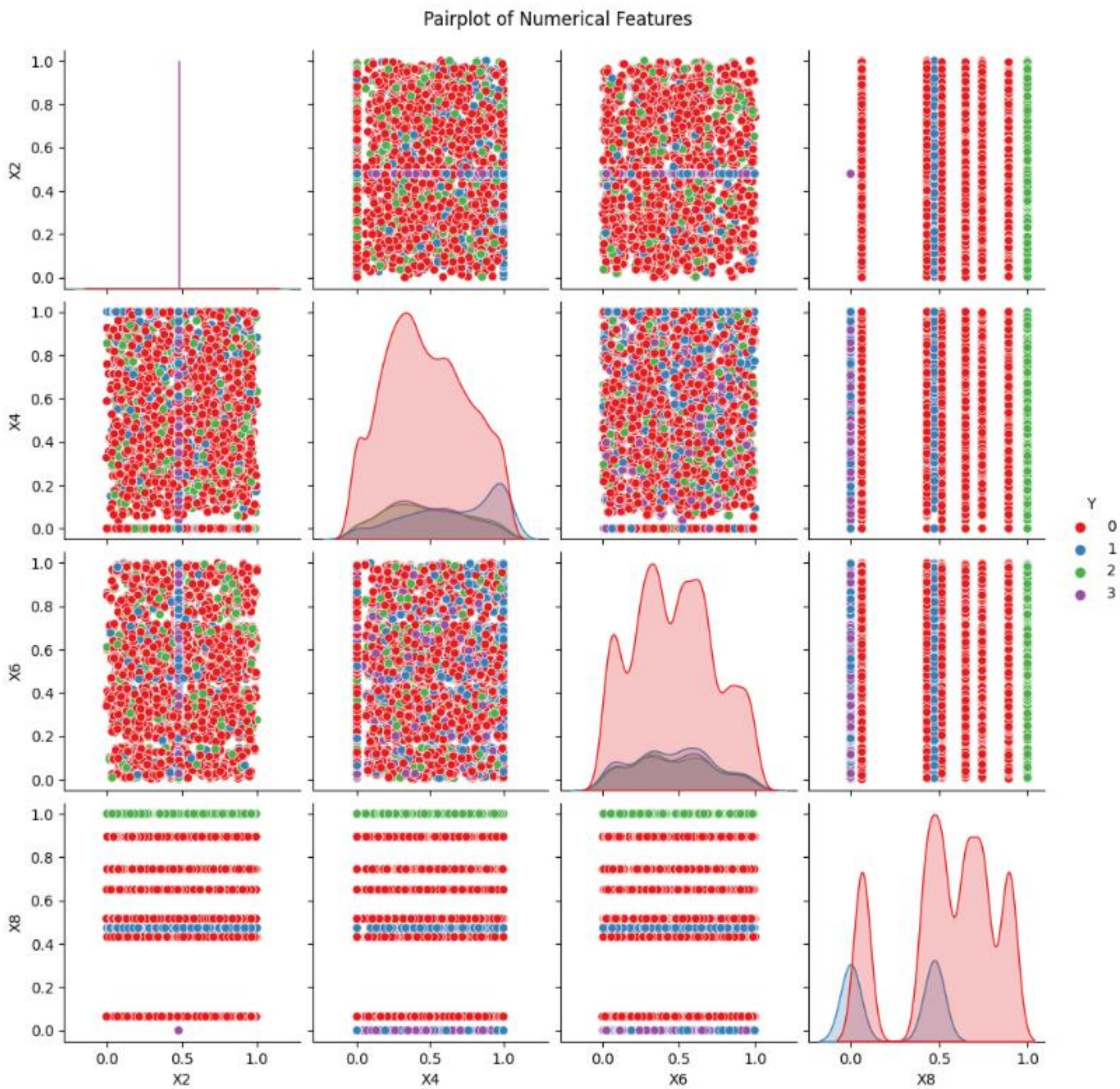


Then we visualized correlation with heat map:



Seeing as how unk_X9 has the highest correlation with Y we decided to drop all other X9 columns and copy unk_X9 into X9.

Using Pair Plot, we found that X8 has distinct distributions for classes while other numerical features contribute little to class separation.



Doing the same for categorical Data showed that X10 and X9 are the best class separators we have, both on their own and combined. X5 and X7 provide some info for class separation and X7 is best paired with X9 and X10:

We performed six different Feature Selection techniques, and they ranked them as follows:

Chi2: X7, X8, X9, X10, X1, X4, X2, X3, X5, X6.

Mutual Information: X7, X8, X9, X10, X2, X4, X1, X3, X5, X6.

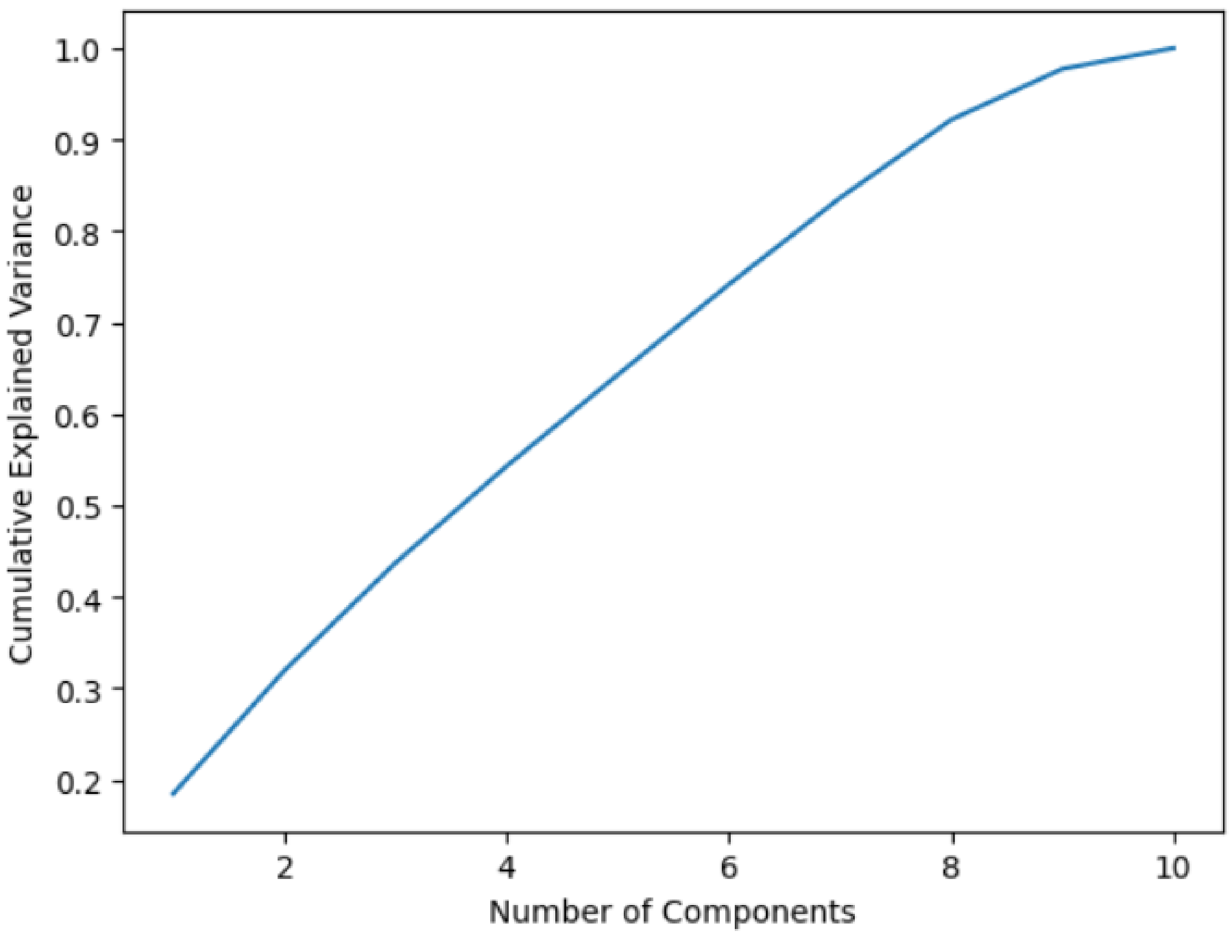
RF Classifier: X7, X8, X9, X10, X2, X4, X1, X3, X5, X6.

RFE (n_features_to_select=9): X1, X2, X4, X5, X6, X7, X8, X9, X10.

SFS (n_features_to_select=9): X1, X2, X3, X4, X5, X6, X7, X8, X9.

Lasso (n_features_to_select=9): X1, X2, X4, X5, X6, X7, X8, X9, X10.

We then used PCA Cumulative Explained Variance to choose the number of features to use:



We can use 9 or 10 features; we will use 10 because the dataset is small.

Modelling and Trails

Lastly, these are the models we use and their accuracies.

Models' Test and Validation Accuracy:

Column Head	Validation	Test
Logistic Regression	1.0	0.979
GMM	1.0	0.979
HMM 1	0.868	0.130
HMM 2	1.0	0.979
Bayesian Network	1.0	0.979
Naïve Bayes	0.86	0.685
Naïve Bayes w/ SMOTE	1.0	0.727