Machine Learning Approaches to Predicting Titanic Survival

Given a dataset containing statistics of each passenger aboard the Titanic, I aim to create a machine learning model to predict whether a passenger survived. This is a binary classification problem, where the outcome variable indicates survival. In order to build a reliable predictive model, I must identify the variables that are the most useful for prediction and clean the data. Data cleaning is important for ensuring the model makes reliable predictions, as noisy data can lead to misleading results.

Once the data is cleaned, I will evaluate whether any variables need to be standardized or normalized. This is important, as variables can have very different ranges, and without scaling, the large-scale features may be treated as more important. I will then apply feature engineering to create new variables that are more meaningful and interpretable. Finally, I will train two types of models: a generative model (Naive Bayes) and a discriminative model (Linear Regression). This comparison will show how these different approaches impact predictive performance on the same dataset.

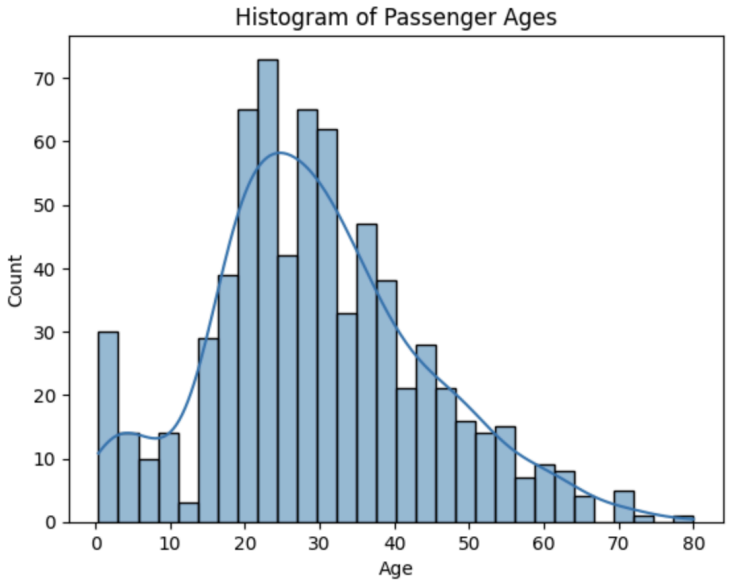
The Titanic dataset includes the target variable ‘Survived’, where 0 indicates the passenger did not survive and 1 indicates the passenger did survive. It also contains ‘PassengerId,’ a unique identifier for each passenger. ‘Pclass’ records the socio-economic class (1 = upper, 2 = middle, 3 = lower). ‘Name’ includes the full name and title of each passenger. ‘Sex’ indicates each passenger’s sex. ‘Age’ records each passenger's age in years. ‘Sibsp’ and ‘Parch’ indicate the number of siblings/spouses and parents/children aboard for each passenger. ‘Ticket’ gives the passenger’s ticket number, while ‘Fare’ represents the ticket price paid. ‘Cabin’ records the cabin number. Lastly, ‘Embarked’ indicates each passenger’s port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton).

The Titanic dataset contains several variables with missing or unstructured data. To prevent noisy data from affecting the model performance, I first examine the variables that have

missing values, as seen in the chart. ‘Age’ has 177 missing entries, representing around 20% of the dataset. Since this is a continuous variable and the percentage of missing values is

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

somewhat low, I imputed the missing entries with the median. I chose the median over the mean because the distribution of ‘Age’ is slightly right-skewed and the median handles skewed



distributions better. ‘Embarked’ has only two missing values. Since ‘Embarked’ is a categorical variable, I filled the missing values with the mode, which turns out to be “S” for Southampton. ‘Cabin’ has 687 missing values, which is about 77% of the dataset. Given this high proportion of missing data, I dropped the variable, as keeping it may produce erroneous results.

I also removed variables that are unlikely to contribute to predictive performance. I dropped ‘Ticket’ since it has inconsistent entries and is hard to classify and ‘PassengerId’ since

it is an identifier with no predictive value. Although ‘Name’ has many unique entries that are hard to classify, I will keep this for feature engineering, as the titles can be extracted to provide useful information.

After cleaning the dataset, we are left with the following variables: ‘Pclass’, ‘Name’, ‘Sex’, ‘Age’, ‘Sibsp’, ‘Parch’, ‘Fare’, and ‘Embarked’. I examine each variable to determine whether any further transformations are needed.

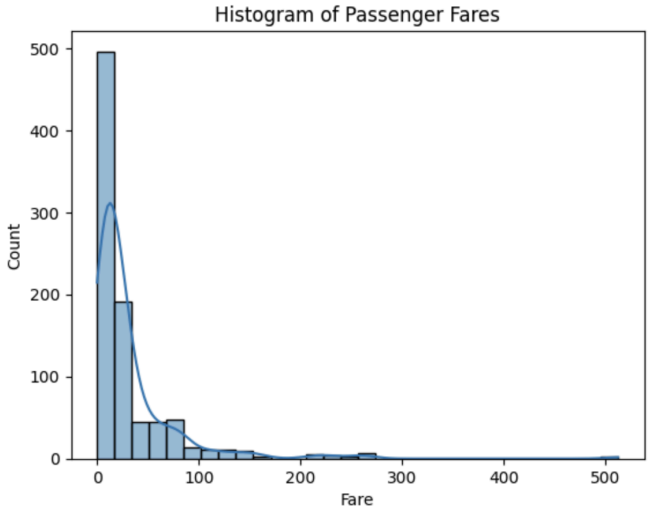
To ensure that categorical data is effectively used in many machine learning models, it must be converted to numeric format. Linear Regression and Naive Bayes, in particular, require numeric input. ‘Pclass’ is already a numeric categorical variable, so it remains as is. ‘Sex,’ is a categorical variable with only two categories (male and female), so I use one-hot encoding to create the binary variables ‘sex_male’ and ‘sex_female’. To avoid multicollinearity, I dropped ‘sex_female,’ since the sex of a passenger can be inferred from ‘sex_male’: a value of 1 indicates male and a value of 0 indicates female. ‘Embarked’, another categorical variable with only 3 categories, is also transformed using one-hot encoding, resulting in the variables ‘Embarked_Q’, ‘Embarked_S’, and ‘Embarked_C’. Once again, to avoid multicollinearity, I dropped ‘Embarked_C,’ as the other two variables allow inference of this category.

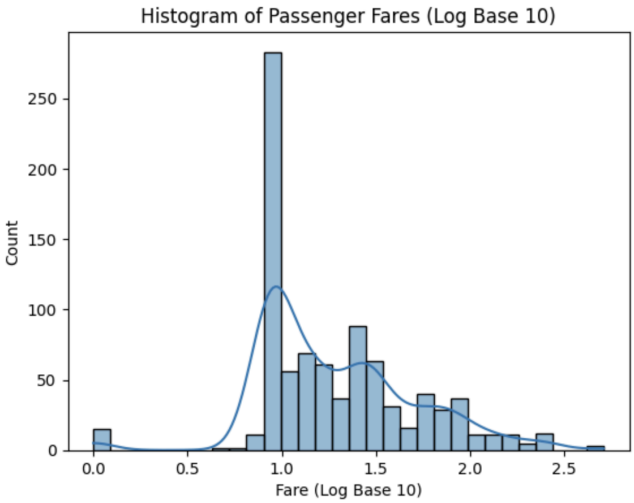
The last non-numeric value we have is ‘Name’. The names themselves do not help us predict the survival rate, but the titles associated with each passenger (e.g., Mr, Mrs) may

Title	
Mr	517
Miss	182
Mrs	125
Master	40
Dr	7
Rev	6
Col	2
Mlle	2
Major	2
Ms	1
Mme	1
Don	1
Lady	1
Sir	1
Capt	1
the Countess	1
Jonkheer	1

provide useful information. Thus, I extracted the titles from each name and examined their frequency. I consolidated similar titles: ‘Mlle’ and ‘Ms’ are updated to ‘Miss’ and ‘Mme’ is updated to ‘Mrs’. I grouped all other titles, which had a count of 7 or fewer (most with value 1), as ‘Rare’. After this transformation, there are five categories: Mr, Miss, Mrs, Master, and Rare. I then applied one-hot encoding to convert these titles into numeric format.

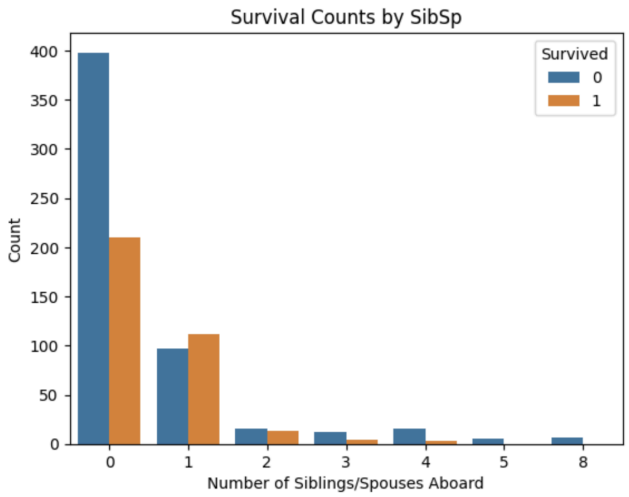
‘Fare’ is a continuous variable that is heavily right-skewed. Many models perform better when continuous features approximate a normal distribution. To make this distribution more

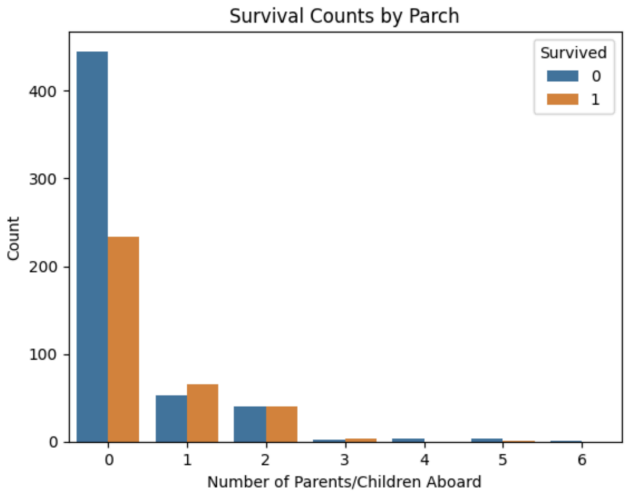




normal, I apply a log base 10 transformation to the new variable ‘Fare_log10’. This transformation spreads out the low values, reducing skewness and producing a distribution that is more normally shaped.

To examine the effect of family relationships on survival, I create visualizations for ‘Sibsp’ and ‘Parch’. Both histograms show that passengers with fewer siblings/spouses and

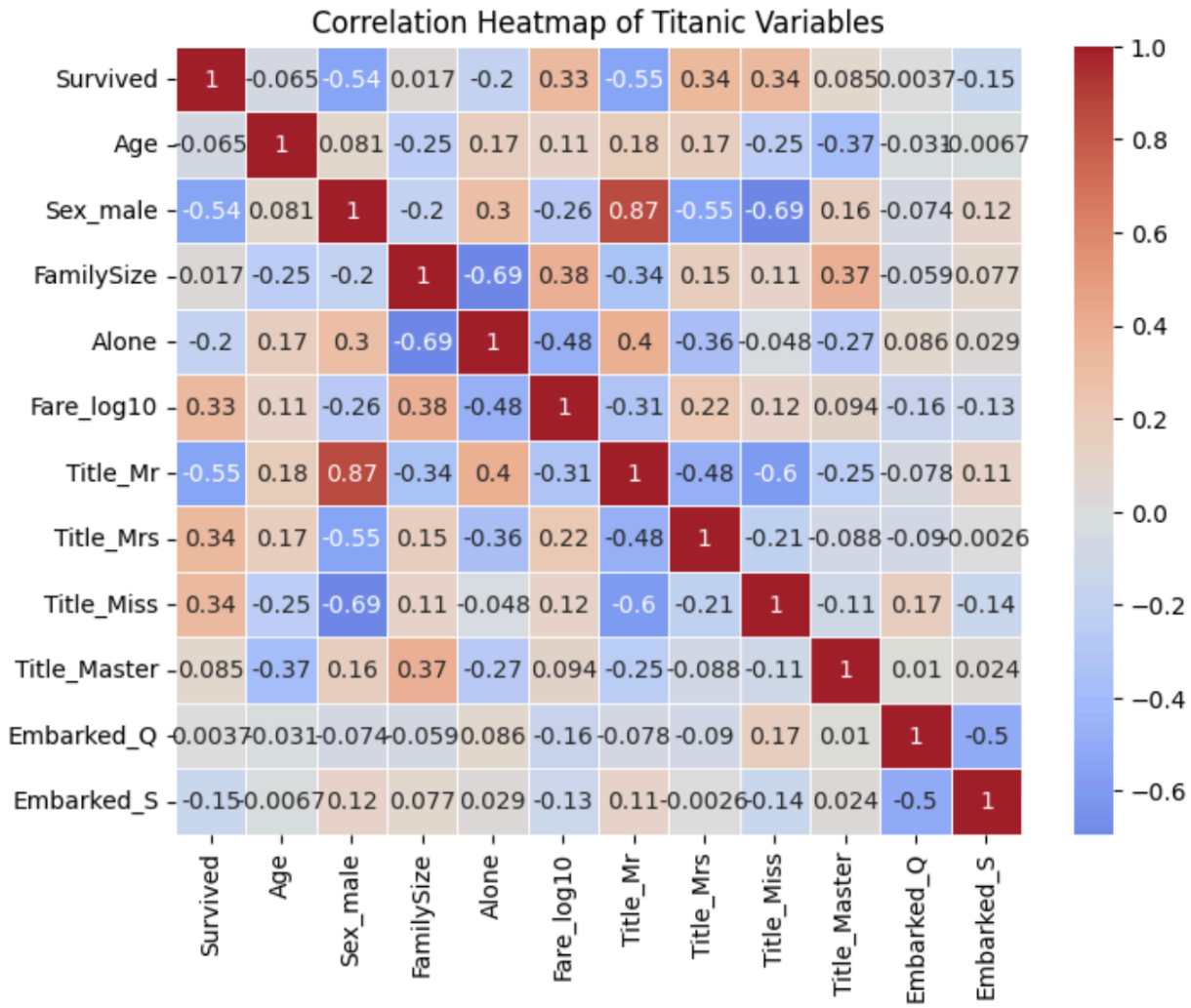




parents/children have a higher likelihood of survival. Since both variables convey similar information, I combine them into a one variable called ‘FamilySize’, which represents the size of each passenger’s family (‘FamilySize’ = ‘Sibsp’ + ‘Parch’ + 1). This combines two similar features into one more meaningful one and simplifies the model. In addition, I created a binary variable called ‘Alone’, where 1 indicates the passenger was alone (‘FamilySize’ = 0) and 0 indicates the passenger was not alone (‘FamilySize’ >= 1).

After these transformations, all variables except ‘Age’ have va lues roughly within the range of 0-3. To ensure ‘Age’ does not disproportionately influence the model, I standardize it by rescaling it using its standard deviation.

Before starting on the model training, I created a heat map to observe correlations between the features and survival. As seen in the map, ‘Sex_male’ and similarly ‘Title_Mr’ have



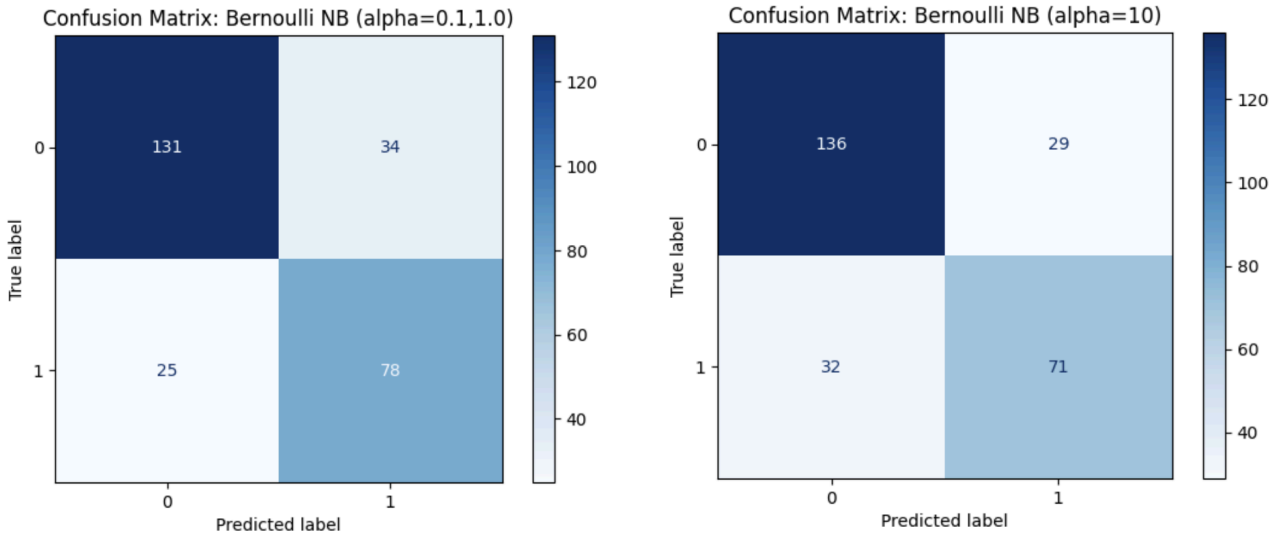
the strongest correlations with survival. The negative relationship tells us that male passengers and passengers with the title “Mr” are less likely to survive. In addition, ‘Fare_log10’, ‘Title_Mrs’, ‘Title_Miss’, and ‘Alone’ also have strong positive correlations. Since the sex and titles have the strongest correlation with survival (binary variables), I will use the Bernoulli Naive Bayes model so that these variables will be included.

Before training the model, I define the dependent and independent variables. Since the goal is to predict whether a passenger survived, the independent variable is ‘Survived’. The Bernoulli Naive Bayes model only handles binary variables, so the independent variables I use are ‘Sex’ (one-hot encoded), ‘Embarked’ (one hot-encoded), ‘Alone’ (created from ‘FamilySize’), and ‘Title’ (extracted from ‘Name’ and one-hot encoded).

First, I train a Naive Bayes model using a 70/30 train/test split. To determine an alpha value for Laplace smoothing, I test alpha values of 0.1 and 1.0, and 10.0. The alpha value of 0.1 and 1.0 result in testing accuracies of 0.780, while an alpha value of 10.0 achieves the lowest testing accuracy of 0.772. The different accuracy results based on the different alpha values showed the bias-variance tradeoff of Laplace smoothing. Very small alpha values allow the model to closely follow the training data, while larger alpha values lead to stronger smoothing and less overfitting. In this case, the lower alpha values perform better because they apply just enough smoothing without adding excessive bias, allowing the model to make more accurate predictions.

To better understand how well each smoothing value predicts survival, I examine the confusion matrices for each alpha value. For lower alpha values of 0.1 and 1.0, the model correctly classifies a large number of non-survivors (true negatives) while also capturing more actual survivors (true positives), though this comes with a small increase in false positives. When the alpha value increases to 10.0, the model improves slightly at identifying non-survivors but misses more actual survivors, as shown by the higher false negatives. Overall, lower smoothing values (smaller alpha) result in a more balanced model that captures both survivors

and non-survivors effectively, while higher smoothing values (larger alpha) increase bias toward predicting the majority class, improving true negatives but causing more false negatives.



Examining the classification report also gives us a lot of information about predictive performance of the model. Here we examine the report for the alpha value of 0.1. The precision

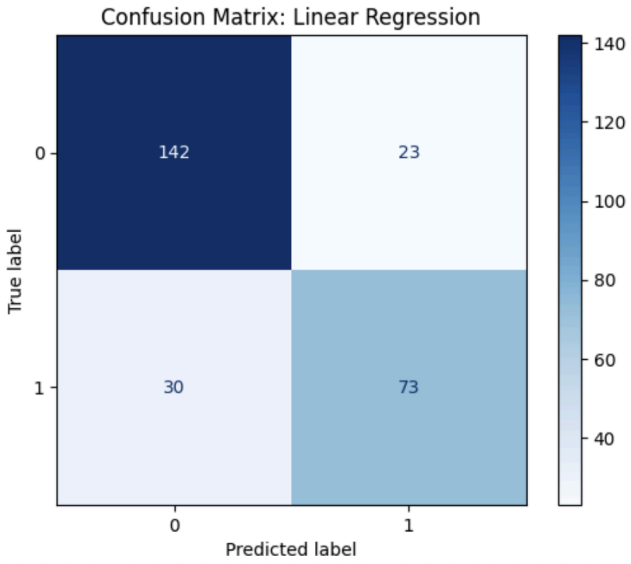
```
Classification Report:
              precision    recall  f1-score   support

           0      0.840     0.794     0.816       165
           1      0.696     0.757     0.726       103

    accuracy                         0.780       268
   macro avg      0.768     0.776     0.771       268
weighted avg      0.785     0.780     0.781       268
```

metric indicates that the model predicts negative cases (non-survivors) more accurately, with 84.0% correct, compared to positive cases (survivors), with 69.6% correct.

I also compare the training accuracy at the alpha value of 0.1, which is 0.799, to the testing accuracy of 0.780. The similarity between the training and testing accuracy suggest that the model performs well without overfitting.

Next, I train a linear regression model on the same Titanic dataset. Linear regression allows for continuous variables, so I will use the following as the independent variables: ‘Age’,

Sex (one-hot encoded), ‘FamilySize’, ‘Alone’, ‘Fare’ (transformed using log base 10), ‘Title’ (one-hot encoded variables), and ‘Embarked’ (one-hot encoded variables). I use the same 70/30 train/test split to ensure a fair comparison. With a probability threshold of 0.5, the linear

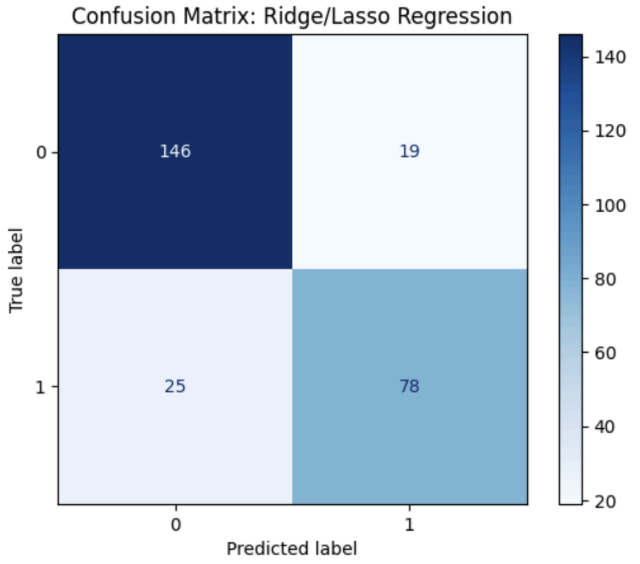


Classification Report:	precision	recall	f1-score	support
0	0.826	0.861	0.843	165
1	0.760	0.709	0.734	103
accuracy			0.802	268
macro avg	0.793	0.785	0.788	268
weighted avg	0.801	0.802	0.801	268

regression model achieves an accuracy score of 0.832. The confusion matrix shows us that the model correctly classifies a large number of non-survivors (true negatives) and actual survivors (true positives). According to the classification report, the model performs better at predicting negative cases (non-survivors), correctly identifying 82.6% of them, compared to positive cases (survivors), correctly predicting 76.0%.

I also explore two versions of linear regression: Ridge Regression and LASSO regression. These methods both add a penalty term to the loss function to reduce overfitting. Ridge Regression penalizes the square of the coefficient magnitudes, while LASSO Regression penalizes their absolute values.

To select the best alpha for each model, I iterate through all logarithmically spaced values from 10^{-3} to 10^{2}. The most accurate models result from an alpha value of 0.12 for Ridge Regression and 0.001 for LASSO Regression, both achieving an accuracy score of 0.836. See below for the confusion matrix and classification report for both regression models (they produce identical results). Similar to the Naive Bayes model, the confusion matrix shows us that



Classification Report:

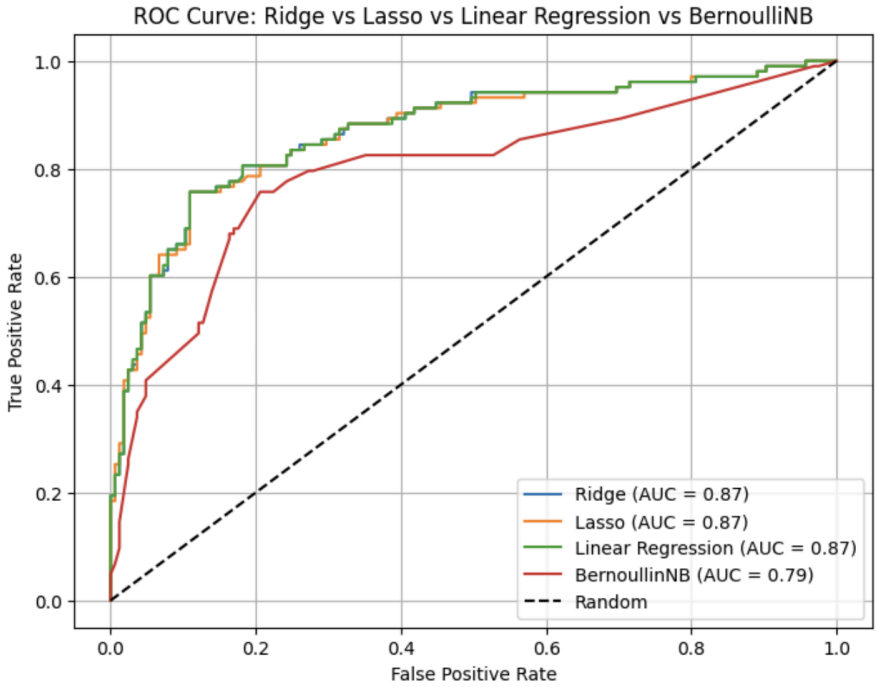
	precision	recall	f1-score	support
0	0.854	0.885	0.869	165
1	0.804	0.757	0.780	103
accuracy			0.836	268
macro avg	0.829	0.821	0.825	268
weighted avg	0.835	0.836	0.835	268

the model correctly classifies a large number of non-survivors (true negatives) and actual survivors (true positives). According to the classification report, the model performs better at predicting negative cases (non-survivors), correctly identifying 85.4% of them, compared to positive cases (survivors), correctly predicting 80.4%.

Ridge and LASSO Regression outperformed the standard linear regression model. By applying regularization, these methods successfully reduce overfitting by shrinking coefficients.

Overall, the Bernoulli Naive Bayesian model achieves an accuracy score of 0.780, while the Linear Regression model achieves a score of 0.832 and the Ridge and LASSO Regressions achieve the highest accuracy score of 0.836. This indicates that the regression models, specifically the Ridge and LASSO variants, are generally better predictors of survival on the Titanic dataset. In order to further compare and visualize the models, I created an ROC curve for all four models. The linear, Ridge, and LASSO regression models all perform very well, correctly distinguishing between survivors and non-survivors approximately 87% of the time, as indicated by their AUC score. Bernoulli Naive Bayes also performs well, though slightly worse with an AUC score of 0.79. Overall, these results suggest that while both generative and

discriminative approaches are effective, the regression models provide a marginally stronger predictive performance for the Titanic dataset.



One of the limitations to the Bernoulli Naive Bayes model is that it only supports binary features. As a result, the continuous and ordinal categorical variables, ‘Age’, ‘Fare’, and ‘Pclass’ cannot be included in the model. Since these variables are correlated with survival, their exclusion likely hurt the model’s predictive power. A workaround would be to group the ‘Age’ and ‘Fare’ variables into categories and apply one-hot encoding. However, this takes away some of the complexity of the data that could be important. Gaussian Naive Bayes, on the other hand, can use these continuous variables. However, I would then need to exclude the binary variables. This is a big limitation to Naive Bayes, since each variant is restricted to a single variable type.

Linear regression also has limitations, specifically when the relationship between the predictors and target is not linear. However, applying Ridge and LASSO techniques, as shown in this analysis, helps regularize the model and prevent overfitting.

I used chatGPT to help me generate some of the code for feature engineering, creating diagrams, and comparing models. I wrote my own code for data cleaning and most of the feature engineering steps. In the feature engineering step, chatGPT helped me with extracting the title from the ‘Name’ variable and with converting the one-hot encoded variables into numeric 0/1 format. I wrote my own code for the training setup process as well as the sections that trained the Naive Bayes and regression models. However, ChatGPT helped me generate starter code for the histograms, confusion matrices, classification reports, and the ROC curve graph. I was able to create the heatmap myself. Lastly, ChatGPT was helpful in suggesting which libraries and functions to import for specific tasks. The written report was done myself, although I occasionally used chatGPT to fact-check certain statements to confirm my understanding of the analysis.

I have noticed that ChatGPT generated code is usually thorough and functional, but often more complex and less readable than the code I write myself or see online. For example, if I ask it to create a graph, it will not only create one, but make it look extremely presentable, with adjusted colors, titles, and labels. This is helpful when I need an attractive graph quickly, but the code can sometimes be difficult to interpret or modify. Similarly, if I ask ChatGPT to write me a function, it creates a very useful one, but it can be quite complex, making it difficult to make manual edits to. On the other hand, my code as well as other human code I see online is much simpler and easier to understand. I prefer human-generated code, as I find it more to the point and easier to add on to as models get more complex, even though it takes longer to create.