*(Notes in italic parathesis)*

*(Reformatting probably needed)*

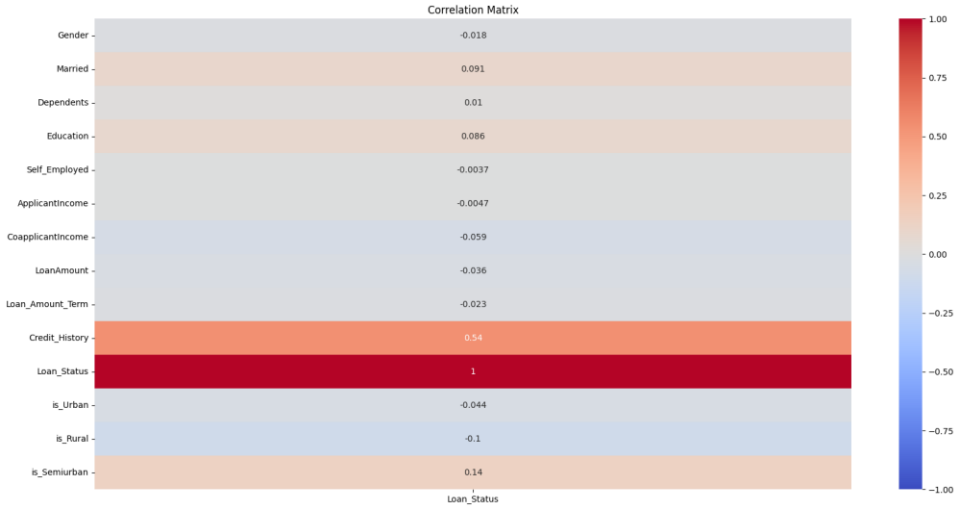
**Predicting Loan Approval Using Machine Learning *(TBD)***

**Intro** *(May need rewording)*

When providing loans banks look at multiple aspects of a person's financial status and using these, they determine if a prospective borrower would be able to repay their loan. We plan to create a machine learning model that will predict if a person would be approved for a loan or not. This model would be helpful for perspective borrowers to determine if they would be able to receive their loan. The model may also be useful for banks to speed up their loan selection process.

**Data** *(May need rewording)*

Our data set is hosted on Kaggle and contains 614 entries and 13 features that describe a person's financial state and their loan status. The data set contains some missing values in the Gender, Married, Dependents, Self_Employed, LoanAmmount, Loan_Amount_Term, and Credit_History columns so first we will impute these missing values. LoanAmmount will be imputed using its mean due to it being a continuous variable while the other columns will be imputed using their modes due to them being either categorical or discreate values. Next, we need to transform the categorical values into binary values so that the algorithms will be able to process them. Gender, Married, Education, Self_Employed, and Loan_Status can be simply encoded as 1 true and 0 is false while Property_Area needs to be encoded as 3 different columns due to it featuring 3 different categories with now inherit ordering. The last cleaning task is to fix the Dependents column as it represents 3 or more dependents as 3+. We will simply remove the + and reencode the attributes as integers instead of characters. Once the data is clean, we can run a correlation matrix to show the relationship between our features.



Looking at the correlation matrix we can see highest correlation to Loan_Status is Credit_History at 0.54

We then split the data into an 80/20 train and test splits and standardized the data sets so that our models would better map the relationship between the data points.

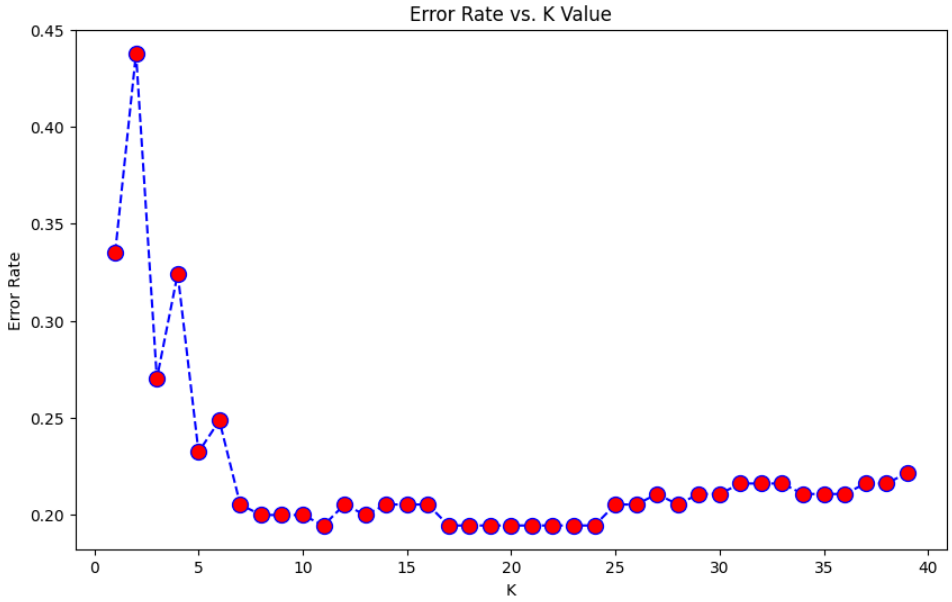
**Results**

**Baseline**

We started our modeling by creating a baseline model. For this we chose a Naive Bayes classifier. *(Say why we chose Naive Bayes)*. Using the python package Scikit Learn's built in GaussianNB we created a default Naive Bayes classifier from which we received an accuracy score of about 78%.

**K Nearest Neighbors**

For our next model we chose a K nearest neighbors classifier using the python package Scikit Learn built in KNeighborsClassifier. *(Say why we chose K nearest neighbors classifier)*. We first ran this model with neighbors = 3 resulting in and accuracy of about *(Rerun on our cleaned data)*%. We then created a plot of error rate vs K value so that we could find the most optimal K value. *(Temp Image)*.




Using this graph, we then created a KNN model with Neighbors = *(Lowest value on new graph)*. This new model gave us an accuracy of about *(new model accuracy)*%.

**Neural Net**

For our most advanced model we created a neural network using the python package Scikit Learn built in MLPClassifier. We started off with the default parameters for the MLPClassifier except for randomstate and verbose. Randomstate to remain consistent when rerunning and verbose to see progress on the runs. Running this model we got an accuracy of about 76%. We then ran a Grid search using these parameters. *(This will definitely need to be reformatted)*

'learning_rate_init': [0.001, 0.01, 0.1], - controls the step-size in updating the weights

'hidden_layer_sizes': [(50, 50, 50), (100, 50, 100), (100, 100, 100), (25, 50, 25)], - represents the hidden layers in the model and their sizes (size of 1st layer, size of second layer, size of third layer)

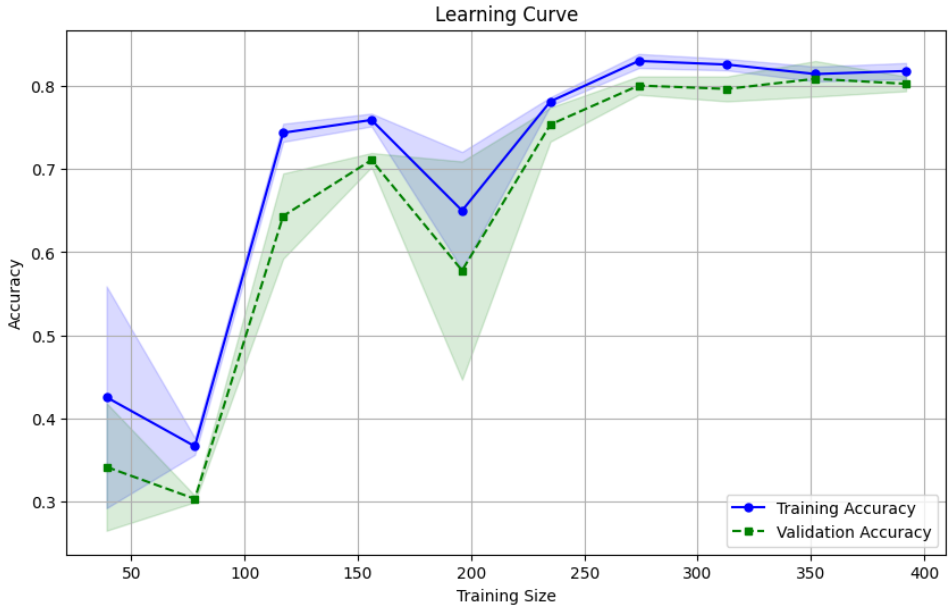
'learning_rate': ['adaptive', 'constant'], - constant learning rate remains learning_rate_init adaptive changes learning rate if training loss does not decrease.

'early_stopping': [True, False] - If True sets aside 10% of data for validation and ends training if validation score does not improve by at least the tolerance for optimization: default= 0.0001 for 10 epochs.

Of the parameters we tested the grid search returned these as the best
MLPClassifier(early_stopping=True, hidden_layer_sizes=(50, 50, 50),
learning_rate='adaptive', random_state=42,

verbose=True, learning_rate_init: 0.001)

Runing this model on our data set returned an accuracy of about 80% with this learning curve.




**Conclusions** *(May need rewording)*

Overall our more advanced models did not provide a marked improvement over our base line model, roughly a 2% increase in accuracy. With an 80% accuracy this model is a decent predictor of loan approval so it may be valuable for banks or customers to use to help speed up the loan process but should not be the end of the line of loan approval. A real person should still look over the results of these cases while using this model to help them decide.

**Contributions**

*(IDK what to put here exactly and we are already over the page limit so reformat an come back to this)*