Loan Approval Predication

Assessment 4

Name: Sai Teja Gilukara

Roll no: 18210

Problem Statement Discussion

We know that loans cannot be given to everyone. A background check of the customer is mandatory. By seeing the customers past and present records, the employee understands the capability of the customer.

This project aims to determine if a customer is eligible for a loan feeding the algorithm with required inputs.

Code

Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plot
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from keras.layers import Dropout
from keras import regularizers
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
```

Importing Data Set

```
train_df = pd.read_csv("train.csv")    #importing the dataset
print(train_df.info())                 #Brief description about the dataset
train_df = train_df.drop(columns= ["Loan_ID"])
```

Analyzing the dataset by graphs

By looking at the graphs, We will try to get some conclusions. This is a very important step. All having a data like this, W

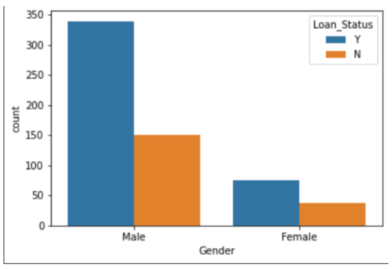
1) generalise or remove the columns which doesnt have any

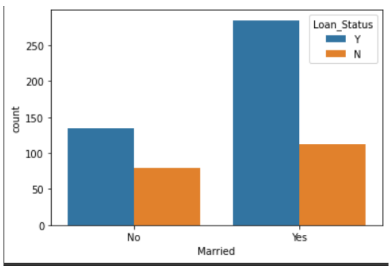
significane on making the decision.

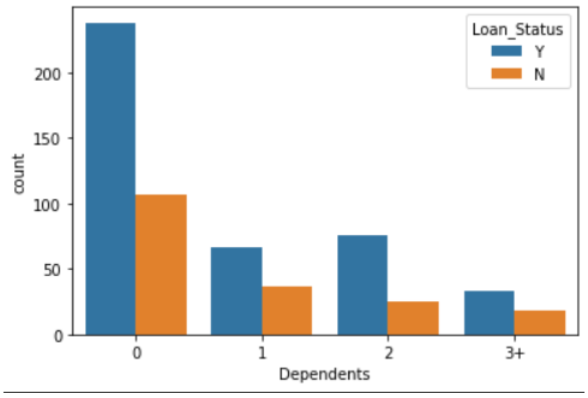
2) remove the outliners. On large datasets, the computation can be decreased

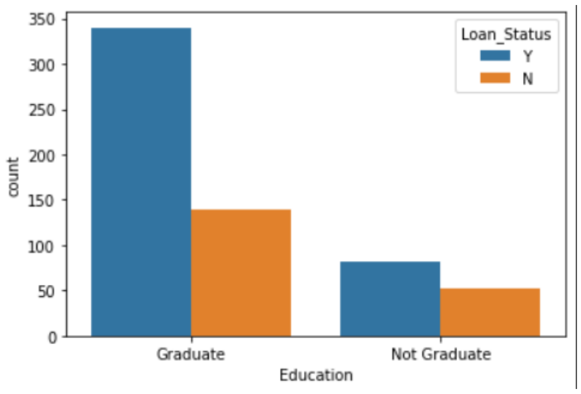
```
sb.countplot(x=train_df['Gender'], data=train_df, hue='Loan_Status')
sb.countplot(x=train_df['Married'], data=train_df, hue='Loan_Status')
sb.countplot(x=train_df['Dependents'], data=train_df, hue='Loan_Status')
sb.countplot(x=train_df['Education'], data=train_df, hue='Loan_Status')
sb.countplot(x=train_df['Self_Employed'], data=train_df, hue='Loan_Status')
sb.countplot(x=train_df['Credit_History'], data=train_df, hue='Loan_Status')
sb.countplot(x=train_df['Loan_Status'], data=train_df, hue='Loan_Status')
```

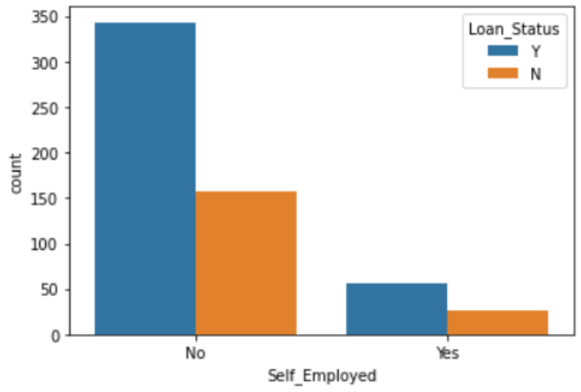
Plots:

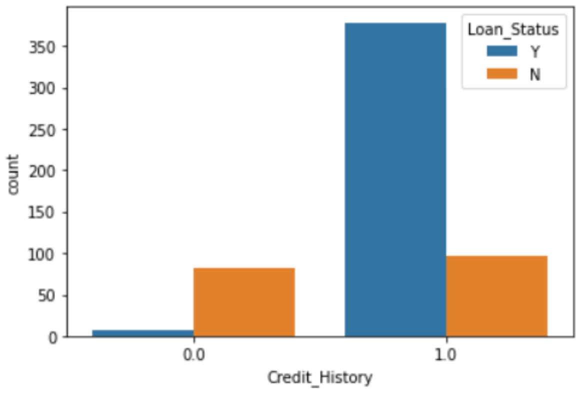


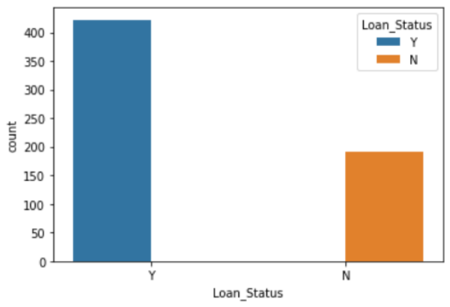








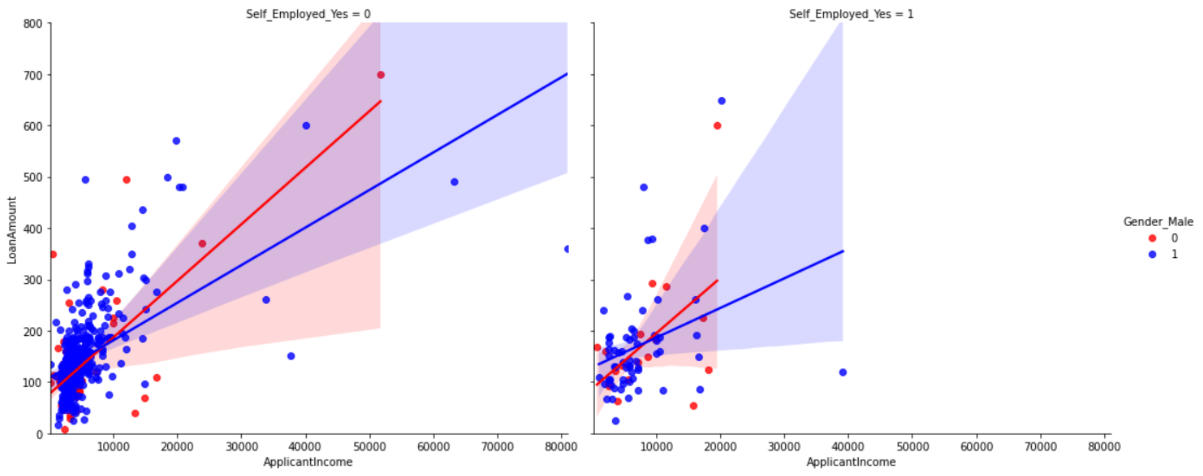




Interpreted data from above graphs

- Most of the people who apply are men (3 times more) and there are higher chances the loan gets sanctioned if the applicant sex is 'Male'
- 2/3 rd of the people who applied for loan are married and married applicants are more likely to get loan sanctioned
- If the dependents are 0, there is highly likely that loan is approved
- Graduated population are more likely to get the loan
- 5/6th of population is not self employed
- Applicant with credit history are far more likely to be accepted.
- We can conclude, 2/3 of people who applied got there loan sanctioned.

```
g = sb.lmplot(x='ApplicantIncome',y='LoanAmount',data= train_df_encoded , col='Self_Employed_Yes', hue='Gender_Male',
             palette= ["Red" , "Blue","Yellow"] ,aspect=1.2,size=6)
g.set(ylim=(0, 800))
```



Above graph tells:

- The male applicants take more amount of loan than female.
- The males are higher in number of "NOT self employed" category.
- Majority of applicants are NOT self employed.
- The majority of income taken is about 0-200 with income in the range 0-20000.

Preprocessing the data

The data needs to be pre processed to fit to the model. 2 reasons why this is mandatory is

- Missing values (Imputing)

- few columns have features in test format. these needs to be converted in number format (Encoding).

```
train_df_encoded = pd.get_dummies(train_df,drop_first=True) #Convert categorical variable into dummy/indicator variables.
print(train_df_encoded.head())
X = train_df_encoded.drop(columns='Loan_Status_Y')
Y = train_df_encoded['Loan_Status_Y']
```

Train and Test split

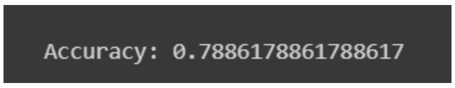
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.20,stratify =Y)
from sklearn.impute import SimpleImputer
temp = SimpleImputer(strategy='mean')
X_train = temp.fit_transform(X_train)
X_test = temp.fit_transform(X_test)
```

Train Models

Linear Regression - Classification

Linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable.

```
lrc = LinearRegression()
lrc.fit(X_train, Y_train)
temp= lrc.predict(X_test)
Y_pred_lrc = []
for i in temp:
  if i>0.5:
    Y_pred_lrc.append(1)
  else:
    Y_pred_lrc.append(0)
# Accuracy score
result2 = accuracy_score(Y_test, Y_pred_lrc)
print("\nAccuracy:",result2)
```

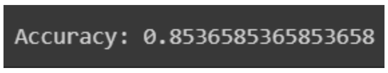


```
Accuracy: 0.7886178861788617
```

Logistic Regression

Logistic Regression gives the probability associated with each category or each individual outcome. The probability function is joined with the linear equation using probability distribution. In Logistic Regression we use binomial distribution where we work on two category problems.

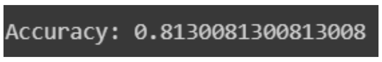
```
LR = LogisticRegression(solver='lbfgs')
LR.fit(X_train, Y_train)
Y_pred_LR = LR.predict(X_test)
# Accuracy score
result2 = accuracy_score(Y_test, Y_pred_LR)
print("\nAccuracy:",result2)
```


```
Accuracy: 0.8536585365853658
```

Support Vector machines (SVM)

Support Vector Machine (SVM) is a supervised binary classification algorithm. Given a set of points of two types in N dimensional place SVM generates a (N−1) dimensional hyperplane to separate those points into two groups.

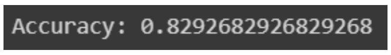
```
SVM = SVC(kernel='linear')
SVM.fit(X_train, Y_train)
Y_pred_SVM = SVM.predict(X_test)
# Accuracy score
result2 = accuracy_score(Y_test, Y_pred_SVM)
print("\nAccuracy:",result2)
```


```
Accuracy: 0.8130081300813008
```

Random Forest Classifier

Random Forest is a mainstream AI algorithm that has a place with the regulated learning strategy. It depends on the idea of ensemble learning, which is a cycle of joining numerous classifiers to tackle an intricate issue and to improve the presentation of the model and it might be used for both Classification and Regression issues in ML.

```
random_forest = RandomForestClassifier(n_estimators= 100)
random_forest.fit(X_train, Y_train)
Y_pred_rf = random_forest.predict(X_test)
result2 = accuracy_score(Y_test, Y_pred_rf)
print("\nAccuracy:",result2)
```

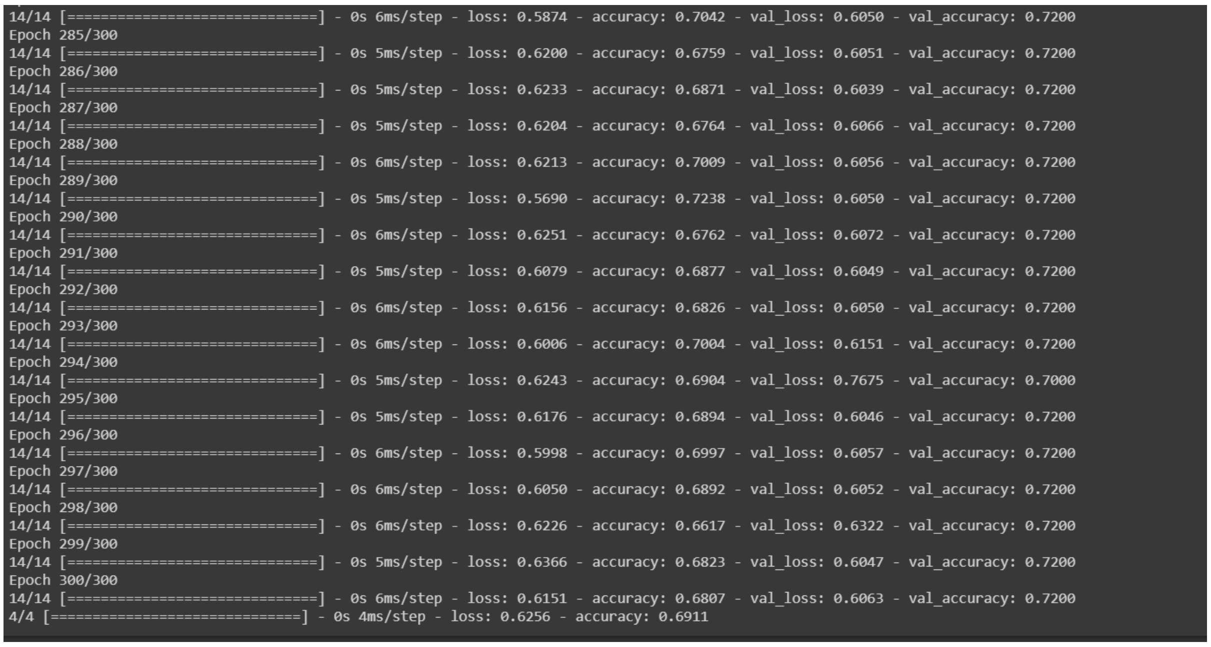

```
Accuracy: 0.8292682926829268
```

Artificial Neural Network

An **artificial neural network** is essentially a computational network based on biological neural networks. These models aim to duplicate the complex network of neurons in our brains (because imagine what a computer can do if it operates like our brains).

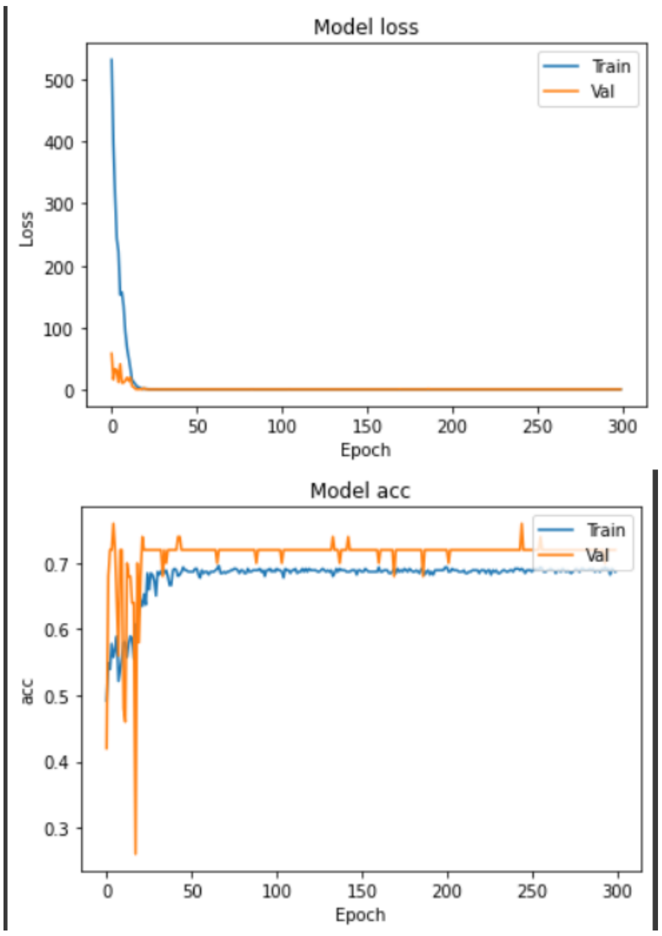
So this time, the nodes are programmed to behave like actual neurons. Although they're really artificial neurons that try to behave like real ones, hence the name "artificial neural network."

```
model = Sequential()
model.add(Dense(16 ,activation='relu',input_shape=(14,)))
model.add(Dense(16,activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='rmsprop', metrics=['accuracy'])

hist = model.fit(X_train, Y_train,epochs=300,verbose=1, validation_split=0.1)
accuracy = model.evaluate(X_test,Y_test)[1]
```



```
14/14 [==============================] - 0s 6ms/step - loss: 0.5874 - accuracy: 0.7042 - val_loss: 0.6050 - val_accuracy: 0.7200
Epoch 285/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6200 - accuracy: 0.6759 - val_loss: 0.6051 - val_accuracy: 0.7200
Epoch 286/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6233 - accuracy: 0.6871 - val_loss: 0.6039 - val_accuracy: 0.7200
Epoch 287/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6204 - accuracy: 0.6764 - val_loss: 0.6066 - val_accuracy: 0.7200
Epoch 288/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6213 - accuracy: 0.7009 - val_loss: 0.6056 - val_accuracy: 0.7200
Epoch 289/300
14/14 [==============================] - 0s 5ms/step - loss: 0.5690 - accuracy: 0.7238 - val_loss: 0.6050 - val_accuracy: 0.7200
Epoch 290/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6251 - accuracy: 0.6762 - val_loss: 0.6072 - val_accuracy: 0.7200
Epoch 291/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6079 - accuracy: 0.6877 - val_loss: 0.6049 - val_accuracy: 0.7200
Epoch 292/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6156 - accuracy: 0.6826 - val_loss: 0.6050 - val_accuracy: 0.7200
Epoch 293/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6006 - accuracy: 0.7004 - val_loss: 0.6151 - val_accuracy: 0.7200
Epoch 294/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6243 - accuracy: 0.6904 - val_loss: 0.7675 - val_accuracy: 0.7000
Epoch 295/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6176 - accuracy: 0.6894 - val_loss: 0.6046 - val_accuracy: 0.7200
Epoch 296/300
14/14 [==============================] - 0s 6ms/step - loss: 0.5998 - accuracy: 0.6997 - val_loss: 0.6057 - val_accuracy: 0.7200
Epoch 297/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6050 - accuracy: 0.6892 - val_loss: 0.6052 - val_accuracy: 0.7200
Epoch 298/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6226 - accuracy: 0.6617 - val_loss: 0.6322 - val_accuracy: 0.7200
Epoch 299/300
14/14 [==============================] - 0s 5ms/step - loss: 0.6366 - accuracy: 0.6823 - val_loss: 0.6047 - val_accuracy: 0.7200
Epoch 300/300
14/14 [==============================] - 0s 6ms/step - loss: 0.6151 - accuracy: 0.6807 - val_loss: 0.6063 - val_accuracy: 0.7200
4/4 [==============================] - 0s 4ms/step - loss: 0.6256 - accuracy: 0.6911
```

```
plot.plot(hist.history['loss'])
plot.plot(hist.history['val_loss'])
plot.title('Model loss')
plot.ylabel('Loss')
plot.xlabel('Epoch')
plot.legend(['Train', 'Val'], loc='upper right')
plot.show()
plot.plot(hist.history['accuracy'])
plot.plot(hist.history['val_accuracy'])
plot.title('Model acc')
plot.ylabel('acc')
plot.xlabel('Epoch')
plot.legend(['Train', 'Val'], loc='upper right')
plot.show()
```



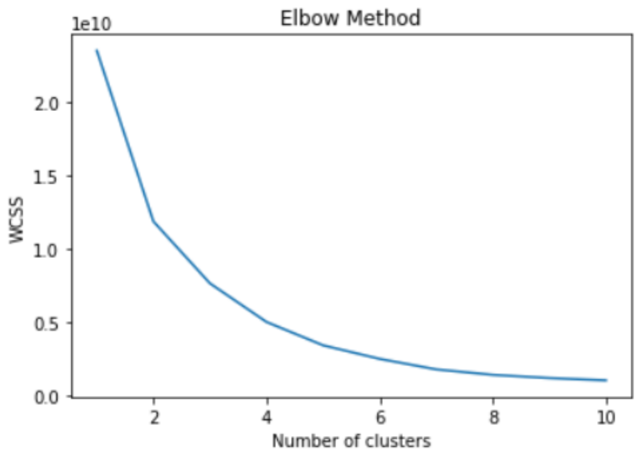
By the graphs, we can that the model is getting fitter properly because both train and validation sets in loss graph are decreasing and the accuracy is increasing.

K means clustering

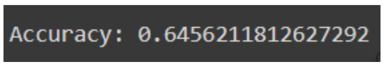
K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem.

The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priority.

```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X_train)
    wcss.append(kmeans.inertia_)
plot.plot(range(1, 11), wcss)
plot.title('Elbow Method')
plot.xlabel('Number of clusters')
plot.ylabel('WCSS')
plot.show()
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=100, n_init=10, random_state=0)
pred_y_kn = kmeans.fit_predict(X_train)
result2 = accuracy_score(Y_train, pred_y_kn)
print("\nAccuracy:",result2)
```



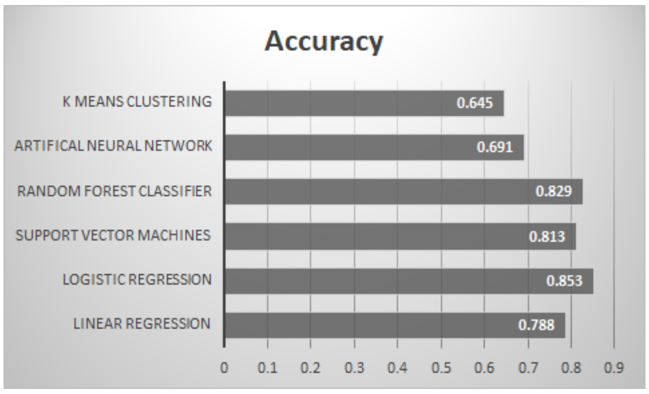
we try to draw the tangent to this curve. this gives the number of clusters which is fed to the model. here, its 4. n_cluster = 4.

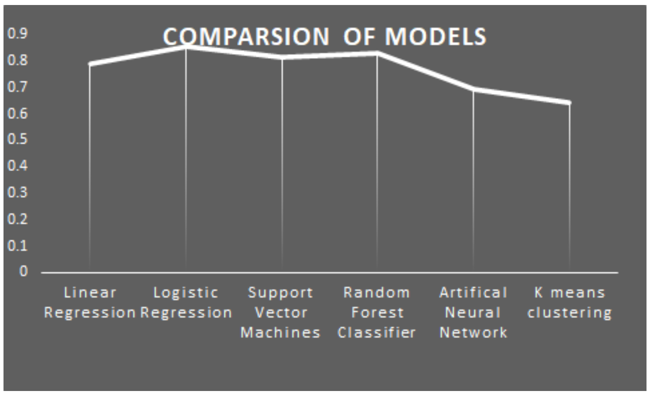


```
Accuracy: 0.6456211812627292
```

Comparative Analysis of models

S.no	Models	Accuracy
1	Linear Regression	0.788
2	Logistic Regression	0.853
3	Support Vector Machines	0.813
4	Random Forest Classifier	0.829
5	Artifical Neural Network	0.691
6	K means clustering	0.645





From above graphs, we can easily say Logistic Regression has done best in classifying the data.

Now, lets try to understand why logistic regressions is better and others dont.

Q1) Why is logistic regression better than SVM?

Ans) In many classification problems, SVM is a better than logistic. Logistic is more suitable than SVM because

- The features are linearly separable.
- The data set is big because of which SVM is failing (while training, it took 2 mins to train)

Q2) The random forest classifier has actually done pretty good. Explain why?

Ans)

- Might be a non linear problem
- Because we used random forest, the data is not overfit.

Q3) Why is ANN and K- means clustering doing bad?

Ans) The possible reasons might be

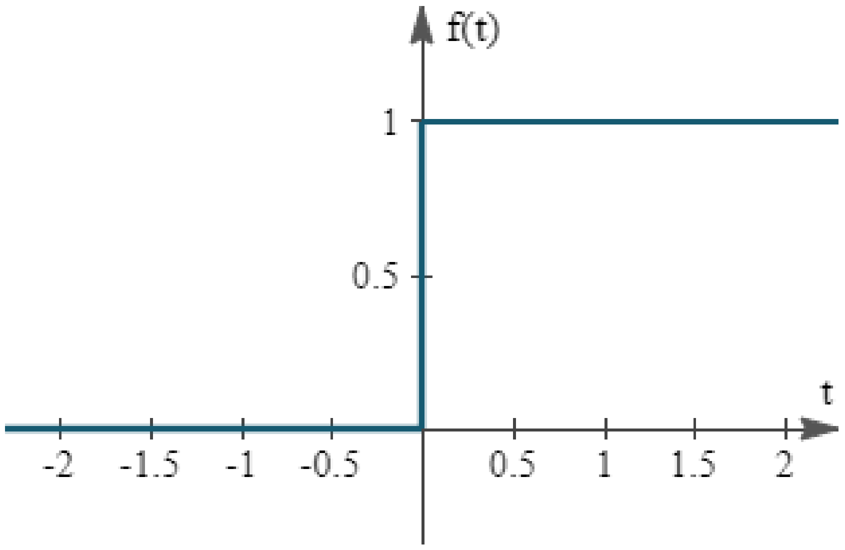
- Ann is like a black box. Hard to explain and the non guaranteed convergence might be a good reason.
- there are outliers. Thats the reason why K means and SVM are going bad

Q4) Why is logistic and linear regression doing the same way?

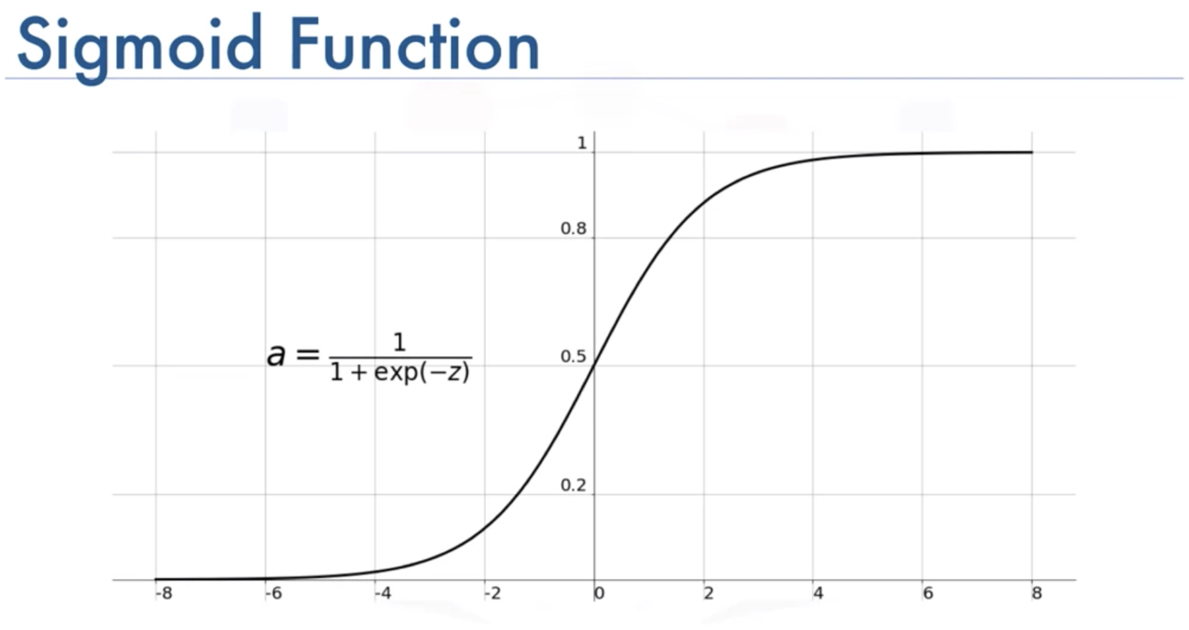
Ans)

The classifier used in linear regression is Unit step function.

Unit Step Function



The classifier used in logistic regression is Sigmoid



They both look alike. that might be the reason why both perform similar way.

Final Model: Logistic Regression

Since this model is doing best, lets predict for unseen data.

```
data1 = pd.read_csv('test.csv')
data1 = data1.drop(columns=['Loan_ID'])
data1_df_encoded = pd.get_dummies(data1,drop_first=True)
data1_df_encoded.head()
imp1 = SimpleImputer(strategy='mean')
imp1_train = imp1.fit(data1_df_encoded)
data1_df_encoded = imp1_train.transform(data1_df_encoded)
predict= LR.predict(data1_df_encoded)
data1['Y/N']= predict
data1.to_csv('predicted.csv')
```

Final conclusion:

Out of all the ML/DL models I have tried, Logistic Regression & DL Model were the one which performed the best with test accuracy of 85%. So if anyone is not married, is graduated, has no dependents, is not self employed, has good income and has good credit history will have good chance of loan application being accepted.