(Mock-Unofficial) Loan Approval Predictor at Velox Finance:

Exploratory Data Analysis + Decision Tree, Random Forest & Logistic Regression Modeling

Problem Statement:

About the Mission

At Velox Fiancne we recieve a loan form with information regaring a laon application. With this information, we will parse through a ML model with previous data sets to determine weather or not the information provided in the loan application is a good loan or not, giving it a score from 0-100 allowing our analysts to prioritise the pipleline.

Approach

Our state-of-the-art Machine Learning (ML) model at Velox Finance is constantly in progress and is revolutionizing the loan prediction process. By harnessing the power of advanced algorithms and big data analysis, our model accurately assesses loan applications and predicts the likelihood of loan approval. By considering various factors such as credit history, income, and property details, our ML model provides valuable insights and helps us make informed decisions regarding loan approvals. With this technology-driven approach, we ensure faster processing times, improved accuracy, and enhanced efficiency, providing our clients with a seamless and transparent loan application experience.

(Mock-Unofficial)- Dataset Description:

The Data is obtaind from Kaggle, an online data service provider. Real client information is kept private under our strict data protection laws. Also note that these are also mock parameters, true parameters are kept internally for intellectual purposes.

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines

Variable	Description
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

In [107…

```
################### Importing Libraries ######################
import pandas as pd

train_df = pd.read_csv(r'C:\Users\05pat\OneDrive\Escritorio\Velox\archive\loan_tra
train_df
```

Out[107]:

	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Inc
0	Male	No	0	Graduate	No	584900	
1	Male	Yes	1	Graduate	No	458300	1508
2	Male	Yes	0	Graduate	Yes	300000	
3	Male	Yes	0	Not Graduate	No	258300	2358
4	Male	No	0	Graduate	No	600000	
...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	290000	
610	Male	Yes	3+	Graduate	No	410600	
611	Male	Yes	1	Graduate	No	807200	240
612	Male	Yes	2	Graduate	No	758300	
613	Female	No	0	Graduate	Yes	458300	

614 rows × 12 columns

Here we aregoing to plot graphs to check if loans were approved or not and in what conditions.

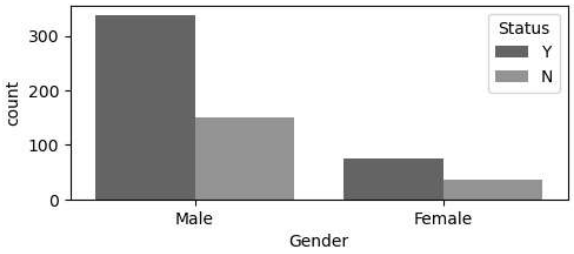
In [108…

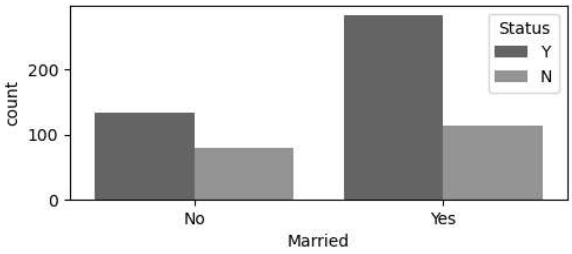
```
############ Count number of Categorical and Numerical Columns ###################
categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Emplo

numerical_columns = ['Applicant_Income', 'Coapplicant_Income', 'Loan_Amount']
```

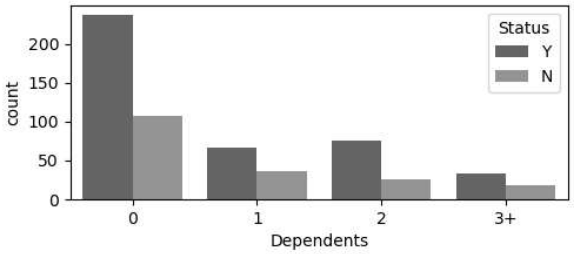
Analyze values assigned to columns and plot

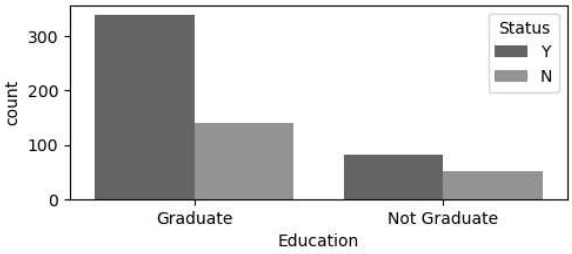
In [109…

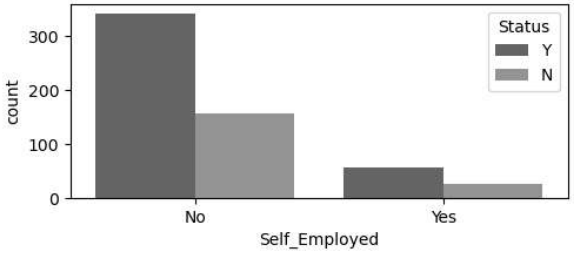
```
### Data Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt


fig,axes = plt.subplots(4,2,figsize=(12,15))
for idx,cat_col in enumerate(categorical_columns):
    row,col = idx//2,idx%2
    sns.countplot(x=cat_col,data=train_df,hue='Status',ax=axes[row,col])


plt.subplots_adjust(hspace=1)
```

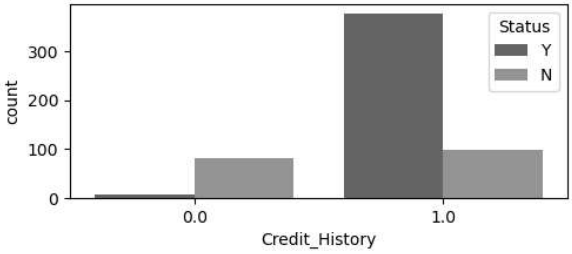

Status
Y
N
count
300
200
100
0
Male
Female
Gender

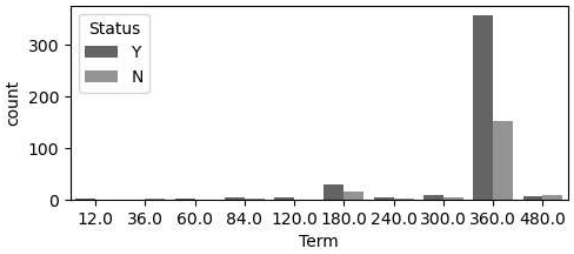

Status
Y
N
count
200
100
0
No
Yes
Married

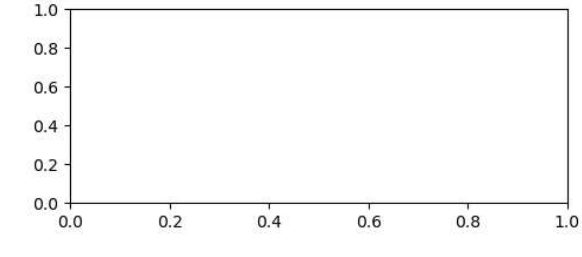

Status
Y
N
count
200
150
100
50
0
0
1
2
3+
Dependents


Status
Y
N
count
300
200
100
0
Graduate
Not Graduate
Education


Status
Y
N
count
300
200
100
0
No
Yes
Self_Employed


Status
Y
N
count
300
200
100
0
0.0
1.0
Credit_History


Status
Y
N
count
300
200
100
0
12.0
36.0
60.0
84.0
120.0
180.0
240.0
300.0
360.0
480.0
Term


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Plots above convey following things about the dataset:

1. Loan Approval Status: About 2/3rd of applicants have been granted loan.
2. Sex: There are more Men than Women (approx. 3x)
3. Martial Status: 2/3rd of the population in the dataset is Marred; Married applicants are more likely to be granted loans.
4. Dependents: Majority of the population have zero dependents and are also likely to accepted for loan.
5. Education: About 5/6th of the population is Graduate and graduates have higher propotion of loan approval
6. Employment: 5/6th of population is not self employed.
7. Property Area: More applicants from Semi-urban and also likely to be granted loans.
8. Applicant with credit history are far more likely to be accepted.
9. Loan Amount Term: Majority of the loans taken are for 360 Months (30 years).

Preprocessing Data:

Input data needs to be pre-processed before we feed it to model. Following things need to be taken care:

1. Encoding Categorical Features.
2. Imputing missing values

In [110…

```
#### Encoding categrical Features into numerical dummies: ##########
train_df_encoded = pd.get_dummies(train_df,drop_first=True)
train_df_encoded.head()
```

Out[110]:

	Applicant_Income	Coapplicant_Income	Loan_Amount	Term	Credit_History	Gender_Male	Mar
0	584900	0.0	15000000	360.0	1.0	1	
1	458300	150800.0	12800000	360.0	1.0	1	
2	300000	0.0	6600000	360.0	1.0	1	
3	258300	235800.0	12000000	360.0	1.0	1	
4	600000	0.0	14100000	360.0	1.0	1	

In [111…

```
########## Split Features and Target Varible ############
X = train_df_encoded.drop(columns='Status_Y')
y = train_df_encoded['Status_Y']

################# Splitting into Train -Test Data #######
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify =y,ran
############### Handling/Imputing Missing values #############
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean')
imp_train = imp.fit(X_train)
X_train = imp_train.transform(X_train)
X_test_imp = imp_train.transform(X_test)
```

Model 1: Decision Tree Classifier

In [112…

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,f1_score


tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train,y_train)
y_pred = tree_clf.predict(X_train)
print("Training Data Set Accuracy: ", accuracy_score(y_train,y_pred))
print("Training Data F1 Score ", f1_score(y_train,y_pred))

print("Validation Mean F1 Score: ",cross_val_score(tree_clf,X_train,y_train,cv=5,s
print("Validation Mean Accuracy: ",cross_val_score(tree_clf,X_train,y_train,cv=5,s
```

```
Training Data Set Accuracy:  1.0
Training Data F1 Score  1.0
Validation Mean F1 Score:  0.6483439356332724
Validation Mean Accuracy:  0.6843331271902701
```

*Note

The F1 score is a metric commonly used in classification tasks to evaluate the performance of a machine learning model. It is the harmonic mean of precision and recall, providing a balanced measure of both metrics.

Overfitting Problem

We can see from above metrics that Training Accuracy > Test Accuracy with default settings of Decision Tree classifier. Hence, model is overfit. We will try some Hyper-parameter tuning and see if it helps.

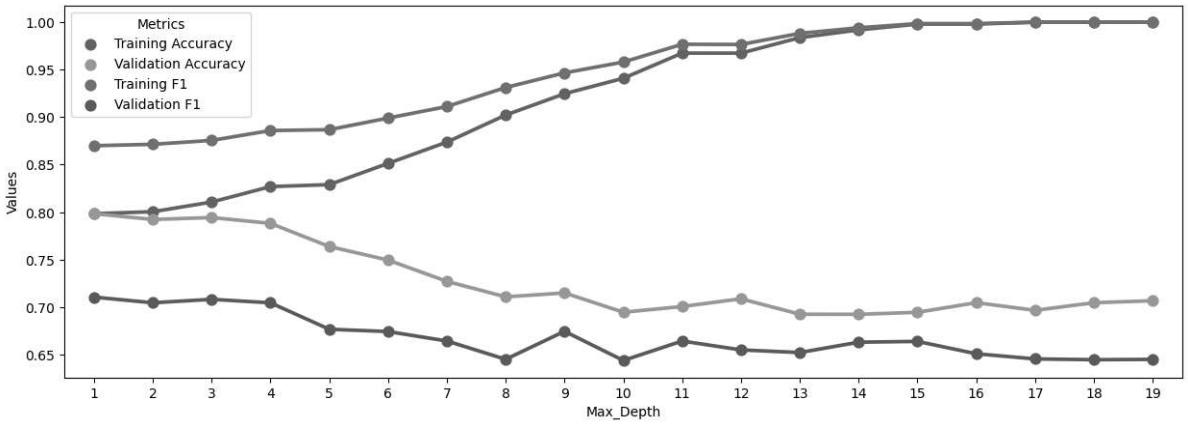
First let's try tuning 'Max_Depth' of tree

In [113…

```
training_accuracy = []
val_accuracy = []
training_f1 = []
val_f1 = []
tree_depths = []

for depth in range(1,20):
    tree_clf = DecisionTreeClassifier(max_depth=depth)
    tree_clf.fit(X_train,y_train)
    y_training_pred = tree_clf.predict(X_train)

    training_acc = accuracy_score(y_train,y_training_pred)
    train_f1 = f1_score(y_train,y_training_pred)
    val_mean_f1 = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro'
    val_mean_accuracy = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='acc

    training_accuracy.append(training_acc)
    val_accuracy.append(val_mean_accuracy)
    training_f1.append(train_f1)
    val_f1.append(val_mean_f1)
    tree_depths.append(depth)


Tuning_Max_depth = {"Training Accuracy": training_accuracy, "Validation Accuracy":
```

```
Tuning_Max_depth_df = pd.DataFrame.from_dict(Tuning_Max_depth)

plot_df = Tuning_Max_depth_df.melt('Max_Depth',var_name='Metrics',value_name="Valu
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Max_Depth", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[113]: <AxesSubplot: xlabel='Max_Depth', ylabel='Values'>



From above graph, we can conclude that keeping 'Max_Depth' = 3 will yield optimum Test accuracy and F1 score Optimum Test Accuracy ~ 0.805; Optimum F1 Score: ~0.7

Visulazing Decision Tree with Max Depth = 3

(Decision tree is the number of neural branches in the ML Model)

From above tree, we could see that some of the leafs have less than 5 samples hence our classifier might overfit. We can sweep hyper-parameter 'min_samples_leaf' to further improve test accuracy by keeping max_depth to 3

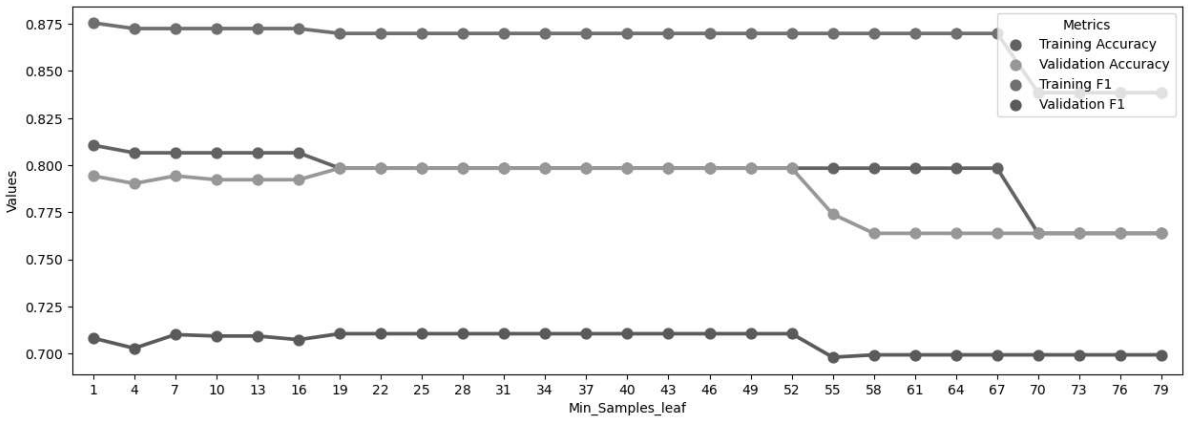
In [114…

```
training_accuracy = []
val_accuracy = []
training_f1 = []
val_f1 = []
min_samples_leaf = []
import numpy as np
for samples_leaf in range(1,80,3): ### Sweeping from 1% samples to 10% samples per
    tree_clf = DecisionTreeClassifier(max_depth=3,min_samples_leaf = samples_leaf)
    tree_clf.fit(X_train,y_train)
    y_training_pred = tree_clf.predict(X_train)

    training_acc = accuracy_score(y_train,y_training_pred)
    train_f1 = f1_score(y_train,y_training_pred)
    val_mean_f1 = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro'
    val_mean_accuracy = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='acc

    training_accuracy.append(training_acc)
    val_accuracy.append(val_mean_accuracy)
    training_f1.append(train_f1)
    val_f1.append(val_mean_f1)
    min_samples_leaf.append(samples_leaf)


Tuning_min_samples_leaf = {"Training Accuracy": training_accuracy, "Validation Acc
Tuning_min_samples_leaf_df = pd.DataFrame.from_dict(Tuning_min_samples_leaf)

plot_df = Tuning_min_samples_leaf_df.melt('Min_Samples_leaf',var_name='Metrics',va
```

```
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Min_Samples_leaf", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[114]:

```
<AxesSubplot: xlabel='Min_Samples_leaf', ylabel='Values'>
```



From above plot, we will choose Min_Samples_leaf to 35 to improve test accuracy.

Let's use this Decision Tree classifier on unseen test data and evaluate **Test Accuracy, F1 Score and Confusion Matrix**

In [115…

```
from sklearn.metrics import confusion_matrix
tree_clf = DecisionTreeClassifier(max_depth=3,min_samples_leaf = 35)
tree_clf.fit(X_train,y_train)
y_pred = tree_clf.predict(X_test_imp)
print("Test Accuracy: ",accuracy_score(y_test,y_pred))
print("Test F1 Score: ",f1_score(y_test,y_pred))
print("Confusion Matrix on Test Data")
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=Tru
```

```
Test Accuracy:  0.8536585365853658
Test F1 Score:  0.903225806451613
Confusion Matrix on Test Data
```

Out[115]:

Predicted	0	1	All
True			
0	21	17	38
1	1	84	85
All	22	101	123

In [116…

```
from sklearn.metrics import confusion_matrix
import pandas as pd

tree_clf = DecisionTreeClassifier(max_depth=6, min_samples_leaf=35)
tree_clf.fit(X_train, y_train)
y_pred = tree_clf.predict(X_test_imp)

print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Test F1 Score:", f1_score(y_test, y_pred))
print(" \nConfusion Matrix on Test Data")
conf_matrix = confusion_matrix(y_test, y_pred)
pd.DataFrame(conf_matrix, index=['True Negative', 'True Positive'], columns=['Pred
```

```
Test Accuracy: 0.8536585365853658
Test F1 Score: 0.903225806451613

Confusion Matrix on Test Data
```

Out[116]:

	Predicted Negative	Predicted Positive
True Negative	21	17
True Positive	1	84

Miss-classifications

It can be seen that majority of the misclassifications are happening because of Loan reject applicants being classified as Accept.

Let's look into Random Forest Classifier if it can reduce mis-classifications

Conclusion

Out of the 3 ML models, the lowest scoring was model 1, decision tree classifier with followign scores:

Test Accuracy: 0.8536585365853658

Test F1 Score: 0.903225806451613

Other models are not presented in this unofficial AI-Engine for intellectual purposes.