In [42]:
```
df['loan_status'] = pd.get_dummies(df['loan_status'])['Charged Off']
```

In [43]:
```
onehot = ['term','verification_status','initial_list_status' , 'application_type']
for col in onehot:
    temp = pd.get_dummies(df[col]).iloc[:,:-1]
    df.drop(columns = [col] , inplace = True)
    df = pd.merge(df, temp, left_index=True, right_index=True)
```

Ordinal Encoding

In [44]:
```
enc = OrdinalEncoder()
df['grade'] = enc.fit_transform(np.array(df['grade']).reshape(-1,1))
```

In [45]:
```
enc1 = OrdinalEncoder()
df['sub_grade'] = enc1.fit_transform(np.array(df['sub_grade']).reshape(-1,1))
```

Target Encoding

In [46]:
```
target = ['emp_title','home_ownership','purpose','address','title']
alist = []
for col in target:
    enc = TargetEncoder()
    df[col] = enc.fit_transform(df[col] , df['loan_status'])
```

In [47]:
```
df.head()
```

Out[47]:

	loan_amnt	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	loan_status	...	pub_rec_bankruptci
0	10000.0	11.44	329.48	1.0	8.0	0.247191	1	0.226620	117000.0	0	...	0
1	8000.0	11.99	265.68	1.0	9.0	0.316979	4	0.169577	65000.0	0	...	0
2	15600.0	10.49	506.97	1.0	7.0	0.181819	1	0.226620	43057.0	0	...	0
3	7200.0	6.49	220.65	0.0	1.0	0.196139	6	0.226620	54000.0	0	...	0
4	24375.0	17.27	609.33	2.0	14.0	0.196139	9	0.169577	55000.0	1	...	0

5 rows × 29 columns

Imbalced data - oversampling the minority

In [48]:
```
df['loan_status'].value_counts()
```

Out[48]:
```
0    318333
1     77672
Name: loan_status, dtype: int64
```

In [49]:
```
oversample = RandomOverSampler(sampling_strategy='minority')
x,y = oversample.fit_resample(df.drop(columns = ['loan_status']), df['loan_status'])
```

In [50]:
```
y.value_counts()
```

Out[50]:
```
0    318333
1    318333
Name: loan_status, dtype: int64
```

Standardizing the Data

In [51]:
```
scaling = StandardScaler()
scaling.fit(x)
scaled = scaling.transform(x)
df_scaled = pd.DataFrame(scaled, columns = x.columns)
```

In [52]:
```
df_scaled.head()
```

Out[52]:

	loan_amnt	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	purpose	...	pub_rec_bankru
0	-0.533747	-0.666588	-0.440616	-0.783166	-0.644126	0.214728	-0.761234	1.066051	0.728663	-0.336095	...	-0.3
1	-0.771181	-0.546418	-0.695803	-0.783166	-0.498364	0.823078	0.426438	-1.028366	-0.106978	0.430700	...	-0.3
2	0.131065	-0.874154	0.269309	-0.783166	-0.789888	-0.355126	-0.761234	1.066051	-0.459602	-1.267692	...	-0.3
3	-0.866154	-1.748118	-0.875914	-1.503971	-1.664459	-0.230296	1.218220	1.066051	-0.283748	-1.267692	...	-0.3
4	1.172803	0.607215	0.678728	-0.062360	0.230446	-0.230296	2.405893	-1.028366	-0.267678	-1.267692	...	-0.3

5 rows × 28 columns

Train - Val - Test Split

In [53]:
```
x_train,x_test,y_train,y_test = train_test_split(df_scaled,y,test_size =.2)
```

In [54]:
```
x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size =.25)
```

Model - 1 Logistic Regression

- 83 percent accuracy is definitely not good

In [55]:
```
model1 = LogisticRegression()
model1.fit(x_train,y_train)
print("Train score : {a}, val socre : {b}".format(a = model1.score(x_train,y_train) , b = model1.score(x_val,y_
```
```
Train score : 0.8295885591323537, val socre : 0.8302953672653594
```

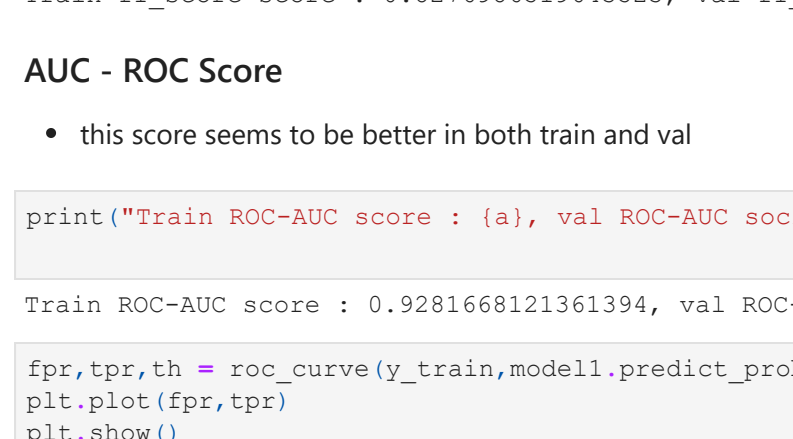
In [56]:
```
print("Train f1_score score : {a}, val f1_score socre : {b}".format(a = f1_score(y_train,model1.predict(x_train
                                                                     b = f1_score(y_val,model1.predict(x_val))))
```
```
Train f1_score score : 0.8276986519643523, val f1_score socre : 0.8289683010803752
```

AUC - ROC Score

- this score seems to be better in both train and val

In [57]:
```
print("Train ROC-AUC score : {a}, val ROC-AUC socre : {b}".format(a = roc_auc_score(y_train,model1.predict_prob
                                                                  b = roc_auc_score(y_val,model1.predict_proba(
```
```
Train ROC-AUC score : 0.9281668121361394, val ROC-AUC socre : 0.9277162314242234
```

In [58]:
```
fpr,tpr,th = roc_curve(y_train,model1.predict_proba(x_train)[:,1])
plt.plot(fpr,tpr)
plt.show()
```



Model - 2 Logistic Regression using elastic net not much improvement in this case

- 83 percent accuracy is definitely not good

In [59]:
```
model2 = LogisticRegression(penalty='elasticnet' , solver = 'saga' , l1_ratio = 0.5)
model2.fit(x_train,y_train)
print("Train score : {a}, val socre : {b}".format(a = model2.score(x_train,y_train) , b = model2.score(x_val,y_
```
```
Train score : 0.8289681386600488, val socre : 0.8290231126259493
```

In [60]:
```
print("Train f1_score score : {a}, val f1_score socre : {b}".format(a = f1_score(y_train,model2.predict(x_train
                                                                     b = f1_score(y_val,model2.predict(x_val))))
```
```
Train f1_score score : 0.8256383705538238, val f1_score socre : 0.826305837674823
```

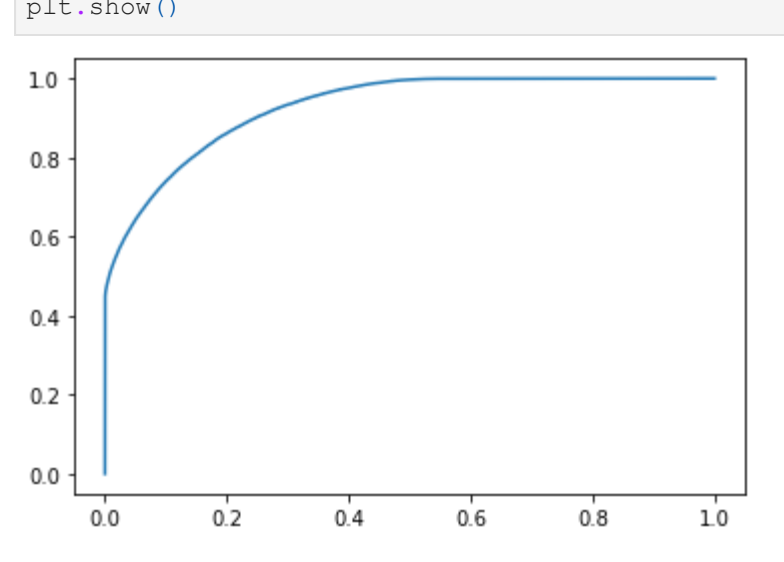
AUC - ROC Score

- this score seems to be better in both train and val

In [61]:
```
print("Train ROC-AUC score : {a}, val ROC-AUC socre : {b}".format(a = roc_auc_score(y_train,model2.predict_prob
                                                                  b = roc_auc_score(y_val,model2.predict_proba(
```
```
Train ROC-AUC score : 0.9281470248667385, val ROC-AUC socre : 0.9277437502442812
```

There doesnt seems to be much difference between train and val score lets plot the ROC- AUC Curve

In [62]:
```
fpr,tpr,th = roc_curve(y_train,model2.predict_proba(x_train)[:,1])
plt.plot(fpr,tpr)
plt.show()
```



Not much difference in both the models , so lets go with model1

Model Interpretation

- Below are the top 5 important feature as per our model

In [63]:
```
coeff = pd.DataFrame({"Coeff" : model1.coef_.reshape(len(df_scaled.columns),) , 'Features' : df_scaled.columns}
coeff.sort_values(by = 'Coeff' , ascending=False).head()
```

Out[63]:

	Coeff	Features
19	13.432821	address
5	0.899045	emp_title
11	0.858744	dti
4	0.549186	sub_grade
10	0.422012	title

Stats Model Summary

- here as well address was given more importance in training the model.
- we can see the 95 percent confidence interval for each features.

In [64]:
```
x_sm = ac(x_train)
model = sm.Logit(y_train,x_sm).fit()
model.summary()
```
```
Optimization terminated successfully.
         Current function value: inf
         Iterations 14
C:\Users\akash\Anaconda3\envs\fiona\lib\site-packages\statsmodels\base\model.py:592: HessianInversionWarning: I
nverting hessian failed, no bse or cov_params available
  warnings.warn('Inverting hessian failed, no bse or cov_params '
C:\Users\akash\Anaconda3\envs\fiona\lib\site-packages\statsmodels\base\model.py:592: HessianInversionWarning: I
nverting hessian failed, no bse or cov_params available
  warnings.warn('Inverting hessian failed, no bse or cov_params '
```

Out[64]:

Logit Regression Results

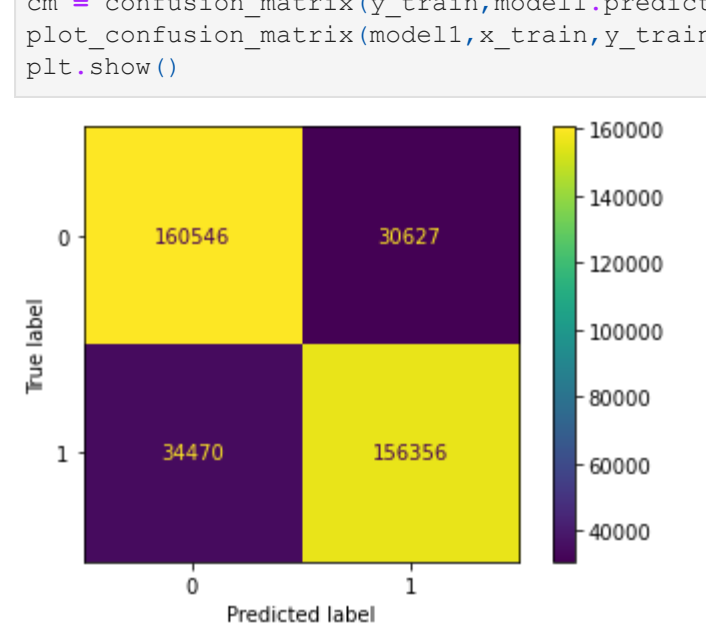
Dep. Variable:	loan_status	**No. Observations:**	381999
Model:	Logit	**Df Residuals:**	381970
Method:	MLE	**Df Model:**	28
Date:	Sun, 20 Nov 2022	**Pseudo R-squ.:**	inf
Time:	18:57:03	**Log-Likelihood:**	-inf
converged:	True	**LL-Null:**	0.0000
Covariance Type:	nonrobust	**LLR p-value:**	1.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	5.2306	0.103	50.899	0.000	5.029	5.432
loan_amnt	0.0483	0.036	1.340	0.180	-0.022	0.119
int_rate	-0.1040	0.024	-4.382	0.000	-0.150	-0.057
installment	0.0230	0.033	0.692	0.489	-0.042	0.088
grade	0.0255	0.024	1.056	0.291	-0.022	0.073
sub_grade	0.5503	0.034	16.349	0.000	0.484	0.616
emp_title	0.8999	0.007	122.258	0.000	0.886	0.914
emp_length	-0.0105	0.005	-2.104	0.035	-0.020	-0.001
home_ownership	0.1334	0.006	23.662	0.000	0.122	0.144
annual_inc	-0.0591	0.008	-7.011	0.000	-0.076	-0.043
purpose	-0.0723	0.005	-13.238	0.000	-0.083	-0.062
title	0.4223	0.007	61.094	0.000	0.409	0.436
dti	0.8612	0.027	31.797	0.000	0.808	0.914
open_acc	0.1456	0.007	19.999	0.000	0.131	0.160
pub_rec	0.0476	0.008	6.104	0.000	0.032	0.063
revol_bal	-0.0570	0.007	-8.725	0.000	-0.070	-0.044
revol_util	0.1215	0.006	21.629	0.000	0.110	0.132
total_acc	-0.1185	0.008	-15.516	0.000	-0.133	-0.104
mort_acc	-0.0583	0.006	-9.487	0.000	-0.070	-0.046
pub_rec_bankruptcies	-0.0268	0.007	-3.802	0.000	-0.041	-0.013
address	14.3221	0.283	50.566	0.000	13.767	14.877
issue_year	-0.0953	0.006	-15.802	0.000	-0.107	-0.083
earliest_cr_line_year	-0.0034	0.005	-0.632	0.527	-0.014	0.007
36 months	-0.2094	0.013	-16.397	0.000	-0.234	-0.184
Not Verified	-0.0557	0.006	-9.330	0.000	-0.067	-0.044
Source Verified	0.0073	0.006	1.286	0.198	-0.004	0.018
f	0.0347	0.005	6.366	0.000	0.024	0.045
DIRECT_PAY	0.0337	0.007	4.589	0.000	0.019	0.048
INDIVIDUAL	0.0553	0.007	7.581	0.000	0.041	0.070

Possibly complete quasi-separation: A fraction 0.26 of observations can be perfectly predicted. This might indicate that there is complete quasi-separation. In this case some parameters will not be identified.

precsion vs recall see how it behaves - it looks to be approximately equal - As it was oversampled due to imbalce in data

In [65]:
```
cm = confusion_matrix(y_train,model1.predict(x_train))
plot_confusion_matrix(model1,x_train,y_train)
plt.show()
```



Threshold - 0.5 gives best results for Accuracy

In [66]:
```
model_thres = LogisticRegression()
model_thres.fit(x_train,y_train)
train_prob = model_thres.predict_proba(x_train)[:,1]
val_prob = model_thres.predict_proba(x_val)[:,1]
train = []
val = []
for i in np.arange(0.1,1,0.1):
    train.append(sum(1*(train_prob > i) == y_train)/len(y_train))
    val.append(sum(1*(val_prob > i) == y_val)/len(y_val))

df = pd.DataFrame(data = [val,train,np.arange(0.1,1,0.1)]).T
df.columns = ['Acc_val','Acc_train','threshold']
df
```

Out[66]:

	Acc_val	Acc_train	threshold
0	0.733965	0.734025	0.1
1	0.773735	0.772772	0.2
2	0.807096	0.806918	0.3
3	0.827555	0.827369	0.4
4	0.830295	0.829589	0.5
5	0.817164	0.819214	0.6
6	0.796125	0.797968	0.7
7	0.767939	0.770601	0.8
8	0.740209	0.742526	0.9

Threshold - 0.5 gives best results for Precision and Recall , But as per the advice from SME whether TN or FP which needs to be handled with care in this case

- FP - will make bank to write off more loans by giving out loans to not eligible candidate.
- FN - will make bank to loss the customer who might pay the loan back depends on what SME or mangement thinks more importance can use the threshold accordingly.

In [67]:
```
train_prob = model_thres.predict_proba(x_train)[:,1]
val_prob = model_thres.predict_proba(x_val)[:,1]
train_p = []
val_p = []
train_r = []
val_r = []
for i in np.arange(0.1,1,0.1):
    train_p.append(precision_score(y_train,1*(train_prob > i)))
    val_p.append(precision_score(y_val,1*(val_prob > i)))
    train_r.append(recall_score(y_train,1*(train_prob > i)))
    val_r.append(recall_score(y_val,1*(val_prob > i)))
df = pd.DataFrame(data = [np.arange(0.1,1,0.1),val_p,train_p,val_r,train_r]).T
df.columns = ['threshold','precision_val','precision_train','recall_val','recall_train']
df
```

Out[67]:

	threshold	precision_val	precision_train	recall_val	recall_train
0	0.1	0.653652	0.652717	0.999343	0.999198
1	0.2	0.692318	0.690452	0.988309	0.988141
2	0.3	0.738459	0.737000	0.953141	0.953879
3	0.4	0.789238	0.787243	0.895467	0.896786
4	0.5	0.838545	0.836204	0.819607	0.819364
5	0.6	0.882990	0.882893	0.732729	0.735681
6	0.7	0.924470	0.924466	0.646524	0.648559
7	0.8	0.962385	0.962804	0.559395	0.562518
8	0.9	0.989111	0.989715	0.487636	0.489671

Tradeoff Questions:

How can we make sure that our model can detect real defaulters and there are less false positives? This is important as we can lose out on an opportunity to finance more individuals and earn interest on it.

- In this case we need less FP - So we have to consider Precision, to have a better precision we can take threshold as .7 becoz anything more than this , will cause to give very poor result in recall.

Since NPA (non-performing asset) is a real problem in this industry, it's important we play safe and shouldn't disburse loans to anyone.

- In this case we need less FN - So we have to consider Recall, to have a better recall we can take threshold as .3 becoz anything less than this , will cause to give very poor result in precision.

Experimenting with different Model

Model- 3 Descision Tree

In [68]:
```
model3 = DecisionTreeClassifier(max_depth = 6)
model3.fit(x_train,y_train)
model3.score(x_val,y_val)
```

Out[68]:
```
0.8162848593844486
```

In [69]:
```
model3 = DecisionTreeClassifier(max_depth = 70)
model3.fit(x_train,y_train)
model3.score(x_val,y_val)
```

Out[69]:
```
0.9226359231306889
```

In [70]:
```
model3 = DecisionTreeClassifier(min_samples_split = 20 , min_samples_leaf = 10)
model3.fit(x_train,y_train)
model3.score(x_val,y_val)
```

Out[70]:
```
0.8663818491671444
```

Model - 4 Random Forest

In [71]:
```
model4 = RandomForestClassifier(min_samples_split = 10 , min_samples_leaf = 5 )
model4.fit(x_train,y_train)
model4.score(x_val,y_val)
```

Out[71]:
```
0.9205940329686727
```