```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sn
from keras.models import Sequential
from keras.layers import Dense,Dropout
import tensorflow as tf
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

Double-click (or enter) to edit

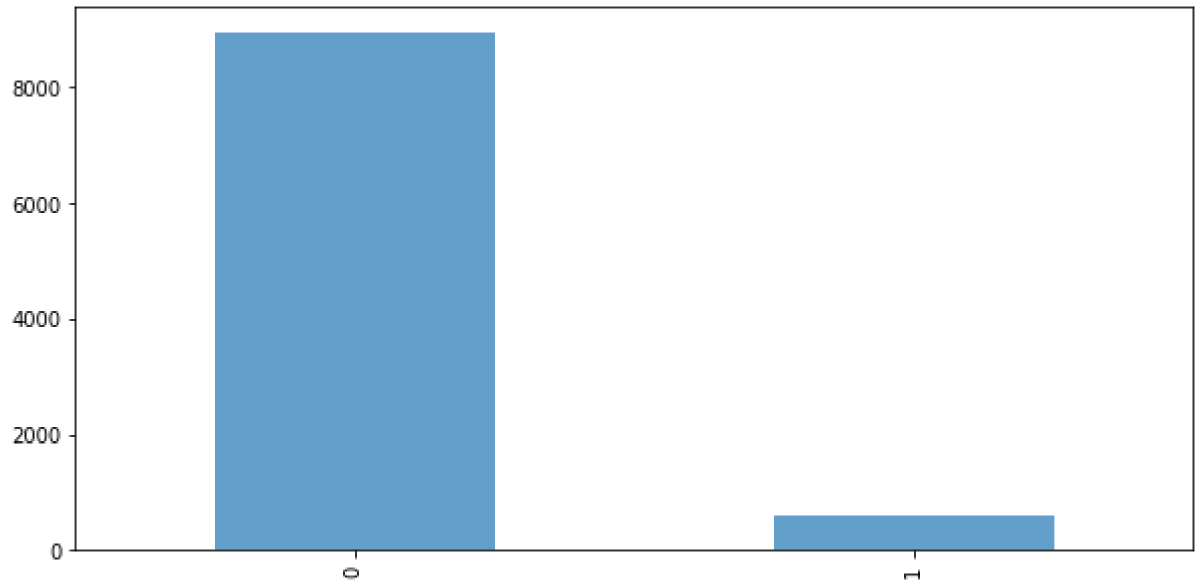
```
data = pd.read_csv('loan_data.csv')
colnames=['A', 'B']
output = pd.read_csv('output.csv',names=colnames,header=None)
```

```
data.head()
```

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5

```
#Y
# It is a highly biased dataset
output['A'].value_counts().plot(kind='bar',figsize=(10,5),alpha=0.7)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdafa6f7d30>
```

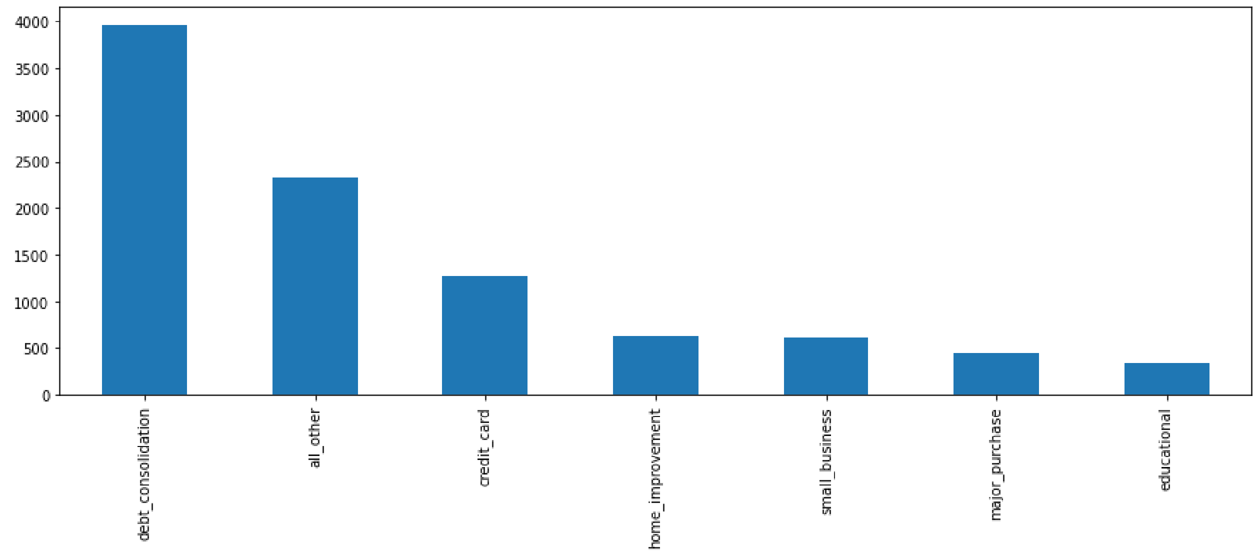


```
data['purpose'].value_counts()
```

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

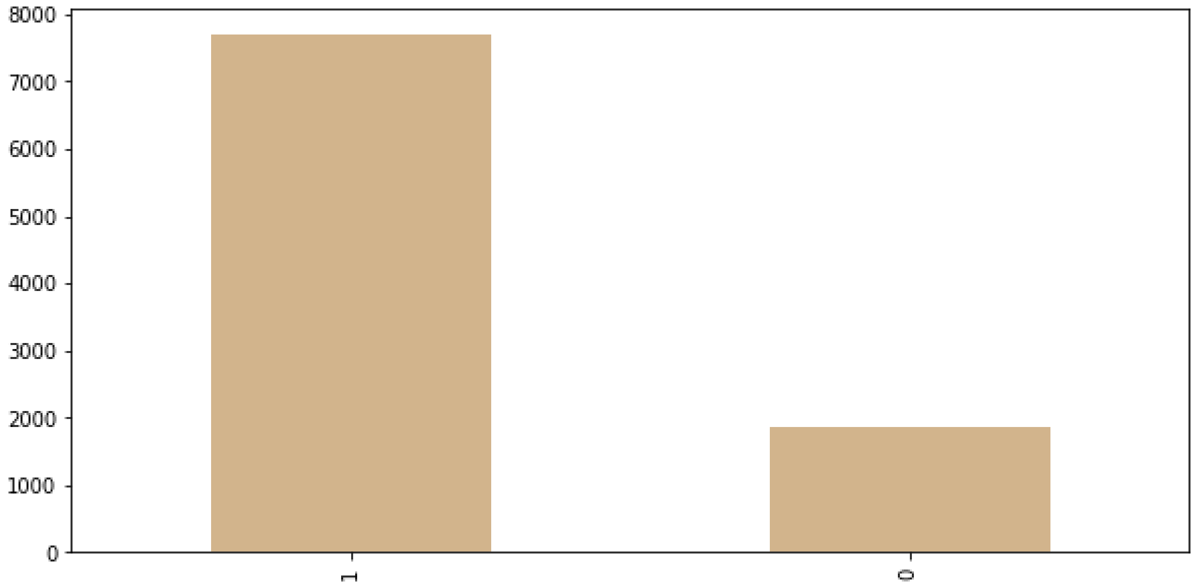
```
data['purpose'].value_counts().plot(kind='bar',figsize=(15,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdaf6aaa940>
```



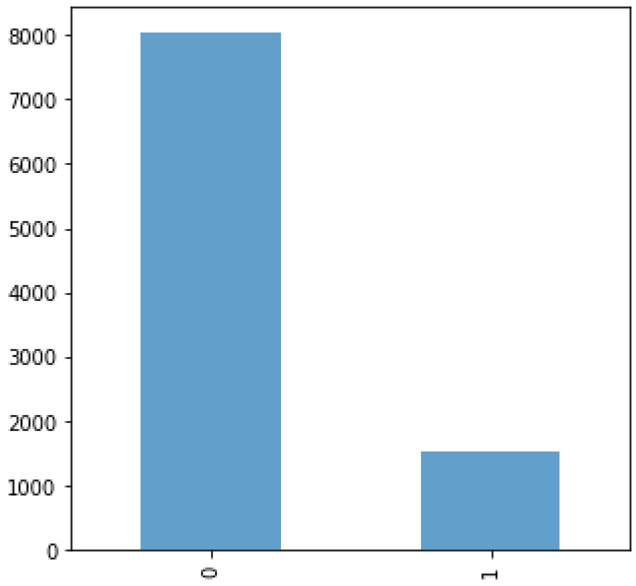
```
data['credit.policy'].value_counts().plot(kind='bar',figsize=(10,5),color='tan')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdaf594a240>
```



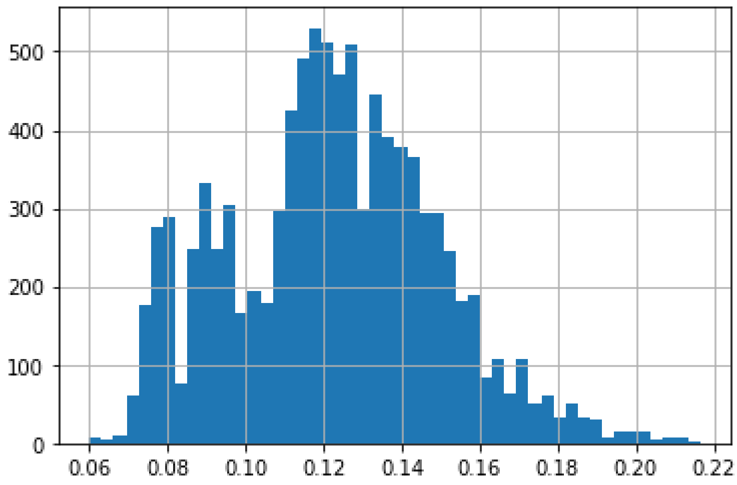
```
data['not.fully.paid'].value_counts().plot(kind='bar',figsize=(5,5),alpha=0.7)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdafc88f1d0>
```



```
data['int.rate'].hist(bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdafa65ebe0>
```

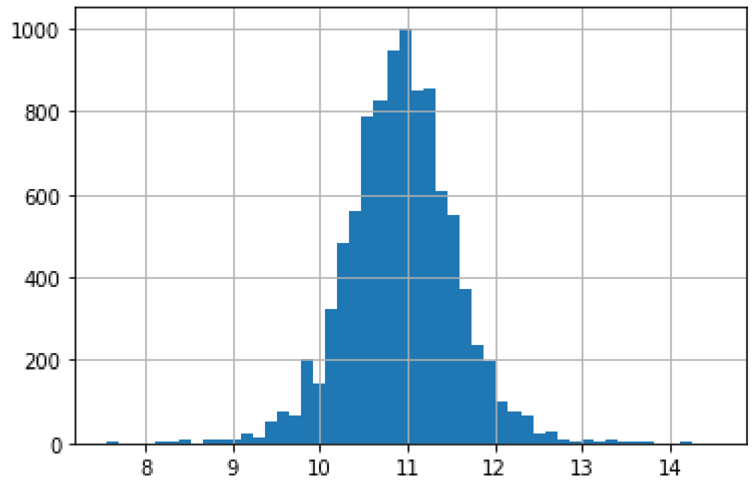


```
data.describe()
```

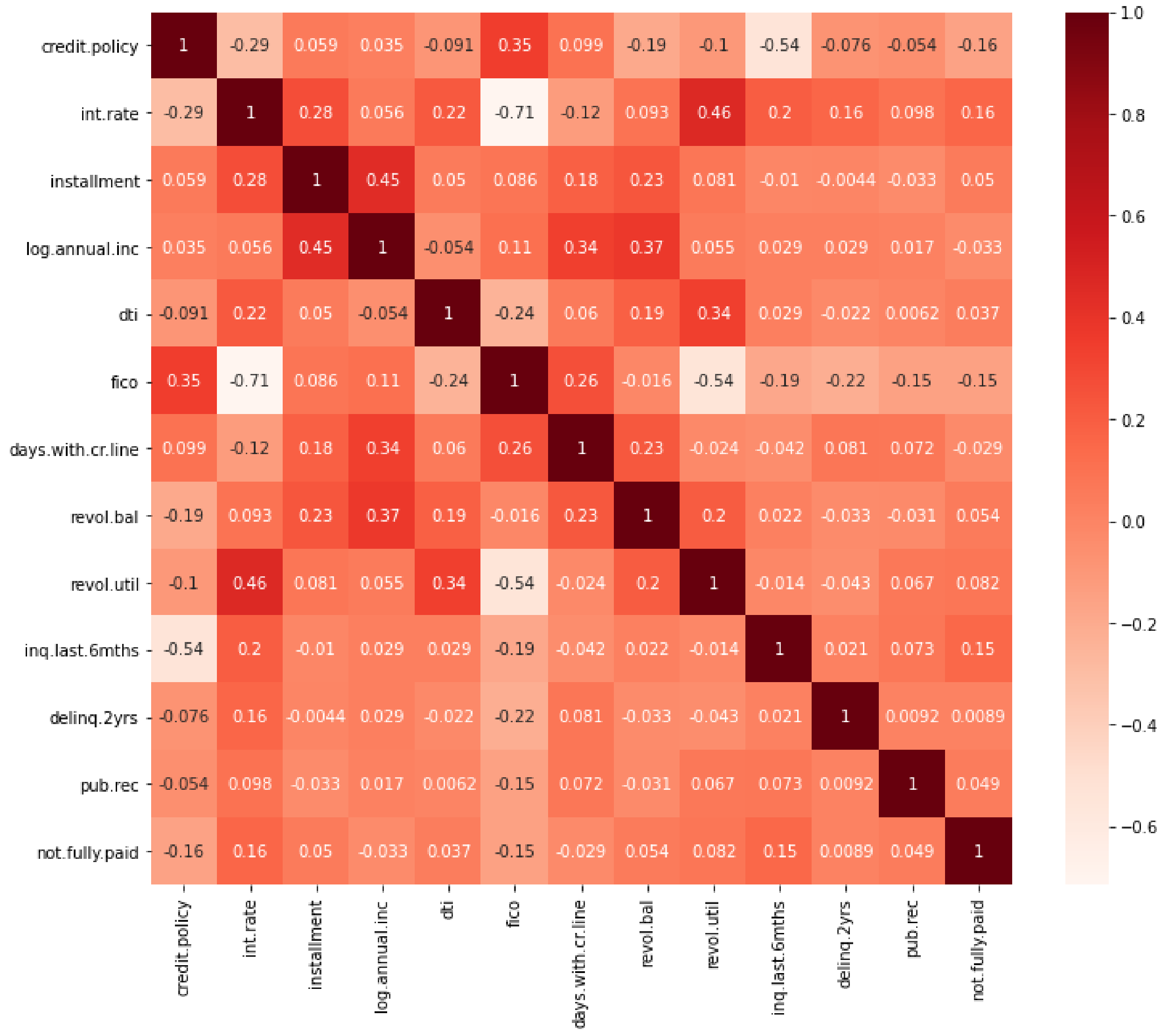
	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.0
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.7
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.0
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.0
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.6
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.3
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.9
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.0

```
data['log.annual.inc'].hist(bins=50)
```

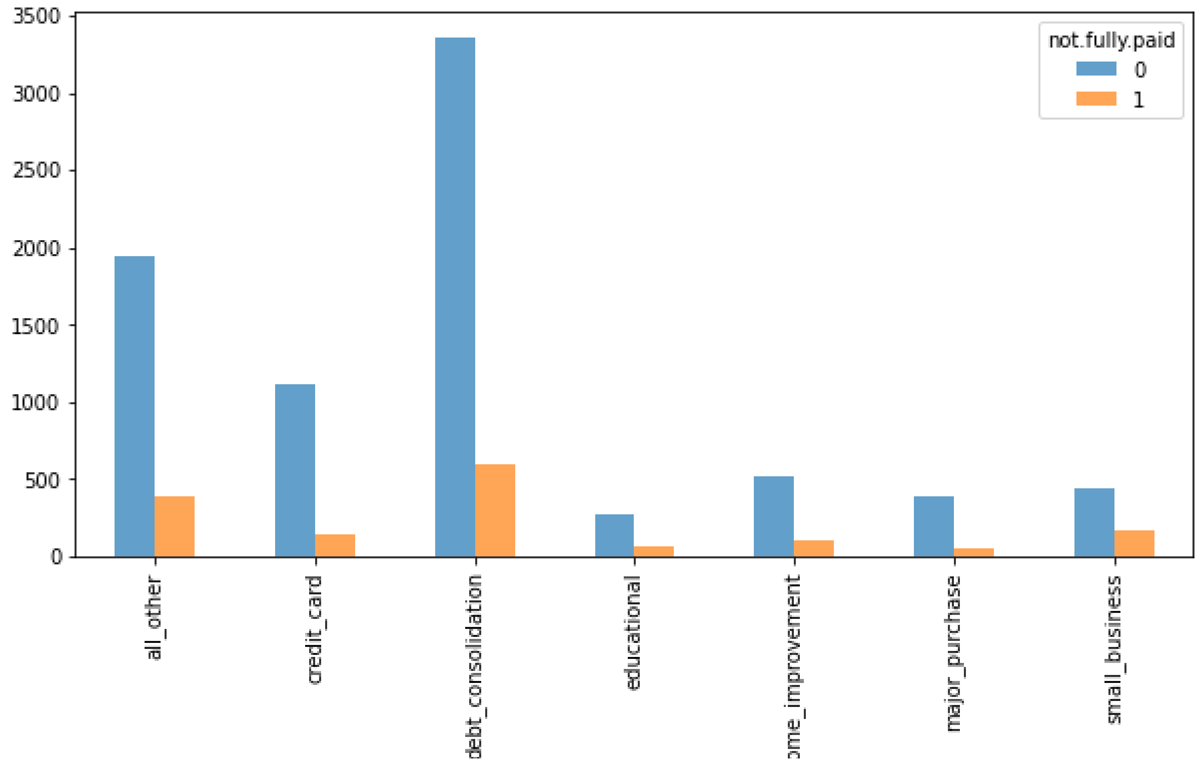
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdafeb31b00>
```



```
#No columns are highly correlated to each other
corr = data.corr()
plt.figure(figsize=(12,10))
sn.heatmap(corr, annot=True, cmap=plt.cm.Reds)
plt.show()
```



```
#Highest loan not paid is under dept_consolidation
data.groupby(["purpose","not.fully.paid"]).size().unstack().plot(kind='bar',stacked=False,legend=True,figsize=(10,5),alpha
<matplotlib.axes._subplots.AxesSubplot at 0x7fdafa6b4ef0>
```



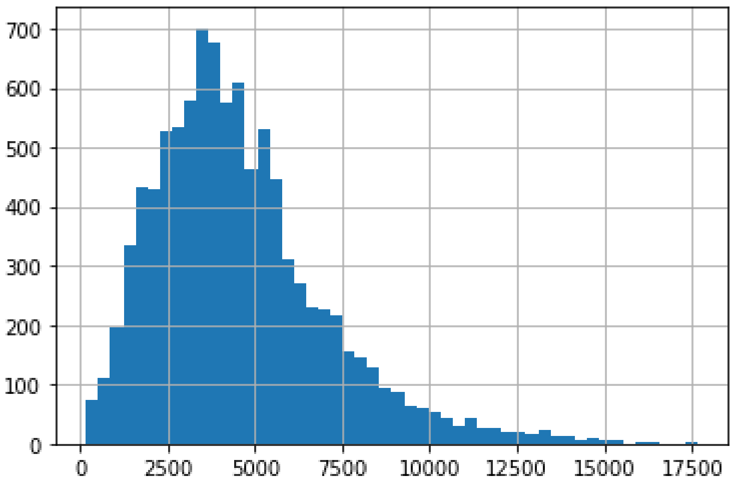
```
data.shape
```

```
(9578, 14)
```

```
data.rename(columns={"int.rate": "int_rate"},inplace=True)
data.rename(columns={"log.annual.inc": "log_annual_inc"},inplace=True)
data.rename(columns={"days.with.cr.line": "days_with_cr_line"},inplace=True)
```

```
data['days_with_cr_line'].hist(bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdb0c004208>
```



```
#Create dummis of column purpose. convert the string
dummy = pd.get_dummies(data['purpose'])
dummy.head()
```

	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purchase	small_business
0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	1	0	0	0	0
4	0	1	0	0	0	0	0

```
data_encoded = pd.concat([data, dummy], axis=1,)
```

```
data_encoded.drop(['purpose'],inplace=True,axis=1)
```

```
X =data_encoded.values
y= output['A'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42,)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train)
scaler = StandardScaler().fit(X_train)

X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)


def create_baseline():

  model = Sequential()

  model.add(Dense(20, activation='relu', input_shape=(20,),))

  #output layer
  model.add(Dense(1, activation='sigmoid'))

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

  return model


model = create_baseline()
model.fit(X_train, y_train,epochs=10, batch_size=5, verbose=1,shuffle=True,validation_split=0.2)
```

```
Epoch 1/10
1027/1027 [==============================] - 1s 1ms/step - loss: 0.1758 - accuracy: 0.9453 - val_loss: 0.0203 - val_accuracy: 1
Epoch 2/10
1027/1027 [==============================] - 1s 1ms/step - loss: 0.0086 - accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy: 1
Epoch 3/10
1027/1027 [==============================] - 1s 1ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0013 - val_accuracy: 1
Epoch 4/10
1027/1027 [==============================] - 1s 1ms/step - loss: 8.0959e-04 - accuracy: 1.0000 - val_loss: 5.4048e-04 - val_acc
Epoch 5/10
1027/1027 [==============================] - 1s 1ms/step - loss: 3.6791e-04 - accuracy: 1.0000 - val_loss: 2.6073e-04 - val_acc
Epoch 6/10
1027/1027 [==============================] - 1s 1ms/step - loss: 1.8266e-04 - accuracy: 1.0000 - val_loss: 1.3352e-04 - val_acc
Epoch 7/10
1027/1027 [==============================] - 1s 1ms/step - loss: 9.4973e-05 - accuracy: 1.0000 - val_loss: 7.0652e-05 - val_acc
Epoch 8/10
1027/1027 [==============================] - 1s 1ms/step - loss: 5.0692e-05 - accuracy: 1.0000 - val_loss: 3.8228e-05 - val_acc
Epoch 9/10
1027/1027 [==============================] - 1s 1ms/step - loss: 2.7577e-05 - accuracy: 1.0000 - val_loss: 2.0953e-05 - val_acc
Epoch 10/10
1027/1027 [==============================] - 1s 1ms/step - loss: 1.5193e-05 - accuracy: 1.0000 - val_loss: 1.1663e-05 - val_acc
<tensorflow.python.keras.callbacks.History at 0x7fdafb7f2f28>
```

```
y_pred = model.predict_classes(X_test)
score = model.evaluate(X_test, y_test,)
print(score)
```

```
99/99 [==============================] - 0s 874us/step - loss: 1.1215e-05 - accuracy: 1.0000
[1.1214959158678539e-05, 1.0]
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[2956    0]
 [   0  205]]
```

```
# I tried with Machine learning models also
```

```
estimator = KerasClassifier(build_fn=create_baseline, epochs=20, batch_size=5, verbose=0,)
kfold = StratifiedKFold(n_splits=10, shuffle=True,)
results = cross_val_score(estimator, X, y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Baseline: 92.90% (3.26%)
```

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
# Veri setini eğitme işlemi:
dtree.fit(X_train, y_train)
```

```
     DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
```

```
predictions = dtree.predict(X_test)
                            min_samples_leaf=1, min_samples_split=2,
```

```
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
[[2956    0]
 [   0  205]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2956
           1       1.00      1.00      1.00       205

    accuracy                           1.00      3161
   macro avg       1.00      1.00      1.00      3161
weighted avg       1.00      1.00      1.00      3161
```

```
from sklearn.model_selection import cross_val_score
print(cross_val_score(dtree, X, y, cv=10))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=600)
rfc.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
r_pred = rfc.predict(X_test)
```

```
print(classification_report(r_pred,y_test))
print('\n')
print(confusion_matrix(r_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2956
           1       1.00      1.00      1.00       205

    accuracy                           1.00      3161
   macro avg       1.00      1.00      1.00      3161
weighted avg       1.00      1.00      1.00      3161



[[2956    0]
 [   0  205]]
```