

# IISPS-INT-2394-EDA-Loan-Status-Prediction

By:- Gorantla Sasi Kumar, Hiren Suresh Ambekar, Shreeramdas Venkata Harendra, Sujay Sunil Pujari

## 1) Importing Libraries

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

## 2) Importing Dataset

```
In [2]: data = pd.read_csv("train.csv")
```

```
In [3]: data.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849	0.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0

```
In [4]: #Data Info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID          614 non-null object
Gender           601 non-null object
Married          611 non-null object
Dependents       599 non-null object
Education        614 non-null object
Self_Employed    582 non-null object
ApplicantIncome  614 non-null int64
CoapplicantIncome 614 non-null float64
LoanAmount       592 non-null float64
Loan_Amount_Term 600 non-null float64
Credit_History  564 non-null float64
Property_Area    614 non-null object
Loan_Status      614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

### 3) Taking care of null values

```
In [5]: #Showing count of missing values in each column  
data.apply(lambda x: sum(x.isnull()),axis=0)
```

```
Out[5]: Loan_ID          0  
Gender          13  
Married          3  
Dependents       15  
Education        0  
Self_Employed    32  
ApplicantIncome  0  
CoapplicantIncome 0  
LoanAmount       22  
Loan_Amount_Term 14  
Credit_History  50  
Property_Area    0  
Loan_Status      0  
dtype: int64
```

```
In [6]: data['Gender'].value_counts()
```

```
Out[6]: Male      489  
Female    112  
Name: Gender, dtype: int64
```

```
In [7]: data.Gender = data.Gender.fillna('Male')
```

```
In [8]: data['Married'].value_counts()
```

```
Out[8]: Yes      398  
No       213  
Name: Married, dtype: int64
```

```
In [9]: data.Married = data.Married.fillna('Yes')
```

```
In [10]: data['Dependents'].value_counts()
```

```
Out[10]: 0      345  
1      102  
2      101  
3+      51  
Name: Dependents, dtype: int64
```

```
In [11]: data.Dependents = data.Dependents.fillna('0')
```

```
In [12]: data['Self_Employed'].value_counts()
```

```
Out[12]: No      500  
Yes       82  
Name: Self_Employed, dtype: int64
```

```
In [13]: data.Self_Employed = data.Self_Employed.fillna('No')
```

```
In [14]: data.LoanAmount = data.LoanAmount.fillna(data.LoanAmount.mean())
```

```
In [15]: data['Loan_Amount_Term'].value_counts()
```

```
Out[15]: 360.0    512
         180.0     44
         480.0     15
         300.0     13
          84.0      4
         240.0      4
         120.0      3
          36.0      2
          60.0      2
          12.0      1
         Name: Loan_Amount_Term, dtype: int64
```

```
In [16]: data.Loan_Amount_Term = data.Loan_Amount_Term.fillna(360.0)
```

```
In [17]: data['Credit_History'].value_counts()
```

```
Out[17]: 1.0     475
         0.0      89
         Name: Credit_History, dtype: int64
```

```
In [18]: data.Credit_History = data.Credit_History.fillna(1.0)
```

```
In [19]: data.apply(lambda x: sum(x.isnull()),axis=0)
```

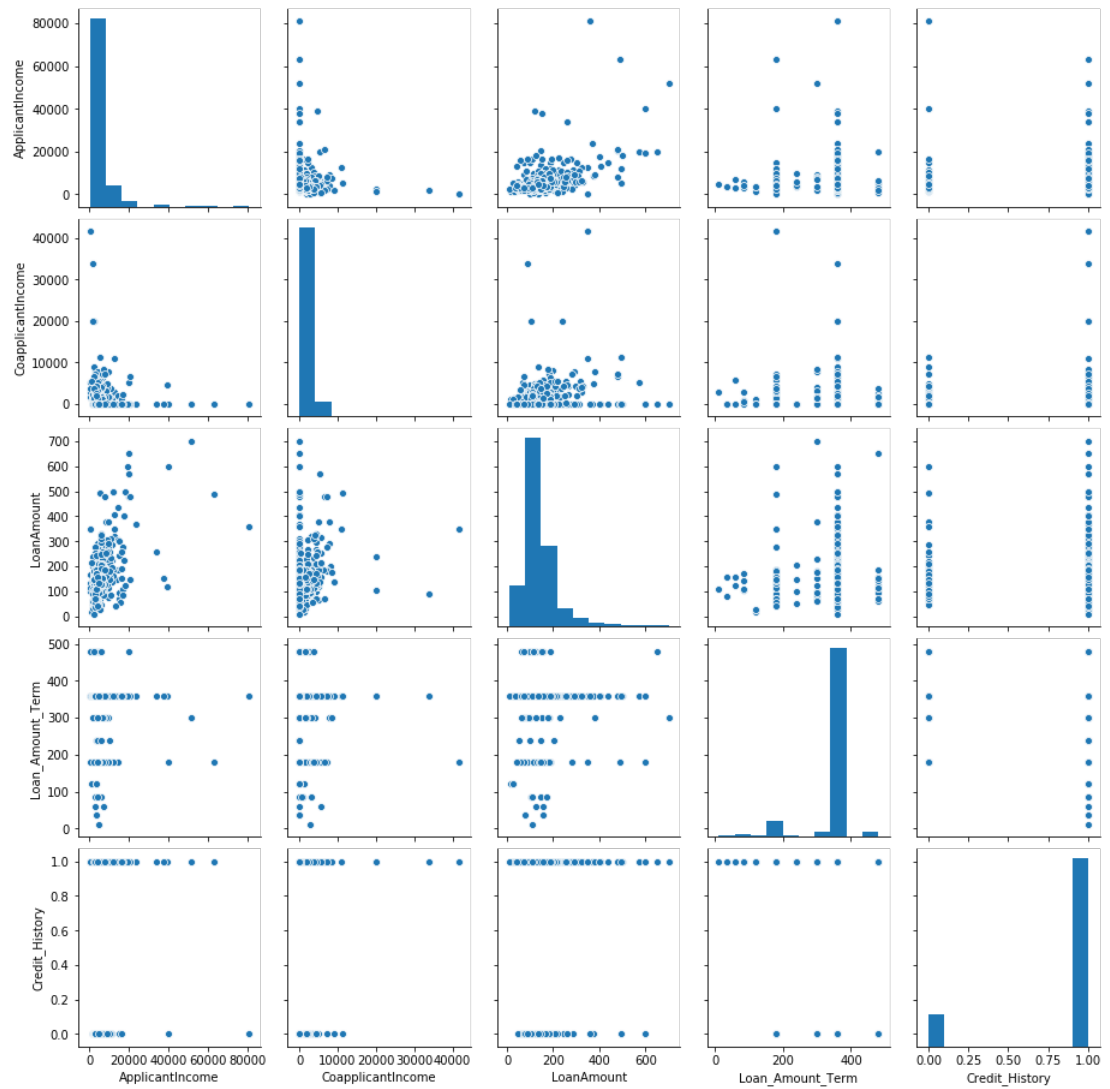
```
Out[19]: Loan_ID           0
         Gender           0
         Married          0
         Dependents       0
         Education        0
         Self_Employed    0
         ApplicantIncome  0
         CoapplicantIncome 0
         LoanAmount       0
         Loan_Amount_Term 0
         Credit_History   0
         Property_Area    0
         Loan_Status      0
         dtype: int64
```

All null values removed

#### 4) Data Visualization

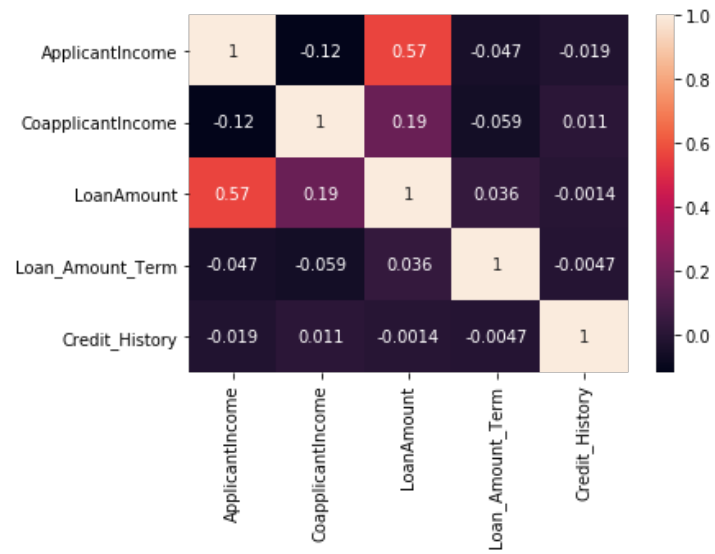
```
In [20]: sns.pairplot(data)
```

```
Out[20]: <seaborn.axisgrid.PairGrid at 0x7f30d115ba50>
```



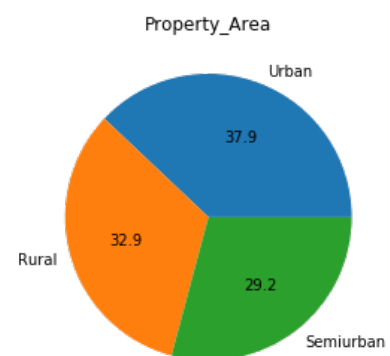
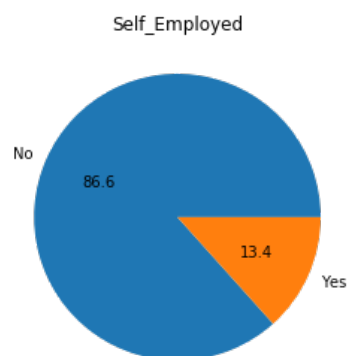
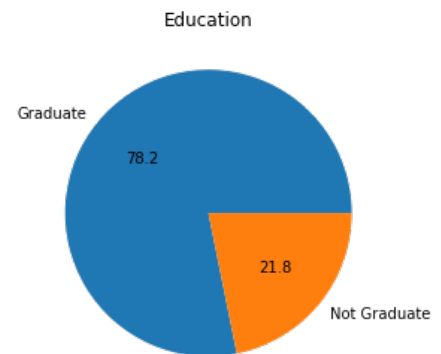
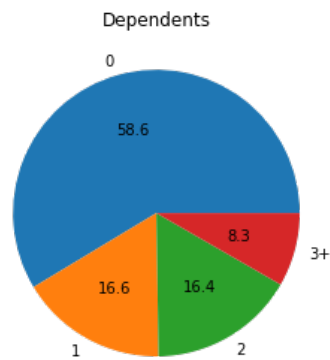
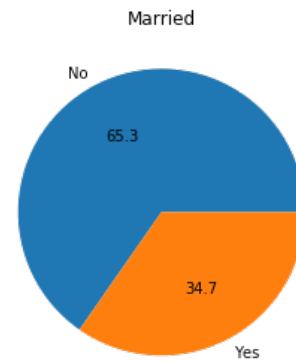
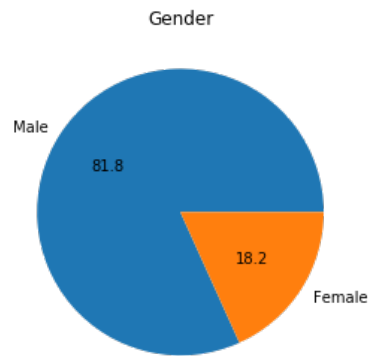
```
In [21]: sns.heatmap(data.corr(), annot = True)
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f30ce2bcb10>
```



```
In [22]: a = data["Gender"].value_counts().to_numpy()
b = data["Married"].value_counts().to_numpy()
c = data["Dependents"].value_counts().to_numpy()
d = data["Education"].value_counts().to_numpy()
e = data["Self_Employed"].value_counts().to_numpy()
f = data["Property_Area"].value_counts().to_numpy()

fig, axs = plt.subplots(3, 2, figsize = (15, 15))
_ = axs[0, 0].pie(a, labels = data["Gender"].unique(), autopct = '%0.1f')
axs[0, 0].set_title('Gender')
_ = axs[0, 1].pie(b, labels = data["Married"].unique(), autopct = '%0.1f')
axs[0, 1].set_title('Married')
_ = axs[1, 0].pie(c, labels = data["Dependents"].unique(), autopct = '%0.1f')
axs[1, 0].set_title('Dependents')
_ = axs[1, 1].pie(d, labels = data["Education"].unique(), autopct = '%0.1f')
axs[1, 1].set_title('Education')
_ = axs[2, 0].pie(e, labels = data["Self_Employed"].unique(), autopct = '%0.1f')
axs[2, 0].set_title('Self_Employed')
_ = axs[2, 1].pie(f, labels = data["Property_Area"].unique(), autopct = '%0.1f')
_ = axs[2, 1].set_title('Property_Area')
```



In [23]: `data.head()`

Out[23]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849	0.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0

### 5) Splitting in X and Y

```
In [24]: # Splitting traing data into X and Y
X = data.iloc[:, 1: 12].values
y = data.iloc[:, 12].values
```

```
In [25]: X
```

```
Out[25]: array([[ 'Male', 'No', '0', ..., 360.0, 1.0, 'Urban'],
                [ 'Male', 'Yes', '1', ..., 360.0, 1.0, 'Rural'],
                [ 'Male', 'Yes', '0', ..., 360.0, 1.0, 'Urban'],
                ...,
                [ 'Male', 'Yes', '1', ..., 360.0, 1.0, 'Urban'],
                [ 'Male', 'Yes', '2', ..., 360.0, 1.0, 'Urban'],
                [ 'Female', 'No', '0', ..., 360.0, 0.0, 'Semiurban']], dtype=object)
```



[illegible]

## 6) Label Encoding

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
In [28]: for i in range(0, 5):  
         X[:,i] = le.fit_transform(X[:,i])  
  
         X[:,10] = le.fit_transform(X[:,10])  
         y = le.fit_transform(y)
```

```
In [29]: X.shape
```

```
Out[29]: (614, 11)
```

## 7) OneHot Encoding

```
In [30]: from sklearn.preprocessing import OneHotEncoder  
         one = OneHotEncoder()  
         z = one.fit_transform(X[:,10:11]).toarray()  
         X = np.delete(X, 10, axis = 1)  
         X = np.concatenate((z,X), axis = 1)
```

```
In [31]: X.shape
```

```
Out[31]: (614, 13)
```

## 8) Splitting into train and test

```
In [32]: # Splitting the dataset into the Training set and Test set  
         from sklearn.model_selection import train_test_split  
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

```
In [33]: X_train.shape
```

```
Out[33]: (409, 13)
```

```
In [34]: X_test.shape
```

```
Out[34]: (205, 13)
```

```
In [35]: y_train.shape
```

```
Out[35]: (409,)
```

```
In [36]: y_test.shape
```

```
Out[36]: (205,)
```

## 9) Feature Scaling

```
In [37]: # Feature Scaling  
         from sklearn.preprocessing import StandardScaler  
         sc = StandardScaler()  
         X_train = sc.fit_transform(X_train)  
         X_test = sc.fit_transform(X_test)
```

**Applying Principal Component analysis (PCA)**

Using to emphasize variation and bring out strong patterns in the dataset and make data easy to explore and visualize further in training and testing

```
In [38]: # Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.fit_transform(X_test)
explained_variance = pca.explained_variance_ratio_
```

## Done

**Decision Tree Classification**

```
In [39]: # Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
Out[39]: DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
In [40]: # Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

```
Out[40]: array([1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
                1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
                1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
                0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
                1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
                0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
                0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 1, 1, 0])
```

```
In [41]: # Measuring Accuracy
from sklearn import metrics
print('The accuracy of Decision Tree Classifier is: ', metrics.accuracy_score(y_test, y_pred))
```

```
The accuracy of Decision Tree Classifier is: 0.5121951219512195
```

```
In [42]: # Making confusion matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[40 20]
 [80 65]]
```

```
In [43]: import sklearn.metrics as metrics
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)
```

```
In [44]: threshold
```

```
Out[44]: array([2, 1, 0])
```

```
In [45]: fpr
```

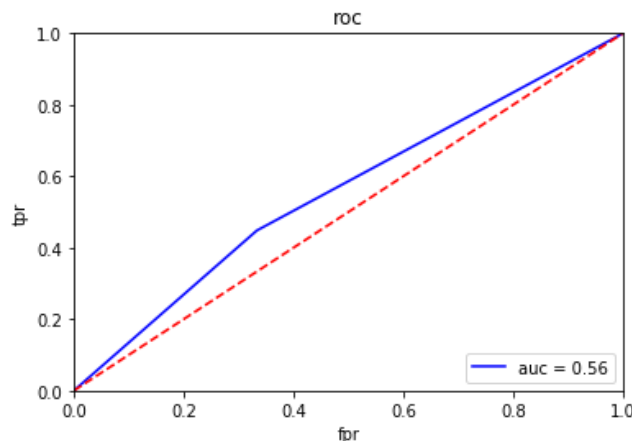
```
Out[45]: array([0.          , 0.33333333, 1.          ])
```

```
In [46]: tpr
```

```
Out[46]: array([0.          , 0.44827586, 1.          ])
```

```
In [47]: plt.title("roc")
plt.plot(fpr, tpr, 'b', label = 'auc = %0.2f'%roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], 'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('tpr')
plt.xlabel('fpr')
```

```
Out[47]: Text(0.5, 0, 'fpr')
```



## Random Forest

```
In [48]: # Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', r
andom_state = 0)
classifier.fit(X_train, y_train)
```

```
Out[48]: RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

```
In [49]: # Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

```
Out[49]: array([1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
                1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
                1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
                0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
                1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
                0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
                0, 0, 0, 1, 1, 1, 0])
```

```
In [50]: # Measuring Accuracy
from sklearn import metrics
print('The accuracy of Random Forest Classification is: ', metrics.accuracy_score(y_pred, y_test))
```

The accuracy of Random Forest Classification is: 0.5609756097560976

```
In [51]: # Making confusion matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[35 25]
 [65 80]]
```

```
In [52]: import sklearn.metrics as metrics
fpr,tpr,threshold = metrics.roc_curve(y_test, y_pred)
roc_auc = metrics.auc(fpr, tpr)
```

```
In [53]: threshold
```

```
Out[53]: array([2, 1, 0])
```

```
In [54]: fpr
```

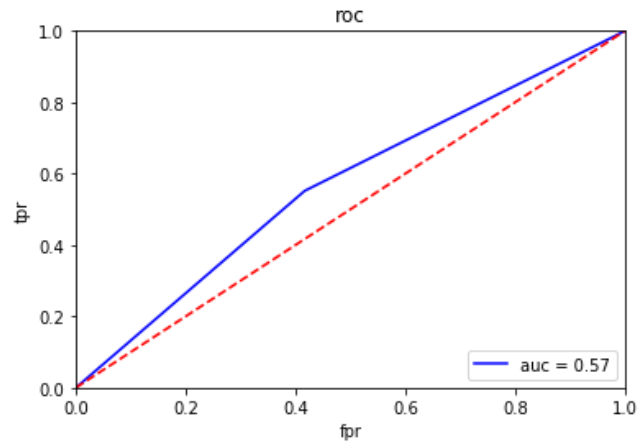
```
Out[54]: array([0.          , 0.41666667, 1.          ])
```

```
In [55]: tpr
```

```
Out[55]: array([0.          , 0.55172414, 1.          ])
```

```
In [56]: plt.title("roc")
plt.plot(fpr, tpr, 'b', label = 'auc = %0.2f'%roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0,1],[0,1], 'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('tpr')
plt.xlabel('fpr')
```

Out[56]: Text(0.5, 0, 'fpr')



## Results:

The accuracy of Decision Tree Classifier is: 51.22 % and auc value = 0.56

The accuracy of Random Forest Classification is: 56.1 % and auc value = 0.57

In [ ]: