Loan Approval Prediction

```
#####
```

About Dataset -

The loan approval dataset is a collection of financial records and associated information used to determine the eligibility of individuals or organizations for obtaining loans from a lending institution. It includes various factors such as cibil score, income, employment status, loan term, loan amount, assets value, and loan status. This dataset is commonly used in machine learning and data analysis to develop models and algorithms that predict the likelihood of loan approval based on the given features.

Importing Libraries

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


from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
```

Loading Dataset

In [2]:
```
df=pd.read_csv(r'C:\Users\user\Downloads\loan_approval_dataset_1.csv')
df
```

2	3	3	Graduate	No	9100000	29700000
3	4	3	Graduate	No	8200000	30700000
4	5	5	Not Graduate	Yes	9800000	24200000
...	...	...	...	...	...	...
4264	4265	5	Graduate	Yes	1000000	2300000
4265	4266	0	Not Graduate	Yes	3300000	11300000
4266	4267	2	Not Graduate	No	6500000	23900000
4267	4268	1	Not Graduate	No	4100000	12800000
4268	4269	1	Graduate	No	9200000	29700000

4269 rows × 13 columns

In [ ]:
```
```

Check null values

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
loan_id                     0
 no_of_dependents           0
 education                  0
 self_employed              0
 income_annum               0
 loan_amount                0
 loan_term                  0
 cibil_score                0
 residential_assets_value   0
 commercial_assets_value    0
 luxury_assets_value        0
 bank_asset_value           0
 loan_status                0
dtype: int64
```

Dataframe info

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   loan_id                   4269 non-null   int64 
 1    no_of_dependents         4269 non-null   int64 
 2    education                4269 non-null   object
 3    self_employed            4269 non-null   object
 4    income_annum             4269 non-null   int64 
 5    loan_amount              4269 non-null   int64 
 6    loan_term                4269 non-null   int64 
 7    cibil_score              4269 non-null   int64 
 8    residential_assets_value 4269 non-null   int64 
 9    commercial_assets_value  4269 non-null   int64 
 10   luxury_assets_value      4269 non-null   int64 
 11   bank_asset_value         4269 non-null   int64 
 12   loan_status              4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['loan_id', ' no_of_dependents', ' education', ' self_employed',
       ' income_annum', ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value', ' loan_status'],
      dtype='object')
```

In [6]:
```
# df.dropna(inplace=True)
```

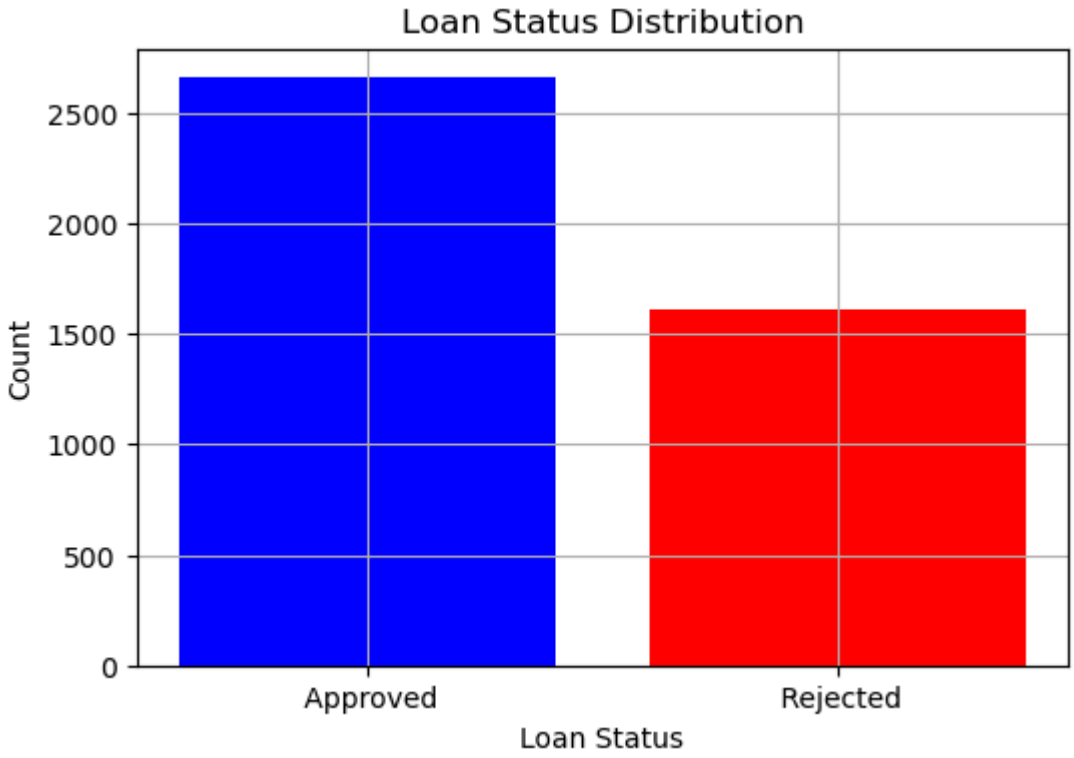
In [7]:
```
#df['loan_status'] = df['loan_status'].astype(int)
#df['loan_status'] = pd.to_numeric(df['loan_status'], errors='coerce')
```

Bar Chart on loan Status

In [8]:
```
df[' loan_status'].value_counts()

loan_status_counts = df[' loan_status'].value_counts()
```

In [9]:
```
plt.figure(figsize=(6, 4))
plt.bar(loan_status_counts.index, loan_status_counts.values, color=['blue',
plt.title('Loan Status Distribution')
plt.xlabel('Loan Status')
plt.ylabel('Count')
plt.grid()
plt.show()
```



From above we observed that our target column has more than 2500 counts for Approvals

And more than 1500 Application got Rejected

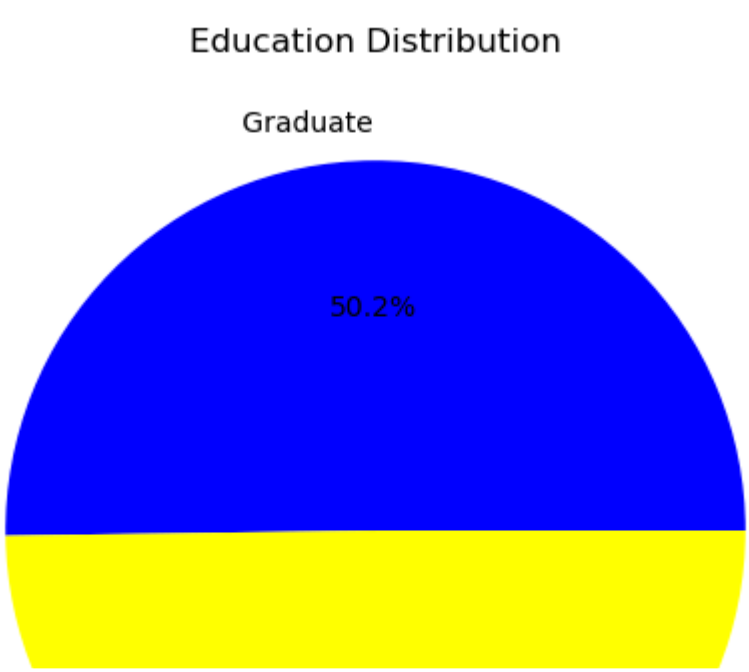
Pie chart

In [11]:
```
df[' education'].value_counts()
```

Out[11]:
```
 Graduate        2144
 Not Graduate    2125
Name:  education, dtype: int64
```

In [14]:
```
education_counts = df[' education'].value_counts()
```

In [15]:
```
plt.figure(figsize=(6, 6))
plt.pie(education_counts, labels=education_counts.index, autopct='%1.1f%%',
plt.title('Education Distribution')
plt.show()
```



From above we can observed that in Education column Graduate's have 50.2%,

49.8% Non graduate so there is risk to get loan approved

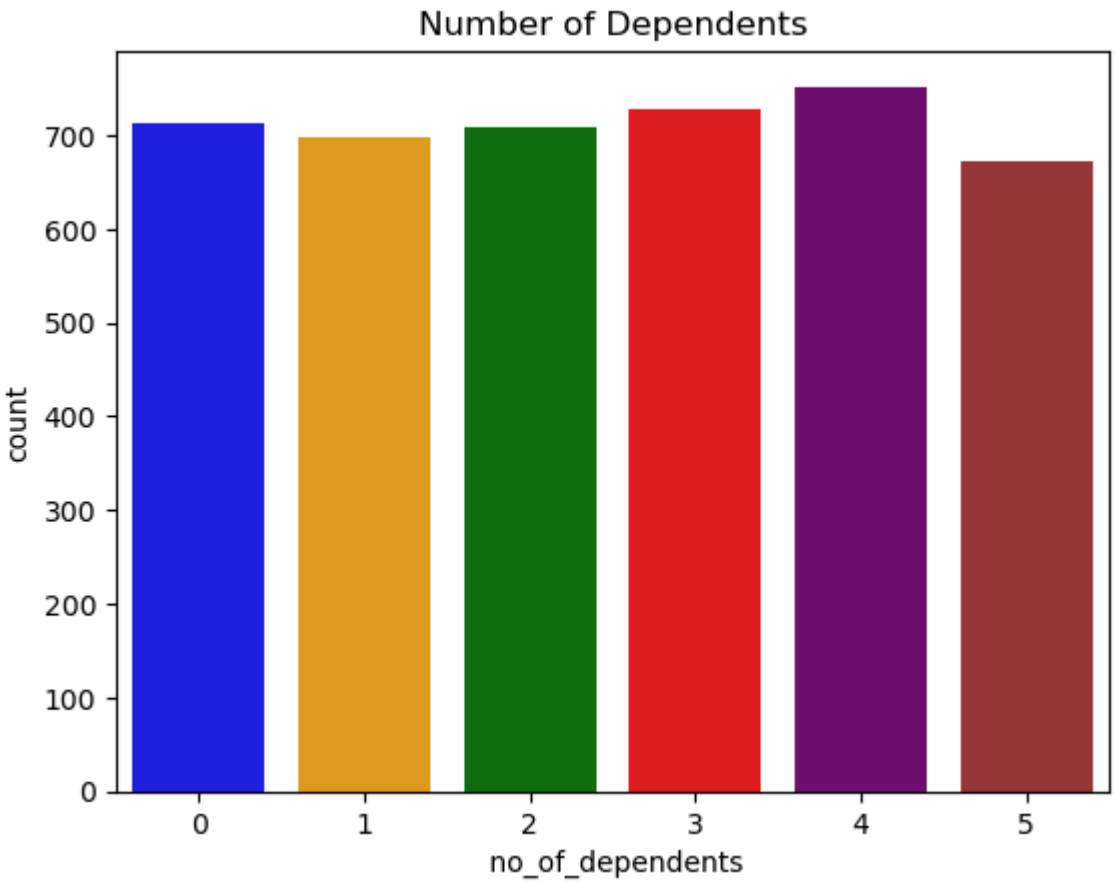
In [ ]:
```
```

Countplot

In [16]:
```
palette = ['blue', 'orange', 'green', 'red', 'purple', 'brown']
```

In [17]:
```
sns.countplot( data = df,x = ' no_of_dependents', palette=palette)
plt.title('Number of Dependents')
```

Out[17]:
```
Text(0.5, 1.0, 'Number of Dependents')
```



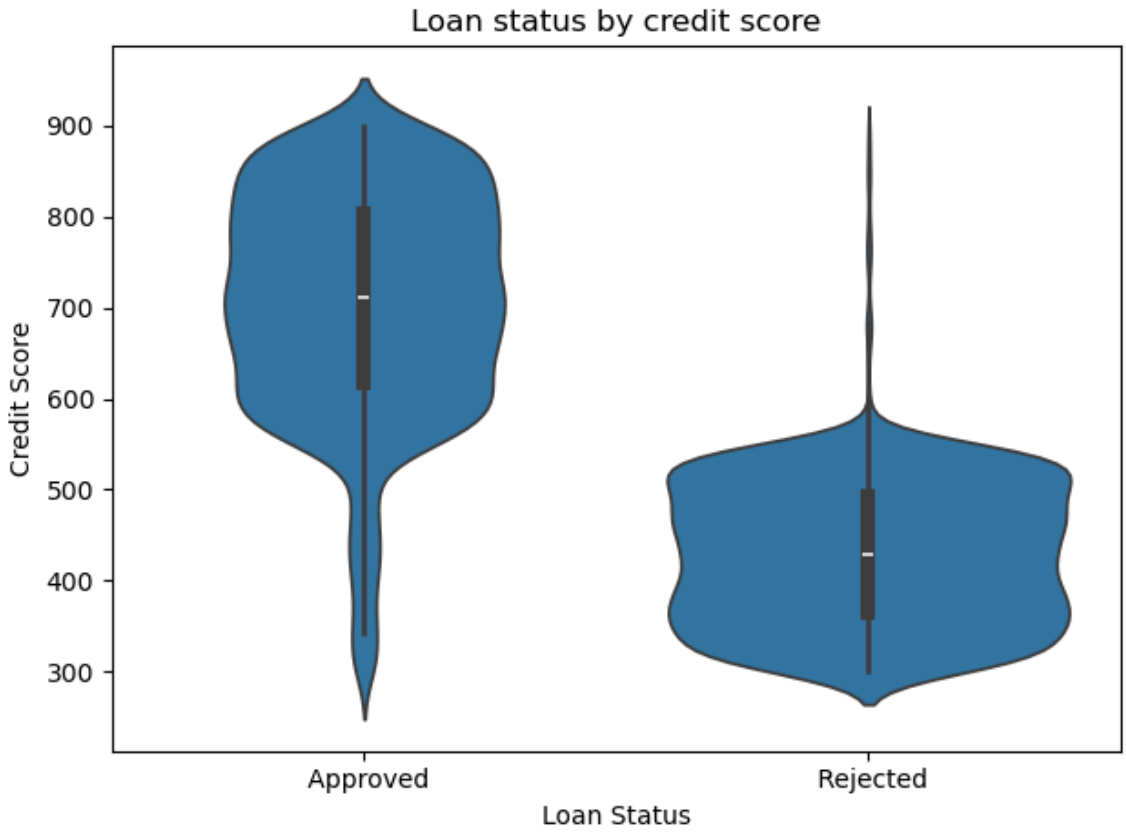
From above we can clearly observed that who had number of dependents 4 have max counts

In [ ]:
```
```

Comparing Cibil score with loan status

$CIBIL$	$Meaning$
$300 - 549$	$Poor$
$550 - 649$	$Fair$
$650 - 749$	$Good$
$750 - 799$	$VeryGood$
$800 - 900$	$Excellent$

In [18]:
```
sns.violinplot(x=' loan_status', y=' cibil_score', data=df)
plt.xlabel('Loan Status')
plt.ylabel('Credit Score')
plt.title('Loan status by credit score')
plt.tight_layout()
plt.show()
```



From above we observed that cibil score falls below 600 will get higher chance of loan rejections

In [ ]:
```
```

In [ ]:
```
```

Remove leading spaces from column names

In [19]:
```
df.rename(columns=lambda x: x.strip(), inplace=True)

# Display the updated DataFrame
print(df.columns)
```

```
Index(['loan_id', 'no_of_dependents', 'education', 'self_employed',
       'income_annum', 'loan_amount', 'loan_term', 'cibil_score',
       'residential_assets_value', 'commercial_assets_value',
       'luxury_assets_value', 'bank_asset_value', 'loan_status'],
      dtype='object')
```

In [ ]:

Data Pre-Processing

In [20]:
```
#Converted the categorical columns to numerical columns
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder instance
le = LabelEncoder()

# Apply label encoding to the 'education' column
df['education'] = le.fit_transform(df['education'])

# Apply label encoding to the 'self_employed' column
df['self_employed'] = le.fit_transform(df['self_employed'])

df['loan_status'] = le.fit_transform(df['loan_status'])


# Display the updated DataFrame with encoded columns
print(df[['education', 'self_employed','loan_status']])
```

```
      education  self_employed  loan_status
0             0              0            0
1             1              1            1
2             0              0            1
3             0              0            1
4             1              1            1
...         ...            ...          ...
4264          0              1            1
4265          1              1            0
4266          1              0            1
4267          1              0            0
4268          0              0            0

[4269 rows x 3 columns]
```

Because Machine doesnt understand categorical data so we converted categorical to

In [ ]:

In [ ]:

separating the data and label

In [21]:
```
# separating the data and label
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [22]:
```
X
```

Out[22]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount
0	1	2	0	0	9600000	29900000
1	2	0	1	1	4100000	12200000
2	3	3	0	0	9100000	29700000
3	4	3	0	0	8200000	30700000
4	5	5	1	1	9800000	24200000
...	...	...	...	...	...	...
4264	4265	5	0	1	1000000	2300000
4265	4266	0	1	1	3300000	11300000
4266	4267	2	1	0	6500000	23900000
4267	4268	1	1	0	4100000	12800000
4268	4269	1	0	0	9200000	29700000

In [23]:
```
y
```

Out[23]:
```
0       0
1       1
2       1
3       1
4       1
       ..
4264    1
4265    0
4266    1
4267    0
4268    0
Name: loan_status, Length: 4269, dtype: int32
```

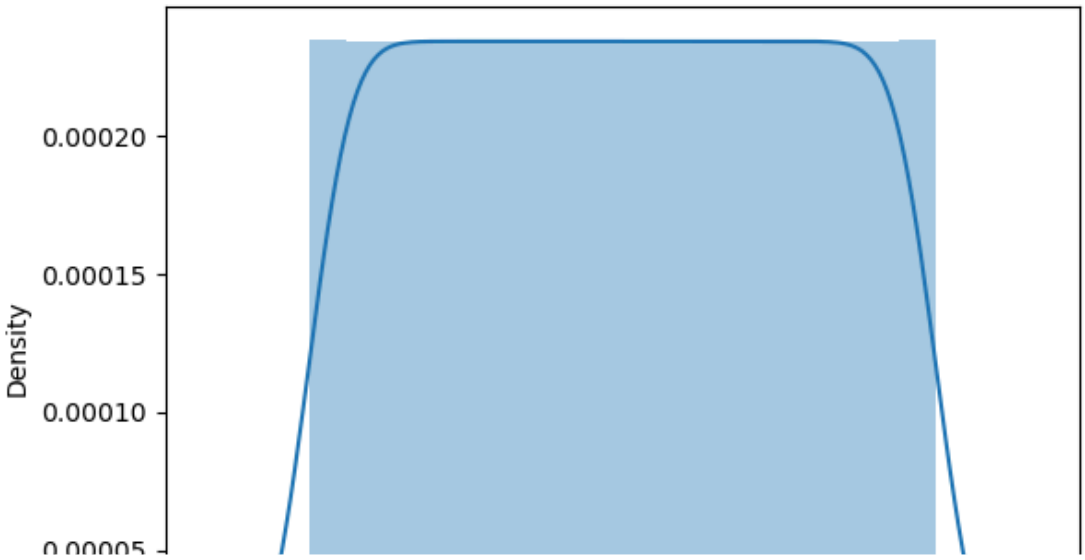
Checking skewness is distribution is distorted or asymmetrical

In [24]:
```
from scipy.stats import skew

colname=df.select_dtypes(['int64','float64']).columns
colname


for col in df[colname]:
    print(col)    #prints column name
    print(skew(X[col])) #prints skewness of col
    plt.figure()
    sns.distplot(X[col])
    plt.show()
```

```
loan_id
0.0
```



$Columns$	$Skewness$	$Skewed - or - not$
$No - of - dependents$	-0.018	$left - Skewed$
$income - annum$	-0.0128	$left - Skewed$
$loan - amount$	0.308	$right - skewed$
$loan - term$	0.036	$no - skew$
$cibil - score$	-0.0090	$no - skew$
$residential - assets$	0.97	$highly - right - skewed$
$commerical - assets$	0.95	$highly - right - skewed$
$luxury - assets$	0.32	$right - skewed$

Using sqrt function to reduce skewness

In [25]:
```
df[' commercial_assets_value']=np.sqrt(df['commercial_assets_value'])
```

In [31]:
```
df[' residential_assets_value']=np.sqrt(df['residential_assets_value'])
```

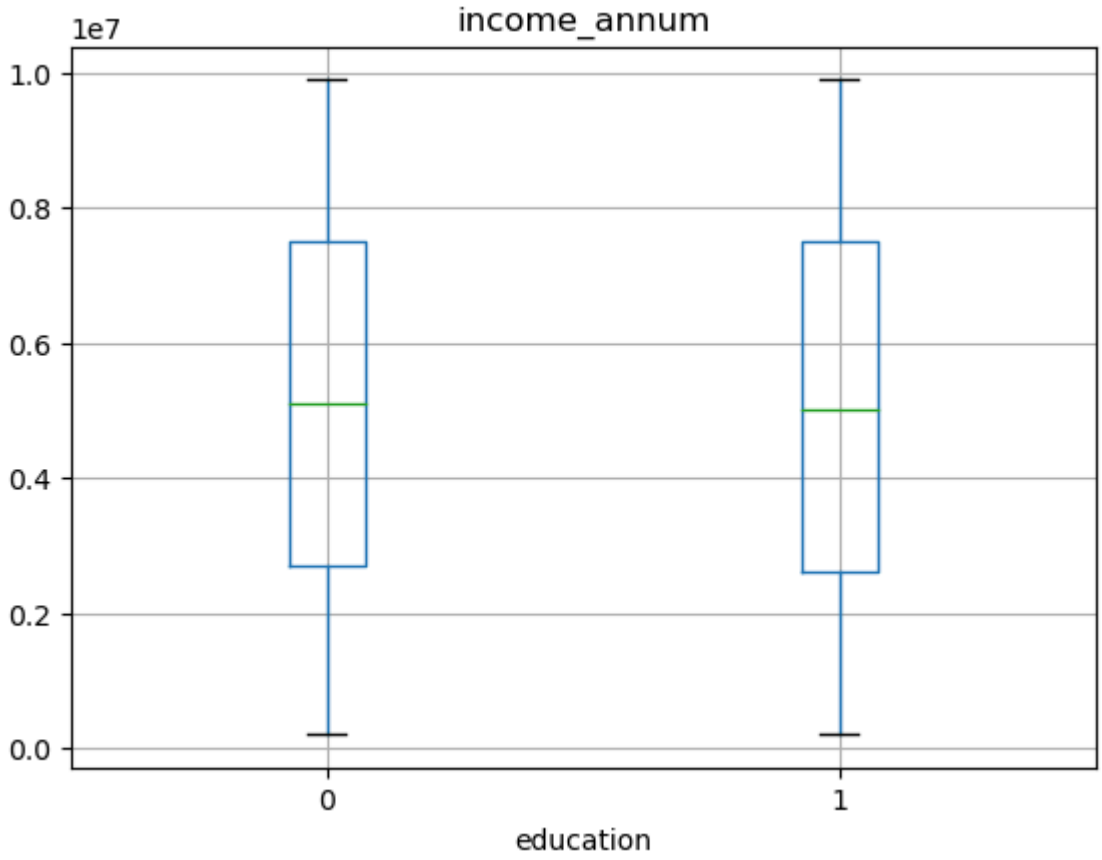
In [32]:
```
skew(df[' commercial_assets_value'])
```

Out[32]:
```
0.14347863248274154
```

Check outlier

In [33]:
```
df.boxplot(column='income_annum', by = 'education')
plt.suptitle("")
```

Out[33]:
```
Text(0.5, 0.98, '')
```



Split into training & testing dataset¶

In [34]:
```
#Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, ran
```

Decision Tree Classifier

In [35]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Create a DecisionTreeClassifier instance
dt = DecisionTreeClassifier(random_state=42)

# Train the decision tree model
dt.fit(X_train, y_train)

# Predict on the test set
y_pred_dt = dt.predict(X_test)
```

Accuracy score

In [36]:
```
score = accuracy_score(y_test, y_pred_dt)

print("Accuracy: ", score)
```

```
Accuracy:  0.9742388758782201
```

Classification Report

In [37]:
```
classification_rep = classification_report(y_test, y_pred_dt)

print("\nClassification Report:\n", classification_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98       536
           1       0.96      0.97      0.97       318

    accuracy                           0.97       854
   macro avg       0.97      0.97      0.97       854
weighted avg       0.97      0.97      0.97       854
```

Confusion matrix

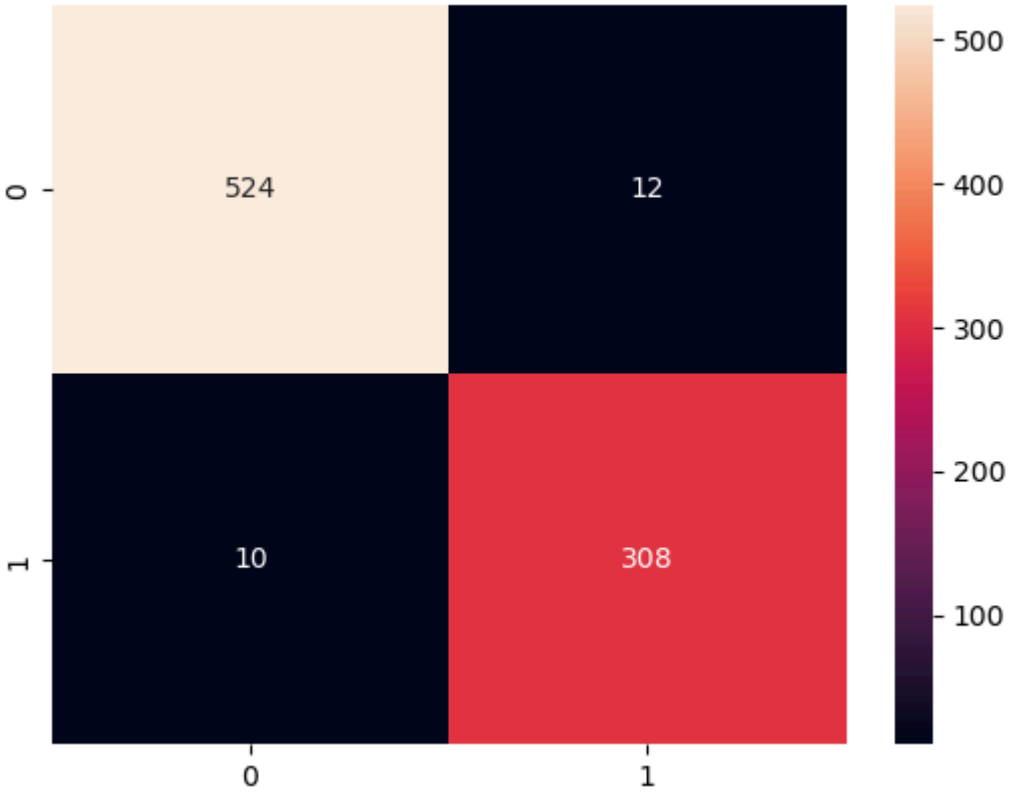
In [38]:
```
cm_dt = confusion_matrix(y_test, y_pred_dt)

confusion = pd.DataFrame(cm_dt, index=['approved', 'not_approved'],
columns=['predicted_approved', 'predicted_not_approved'])
confusion
```

Out[38]:

	predicted_approved	predicted_not_approved
approved	524	12
not_approved	10	308

In [39]:
```
# heatmap
sns.heatmap(cm_dt, annot=True, fmt="d")
plt.show()
```



Decision Tree Classifier : Confusion Matrix Results

- True positive ("Approved" and predicted value "Approved") True prediction --- 527
- True negative ("Rejected" and predicted value "Rejected") True prediction --- 307
- False positive ("Rejected" and predicted value "Approved") Type-1 error --- 9
- False negative ("Approved" and predicted value "Rejected") Type-2 error --- 11

Random forest classifier

In [40]:
```
#Model Trainig
rf = RandomForestClassifier(random_state=42)

# Train the random forest model
rf.fit(X_train,y_train)

# Predict on the test set
y_pred_rf = rf.predict(X_test)
```

Accuracy score

In [41]:
```
score = accuracy_score(y_test, y_pred_rf)
print("Accuracy: ", score)
```

```
Accuracy:  0.9765807962529274
```

Classfication Report

In [42]:
```
classification_rep = classification_report(y_test, y_pred_rf)

print("\nClassification Report:\n", classification_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98       536
           1       0.97      0.97      0.97       318

    accuracy                           0.98       854
   macro avg       0.98      0.97      0.97       854
weighted avg       0.98      0.98      0.98       854
```

Confusion matrix

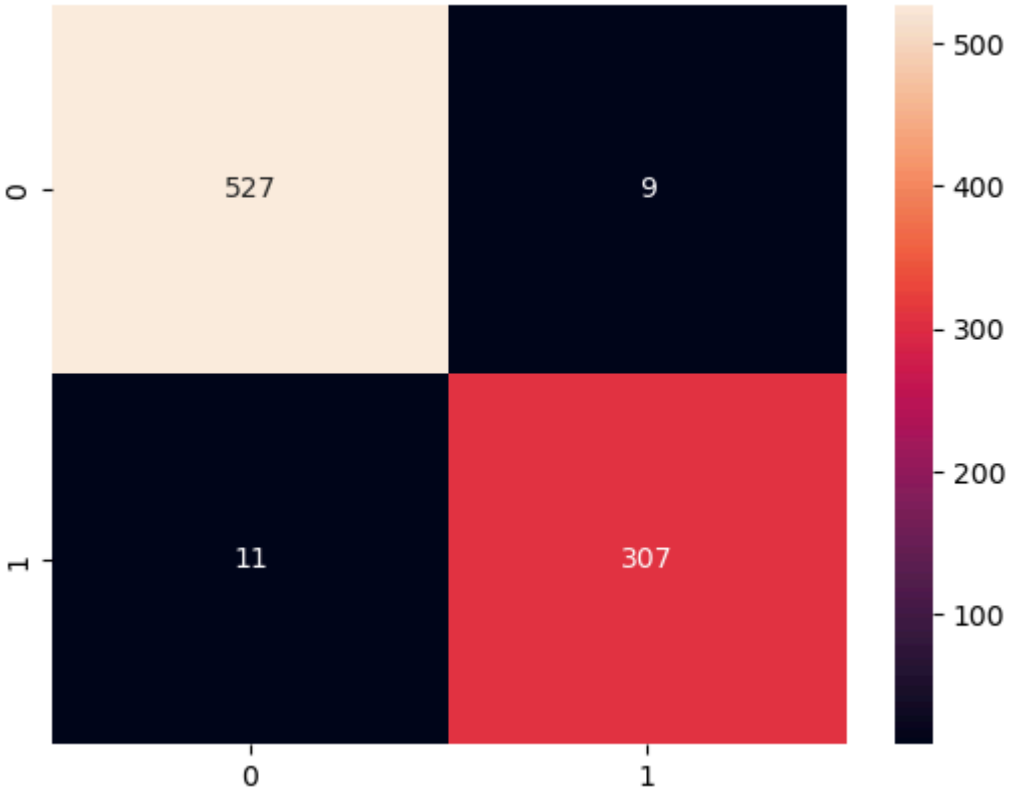
In [43]:
```
cm_rf = confusion_matrix(y_test, y_pred_rf)

confusion = pd.DataFrame(cm_rf, index=['approved', 'not_approved'],
columns=['predicted_approved', 'predicted_not_approved'])
confusion
```

Out[43]:

	predicted_approved	predicted_not_approved
approved	527	9
not_approved	11	307

In [44]:
```
# heatmap
sns.heatmap(cm_rf, annot=True, fmt="d")
plt.show()
```



Random Forest Classifier: Confusion Matrix Result

- True positive ("Approved" and predicted value "Approved") True prediction --- 527
- True negative ("Rejected" and predicted value "Rejected") True prediction --- 303
- False positive ("Rejected" and predicted value "Approved") Type-1 error --- 9
- False negative ("Approved" and predicted value "Rejected") Type-2 error --- 9

In [ ]:
```
```

XGBoost

In [45]:
```
xgb = XGBClassifier(n_jobs=-1)

# Train the random forest model
xgb.fit(X_train, y_train)

# Predict on the test set
y_pred_xgb = xgb.predict(X_test)
```

Accuracy score

In [46]:
```
score = accuracy_score(y_test, y_pred_xgb)

print("Accuracy: ", score)
```

```
Accuracy:  0.9836065573770492
```

Classfication Report

In [47]:
```
classification_rep = classification_report(y_test, y_pred_xgb)

print("\nClassification Report:\n", classification_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       536
           1       0.98      0.97      0.98       318

    accuracy                           0.98       854
   macro avg       0.98      0.98      0.98       854
weighted avg       0.98      0.98      0.98       854
```

Confusion matrix

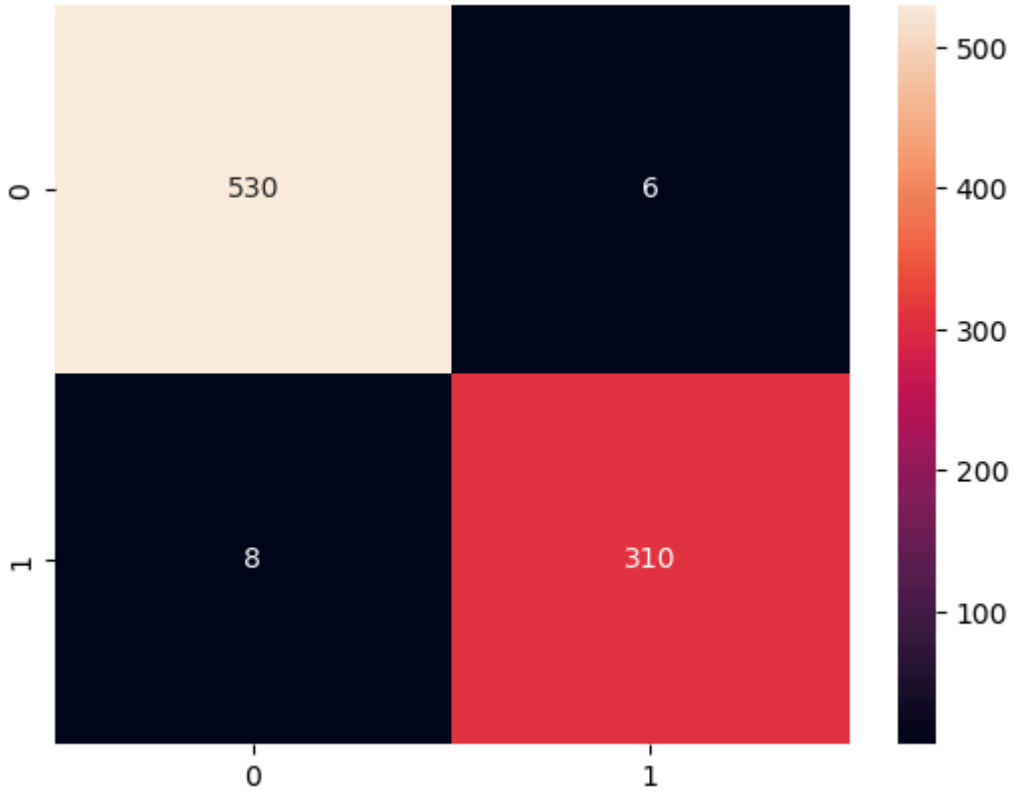
In [48]:
```
cm_xgb = confusion_matrix(y_test, y_pred_xgb)


confusion = pd.DataFrame(cm_xgb, index=['approved', 'not_approved'],
columns=['predicted_approved', 'predicted_not_approved'])
confusion
```

Out[48]:

	predicted_approved	predicted_not_approved
approved	530	6
not_approved	8	310

In [49]:
```
# heatmap
sns.heatmap(cm_xgb, annot=True, fmt="d")
plt.show()
```



XGBoost Classifier : Confusion Matrix Results

- True positive ("Approved" and predicted value "Approved") True prediction --- 530
- True negative ("Rejected" and predicted value "Rejected") True prediction --- 310
- False positive ("Rejected" and predicted value "Approved") Type-1 error --- 6
- False negative ("Approved" and predicted value "Rejected") Type-2 error --- 9

In [ ]:

So it seems Xgboost has lesser Amount for this Type-1 error and Type-2 error as compared to Decision tree and Random forest

So XGBoost is reliable and strong And Accuracy also good by 98%

In [ ]: