CREDIT RISK MODELLING ¶

Group Members-

Purva Burundkar A018

Upasana Yadav A015

Praveena Panicker A014

Objective

The objective of credit risk modeling is to assess and quantify the risk associated with lending money or extending credit to individuals, businesses, or other entities.

Our aim is to build a predictive credit risk model with the help of statistical techniques.

Data Description

This is a credit history of the customers from a financial institution. Agenda is to predict for possible credit defaulters upfront and help the financial institutions to take steps accordingly.

Source- Downloaded dataset from Kaggle

Column Description-

1. loan_amnt- Amount of loan given to the customer

2. term- duration of the loan amount within which the customer has to pay it back

3. int_rate- interest rate is the amount the customer has to pay on the loan amount

4. installment

5. grade

6. emp_length

7. home_ownership

8. annual_inc- it is the Annual Income of the customer

9. verification_status

10. loan_status- loan status shows us whether the customer is a good customer or a bad customer and it our Target dependent variable

11. purpose- it defines the reason for which the customer has taken the loan

Nominal Data:

Nominal data consists of categories or labels that do not have any inherent order or ranking. They represent distinct categories without any numerical value. eg: Home ownership.

Ordinal Data:

Ordinal Data: Ordinal data, unlike nominal data, have a meaningful order or ranking but the intervals between the categories are not consistent. eg: Loan status.

Continuous Data:

Continuous data is data that can take on an infinite number of values within a given range and can be measured with a high level of precision. eg: Employee length.

Discrete Data:

Discrete data is data that can only take on specific, distinct values, often in whole numbers. eg: Grade.

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

In [2]:
```
df=pd.read_csv('D:/LOANC.csv')
df
```

Out[2]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_i
0	5000	36 months	10.65	162.87	B	10+ years	RENT	24000
1	2500	60 months	15.27	59.83	C	< 1 year	RENT	30000
2	2400	36 months	15.96	84.33	C	10+ years	RENT	12252
3	10000	36 months	13.49	339.31	C	10+ years	RENT	49200
4	3000	60 months	12.69	67.79	B	1 year	RENT	80000
...	...	...	...	...	...	...	...	
887374	10000	36 months	11.99	332.10	B	8 years	RENT	31000
887375	24000	36 months	11.99	797.03	B	10+ years	MORTGAGE	79000
887376	13000	60 months	15.99	316.07	D	5 years	RENT	35000
887377	12000	60 months	19.99	317.86	E	1 year	RENT	64400
887378	20000	36 months	11.99	664.20	B	10+ years	RENT	100000

887379 rows × 11 columns

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

Out[4]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc	v
0	5000	36 months	10.65	162.87	B	10+ years	RENT	24000.0	
1	2500	60 months	15.27	59.83	C	< 1 year	RENT	30000.0	
2	2400	36 months	15.96	84.33	C	10+ years	RENT	12252.0	
3	10000	36 months	13.49	339.31	C	10+ years	RENT	49200.0	
4	3000	60 months	12.69	67.79	B	1 year	RENT	80000.0	

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

Out[5]:
```
(887379, 11)
```

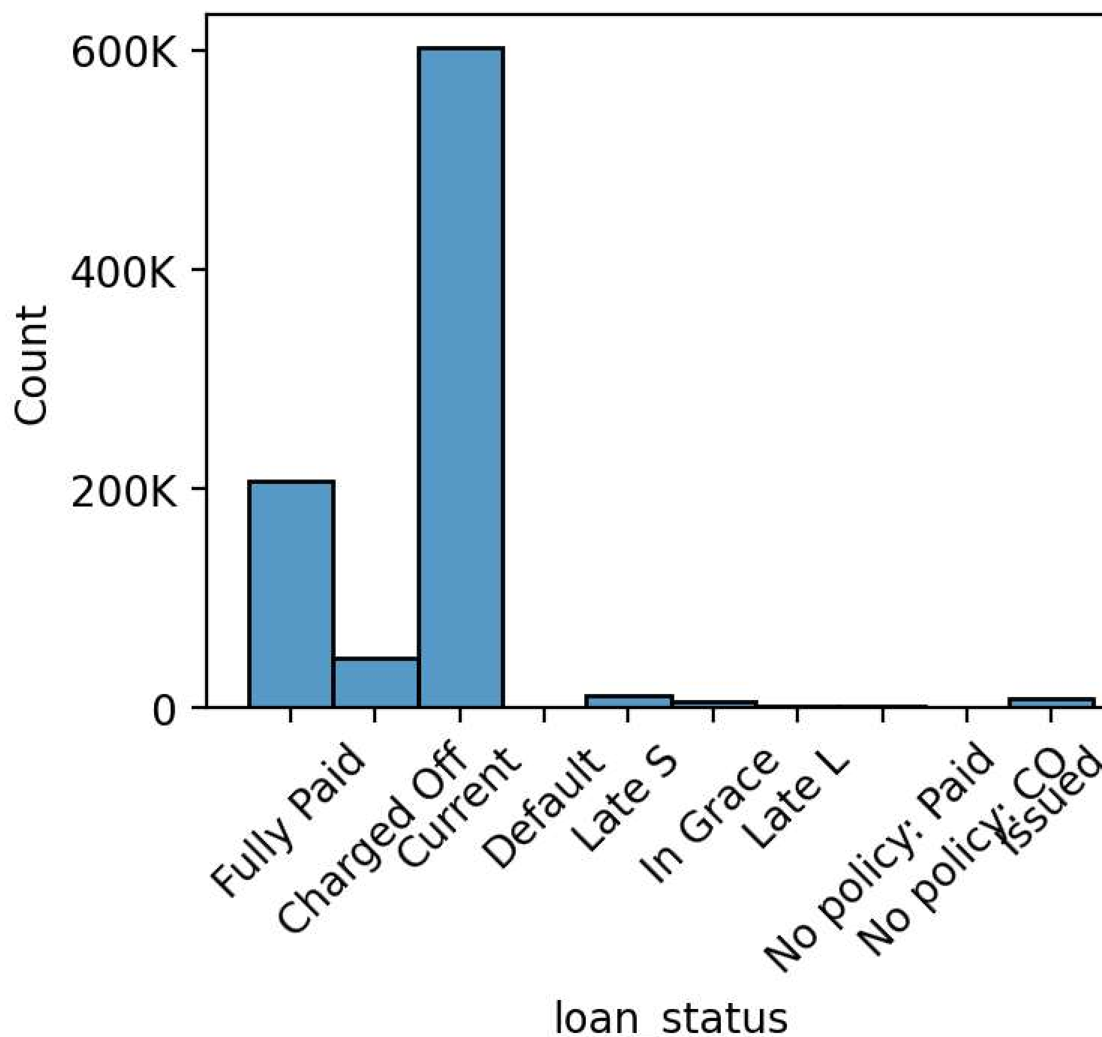
In [6]:
```
df.describe
```

Out[6]:
```
<bound method NDFrame.describe of         loan_amnt        term  int_rate  i
nstallment grade emp_length  \
0            5000   36 months     10.65       162.87     B  10+ years
1            2500   60 months     15.27        59.83     C   < 1 year
2            2400   36 months     15.96        84.33     C  10+ years
3           10000   36 months     13.49       339.31     C  10+ years
4            3000   60 months     12.69        67.79     B     1 year
...           ...         ...       ...          ...   ...        ...
887374      10000   36 months     11.99       332.10     B    8 years
887375      24000   36 months     11.99       797.03     B  10+ years
887376      13000   60 months     15.99       316.07     D    5 years
887377      12000   60 months     19.99       317.86     E     1 year
887378      20000   36 months     11.99       664.20     B  10+ years

       home_ownership  annual_inc verification_status loan_status  \
0                RENT     24000.0            Verified  Fully Paid
1                RENT     30000.0     Source Verified Charged Off
2                RENT     12252.0        Not Verified  Fully Paid
3                RENT     49200.0     Source Verified  Fully Paid
4                RENT     80000.0     Source Verified     Current
...               ...         ...                 ...         ...
887374           RENT     31000.0            Verified     Current
887375       MORTGAGE     79000.0            Verified     Current
887376           RENT     35000.0            Verified     Current
887377           RENT     64400.0     Source Verified     Current
887378           RENT    100000.0            Verified     Current

                   purpose
0              credit_card
1                      car
2           small_business
3                    other
4                    other
...                    ...
887374  debt_consolidation
887375    home_improvement
887376  debt_consolidation
887377  debt_consolidation
887378         credit_card

[887379 rows x 11 columns]>
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887379 entries, 0 to 887378
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   loan_amnt            887379 non-null  int64  
 1   term                 887379 non-null  object 
 2   int_rate             887379 non-null  float64
 3   installment          887379 non-null  float64
 4   grade                887379 non-null  object 
 5   emp_length           842554 non-null  object 
 6   home_ownership       887379 non-null  object 
 7   annual_inc           887375 non-null  float64
 8   verification_status  887379 non-null  object 
 9   loan_status          887379 non-null  object 
 10  purpose              887379 non-null  object 
dtypes: float64(3), int64(1), object(7)
memory usage: 74.5+ MB
```

In [8]:
```
plt.figure(figsize=(4, 3), dpi=200)

# Create a histogram using seaborn
sns.histplot(df.loan_status, kde=False)

# Specify the locations of the x-tick marks
plt.xticks([0,1, 2, 3, 4, 5, 6, 7, 8, 9], ['Fully Paid', 'Charged Off', 'Curre

plt.yticks([0, 200000, 400000, 600000], ['0','200K','400K','600K'])

# Rotate the x-labels by 45 degrees
plt.xticks(rotation=45)

plt.savefig('status.svg', format='svg')

# Show the plot
plt.show()
```



In [9]:
```
df.isnull().sum().sort_values()/len(df) *100
```

Out[9]:
```
loan_amnt              0.000000
term                   0.000000
int_rate               0.000000
installment            0.000000
grade                  0.000000
home_ownership         0.000000
verification_status    0.000000
loan_status            0.000000
purpose                0.000000
annual_inc             0.000451
emp_length             5.051393
dtype: float64
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
loan_amnt                  0
term                       0
int_rate                   0
installment                0
grade                      0
emp_length             44825
home_ownership             0
annual_inc                 4
verification_status        0
loan_status                0
purpose                    0
dtype: int64
```

In [11]:
```
#Description of loan status
#Encoding loan status
# binary classification
label_categories = [
    (0, ['Fully Paid', 'Does not meet the credit policy. Status:Fully Paid', '
    (1, ['Late (31-120 days)', 'Late (16-30 days)', 'In Grace Period',
         'Charged Off', 'Default', 'Does not meet the credit policy. Status:Ch
]

# function to apply the transformation
def classify_label(text):
    for category, matches in label_categories:
        if any(match in text for match in matches):
            return category
    return None

df.iloc[:,-2] = df['loan_status'].apply(classify_label)
```

In [12]:
```
df.head()
```

Out[12]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc	v
0	5000	36 months	10.65	162.87	B	10+ years	RENT	24000.0	
1	2500	60 months	15.27	59.83	C	< 1 year	RENT	30000.0	
2	2400	36 months	15.96	84.33	C	10+ years	RENT	12252.0	
3	10000	36 months	13.49	339.31	C	10+ years	RENT	49200.0	
4	3000	60 months	12.69	67.79	B	1 year	RENT	80000.0	

In [13]:

```
# label several label with specific grading system.
def SC_LabelEncoder1(text):
    if text == "E":
        return 1
    elif text == "D":
        return 2
    elif text == "C":
        return 3
    elif text == "B":
        return 4
    elif text == "A":
        return 5
    else:
        return 0



def SC_LabelEncoder2(text):
    if text == "< 1 year":
        return 1
    elif text == "1 year":
        return 2
    elif text == "2 years":
        return 3
    elif text == "3 years":
        return 4
    elif text == "4 years":
        return 5
    elif text == "5 years":
        return 6
    elif text == "6 years":
        return 7
    elif text == "7 years":
        return 8
    elif text == "8 years":
        return 9
    elif text == "9 years":
        return 10
    elif text == "10 years":
        return 11
    elif text == "10+ years":
        return 12
    else:
        return 0

def SC_LabelEncoder3(text):
    if text == "RENT":
        return 1
    elif text == "MORTGAGE":
        return 2
    elif text == "OWN":
        return 3
    else:
        return 0

df["grade"] = df["grade"].apply(SC_LabelEncoder1)
df["emp_length"] = df["emp_length"].apply(SC_LabelEncoder2)
```

```
df["home_ownership"] = df["home_ownership"].apply(SC_LabelEncoder3)
```

In [14]:
```
df.head(10)
```

Out[14]:

	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc	v
0	5000	36 months	10.65	162.87	4	12	1	24000.0	
1	2500	60 months	15.27	59.83	3	1	1	30000.0	
2	2400	36 months	15.96	84.33	3	12	1	12252.0	
3	10000	36 months	13.49	339.31	3	12	1	49200.0	
4	3000	60 months	12.69	67.79	4	2	1	80000.0	
5	5000	36 months	7.90	156.46	5	4	1	36000.0	
6	7000	60 months	15.96	170.08	3	9	1	47004.0	
7	3000	36 months	18.64	109.43	1	10	1	48000.0	
8	5600	60 months	21.28	152.39	0	5	3	40000.0	
9	5375	60 months	12.69	121.45	4	1	1	15000.0	

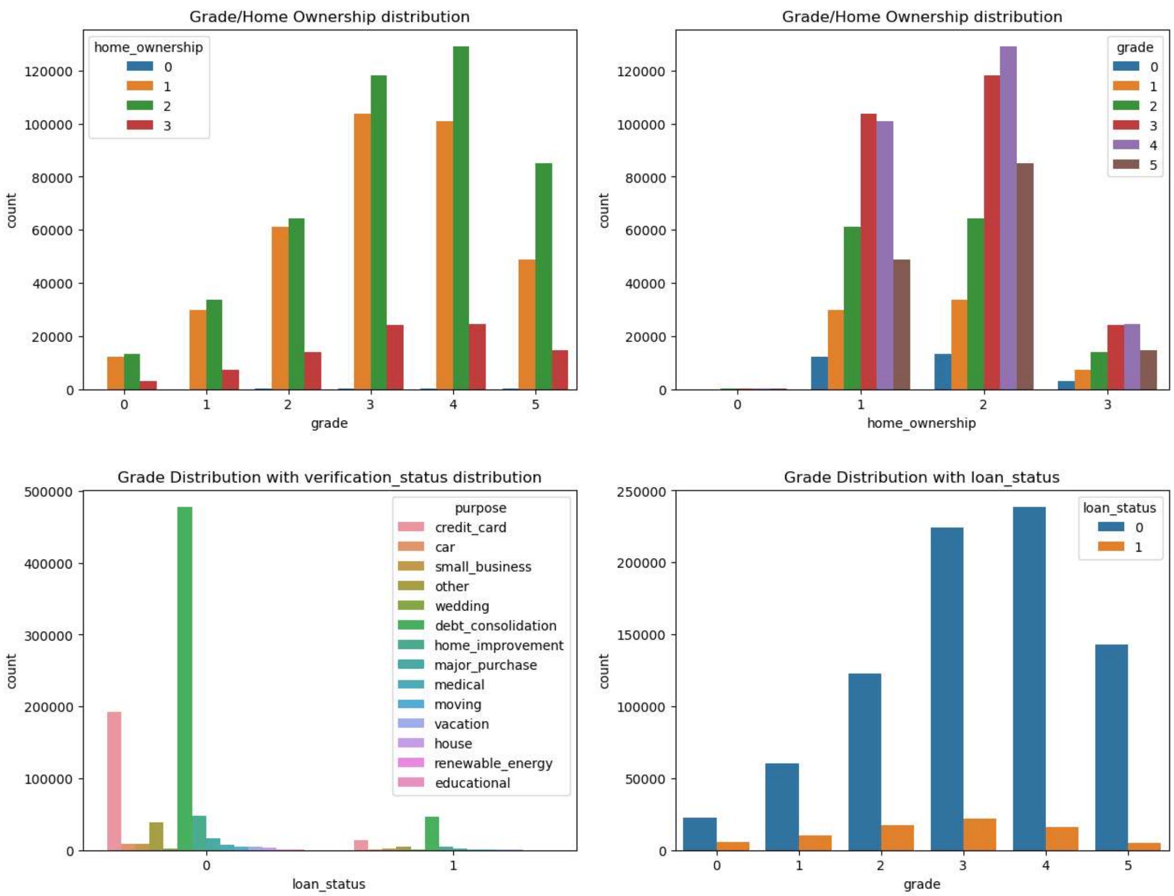
In [15]:
```
df.annual_inc = df.annual_inc.fillna(0)
```

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
loan_amnt              0
term                   0
int_rate               0
installment            0
grade                  0
emp_length             0
home_ownership         0
annual_inc             0
verification_status    0
loan_status            0
purpose                0
dtype: int64
```

EXPLORATORY DATA ANALYSIS(EDA)

```
In [17]: fig, ax = plt.subplots(1,2,figsize=(15,5))
         sns.countplot(data=df, x='grade', hue="home_ownership", ax=ax[0]).set_title("G
         sns.countplot(data=df, x='home_ownership', hue='grade', ax=ax[1]).set_title("G

         fig, ax = plt.subplots(1,2,figsize=(15,5))
         sns.countplot(data=df, x='loan_status', hue='purpose', ax=ax[0]).set_title("Gr
         sns.countplot(data=df, x='grade', hue='loan_status', ax=ax[1]).set_title("Grad
```

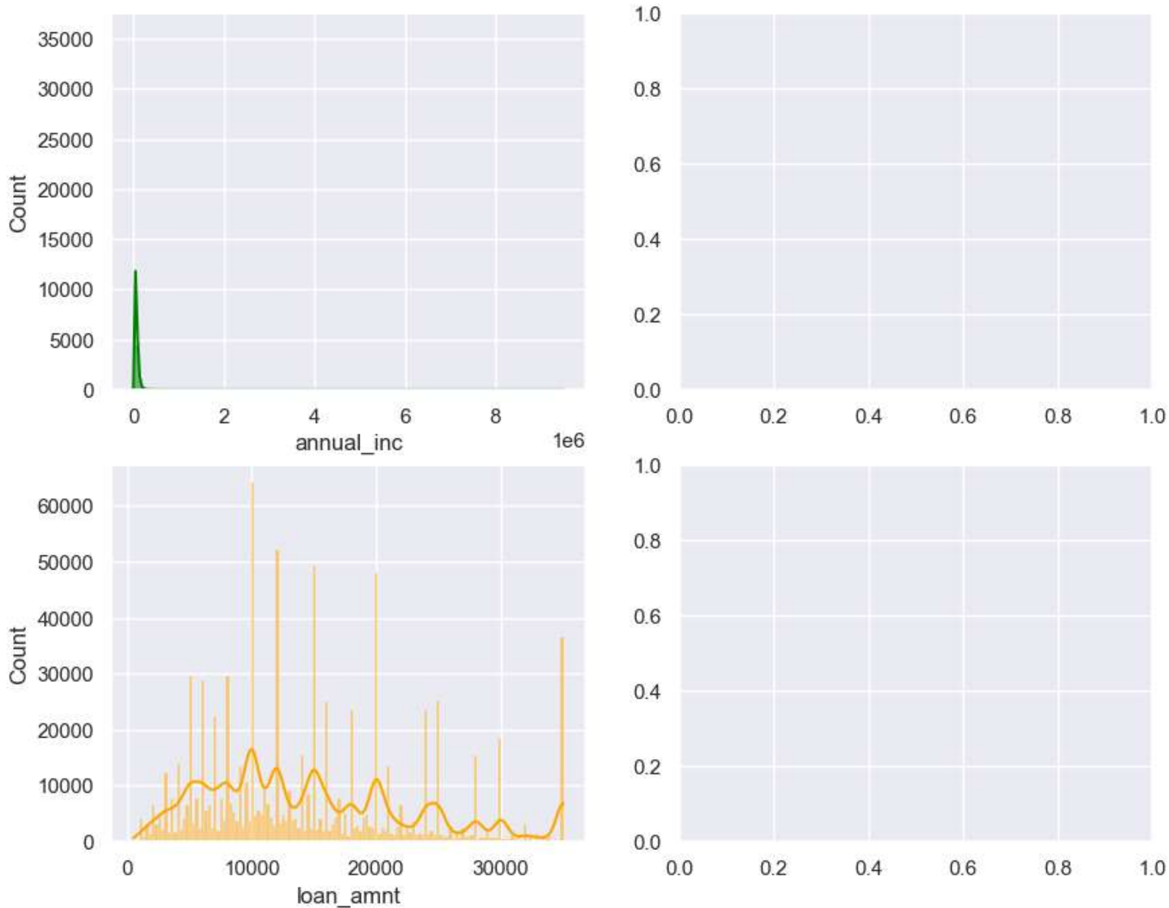


Analysis

The number of Borrowers with high grade will be small compared to low grade Most money borrowers' goals from labels 0 and 1 are debt consolidation The highest number of grades who were able to complete the loan was grade 4, while the most failed to complete the loan was grade 3

Checking if data is normally distributed ?

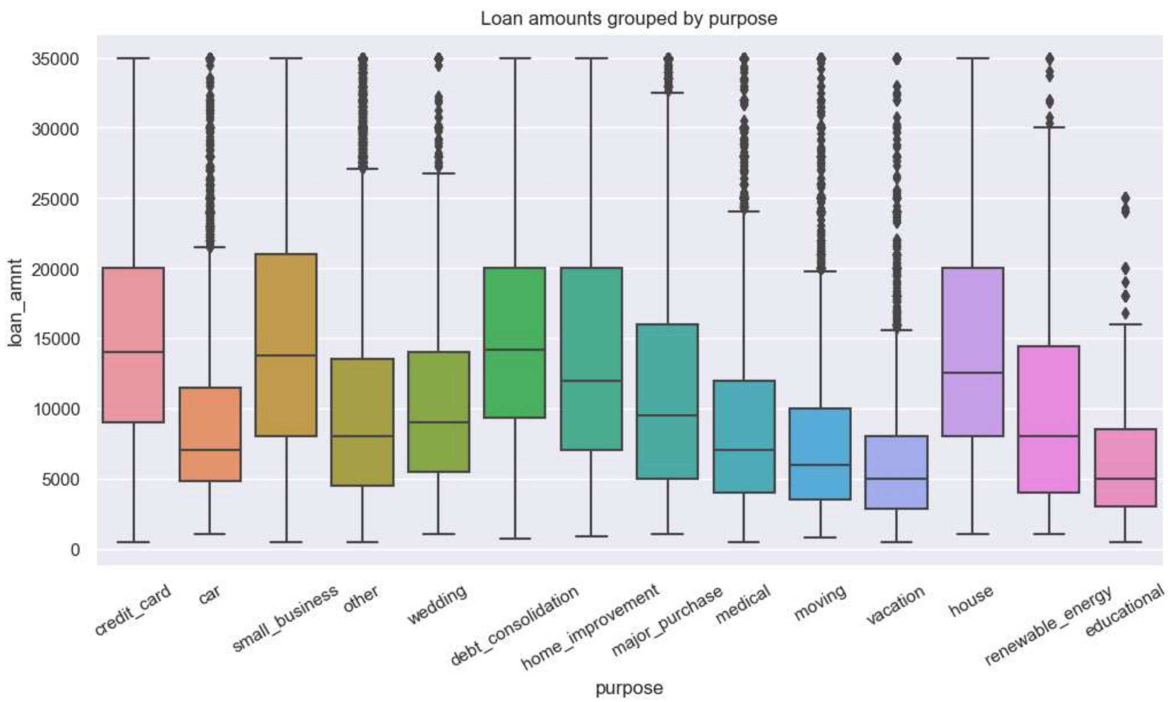
In [18]:
```
sns.set(style="darkgrid")
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

sns.histplot(data=df, x="annual_inc", kde=True, ax=axs[0, 0], color='green')
#sns.histplot(data=df, x="annual_inc", kde=True, ax=axs[0, 1], color='skyblue
sns.histplot(data=df, x="loan_amnt", kde=True, ax=axs[1, 0], color='orange');
```



Box plot of loan amount grouped by purpose

In [19]:
```
plt.figure(figsize=(12,6))
sns.boxplot(x='purpose', y='loan_amnt', data=df)
plt.xticks(rotation=30)
plt.title('Loan amounts grouped by purpose')
```

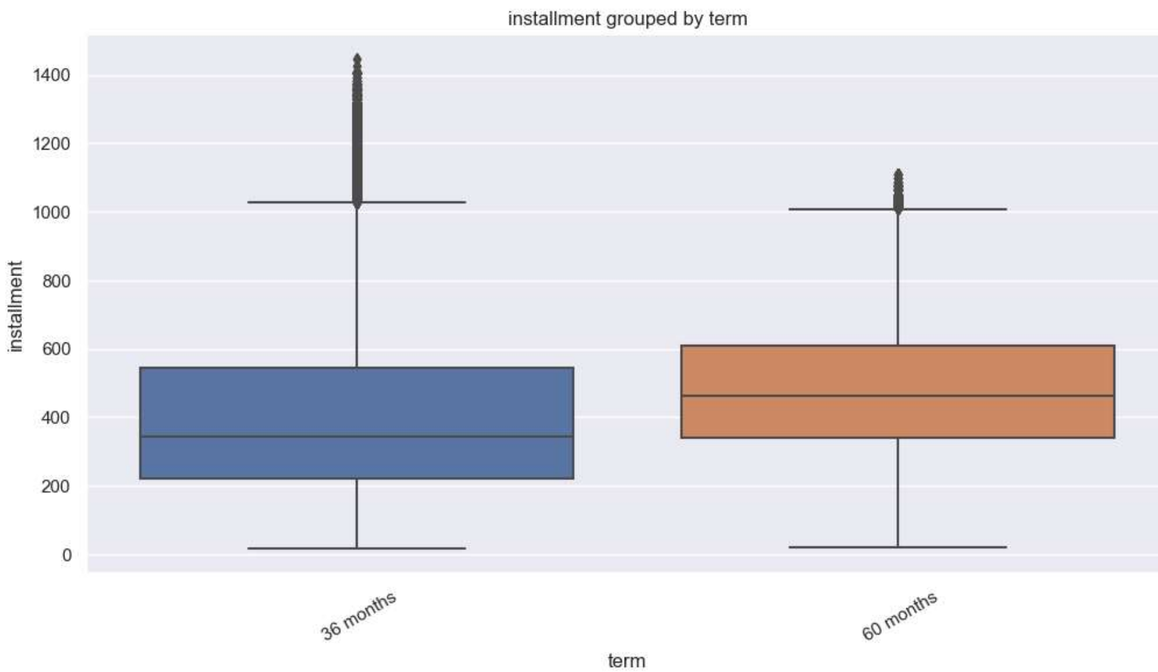
Out[19]:
```
Text(0.5, 1.0, 'Loan amounts grouped by purpose')
```



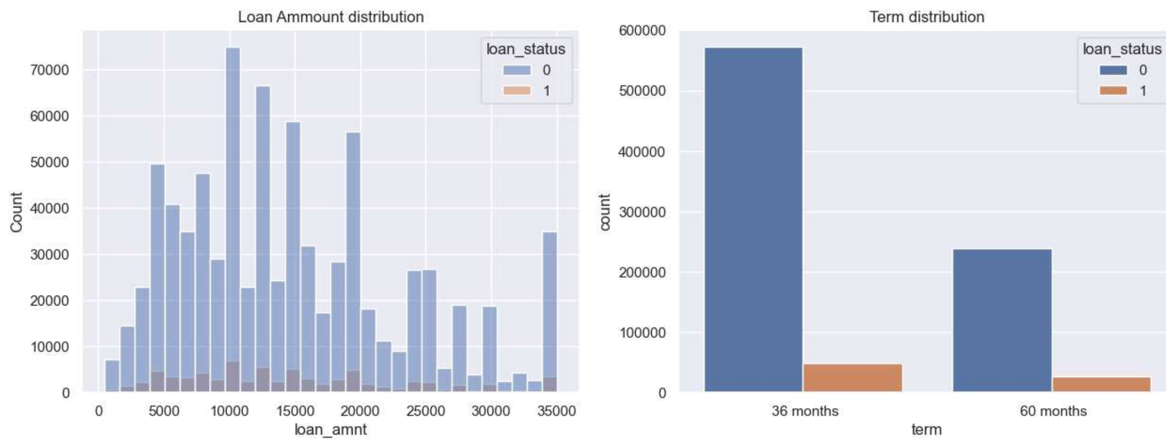
Box plot of installment grouped by term

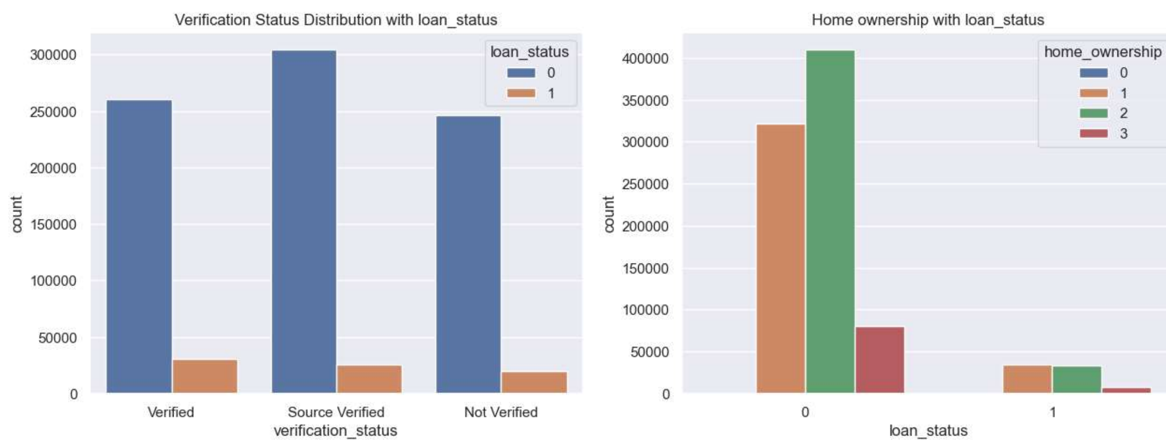
In [20]:
```
plt.figure(figsize=(12,6))
sns.boxplot(x='term', y='installment', data=df)
plt.xticks(rotation=30)
plt.title('installment grouped by term')
```

Out[20]:
```
Text(0.5, 1.0, 'installment grouped by term')
```



```
In [21]: fig, ax = plt.subplots(1,2,figsize=(15,5))
         sns.histplot(df, x='loan_amnt',hue="loan_status", bins=30, ax=ax[0]).set_title
         sns.countplot(data=df, x='term', hue="loan_status", ax=ax[1]).set_title("Term

         fig, ax = plt.subplots(1,2,figsize=(15,5))
         sns.countplot(data=df, hue='home_ownership', x='loan_status', ax=ax[1]).set_ti
         sns.countplot(data=df, x='verification_status', hue='loan_status', ax=ax[0]).s
```



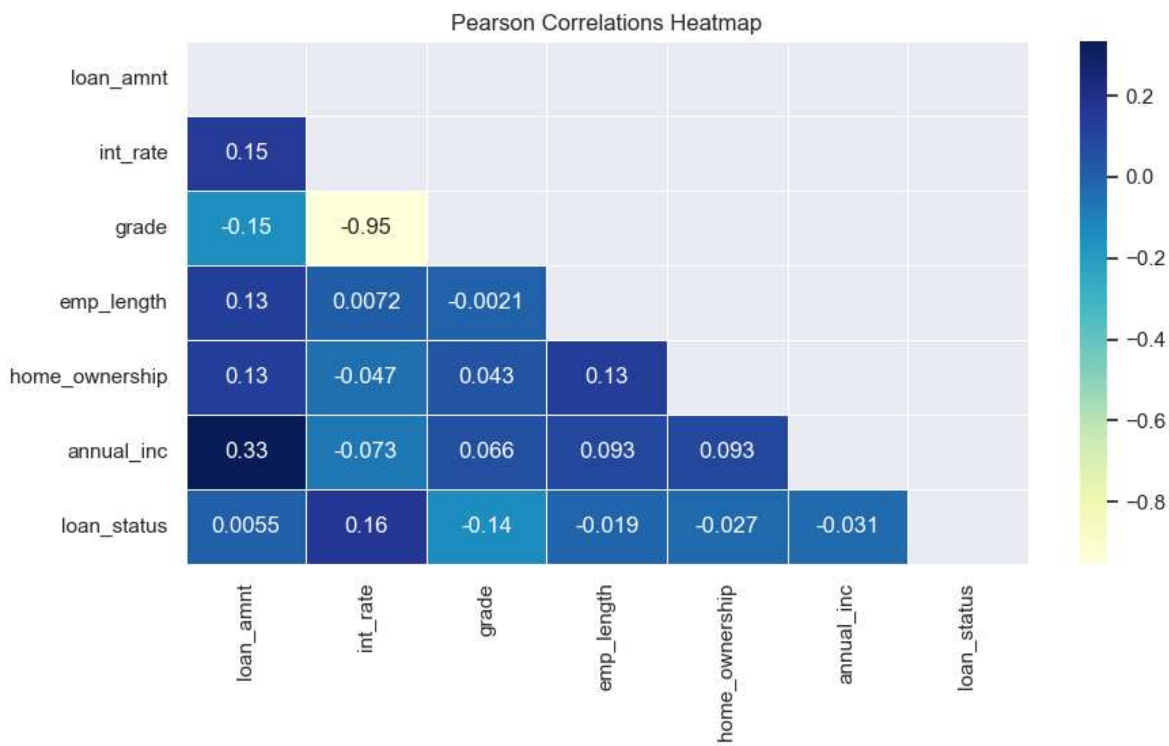


Analysis

The nominal value of the largest debt is 10000 USD The maximum maturity is 36 months, while for 60 months it is almost a third Most of the credits that can be paid in full are obtained from the "Verified" verification status

In [22]:
```
#seeing the correlation

corr = df[['loan_amnt', 'int_rate', 'grade', 'emp_length', 'home_ownership', '
sns.set(rc={'figure.figsize':(10,5)})
sns.heatmap(corr,linewidths=.5, annot=True, cmap="YlGnBu",mask=np.triu(np.ones
    .set_title("Pearson Correlations Heatmap");
```

```
C:\Users\Praveena p\AppData\Local\Temp\ipykernel_22284\1188946961.py:5: Depr
ecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To s
ilence this warning, use `bool` by itself. Doing this will not modify any be
havior and is safe. If you specifically wanted the numpy scalar type, use `n
p.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/d
evdocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/rel
ease/1.20.0-notes.html#deprecations)
  sns.heatmap(corr,linewidths=.5, annot=True, cmap="YlGnBu",mask=np.triu(np.
ones_like(corr, dtype=np.bool)))\
```



Analysis

the credit is dependent on the annual income of the borrower

In [23]:
```python
# using LabelEncoder() to encode another category column:
from sklearn.preprocessing import LabelEncoder
for col in ["verification_status", "purpose","term"]:
    le = LabelEncoder()
    le.fit(df[col])
    df[col] = le.transform(df[col])
df.head()
```

Out[23]:

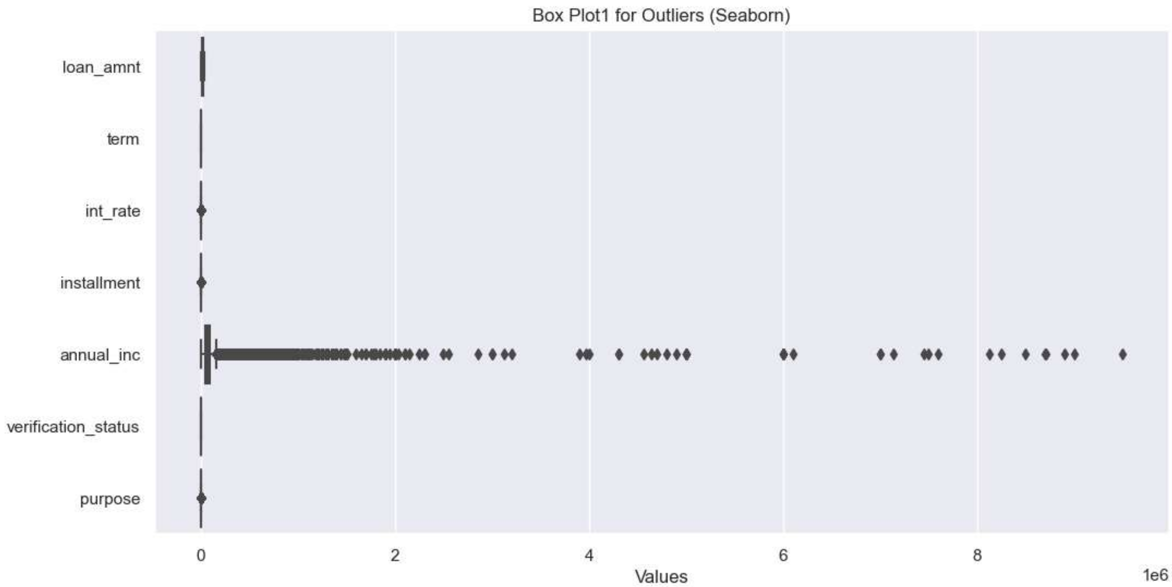
	loan_amnt	term	int_rate	installment	grade	emp_length	home_ownership	annual_inc	veri
0	5000	0	10.65	162.87	4	12	1	24000.0	
1	2500	1	15.27	59.83	3	1	1	30000.0	
2	2400	0	15.96	84.33	3	12	1	12252.0	
3	10000	0	13.49	339.31	3	12	1	49200.0	
4	3000	1	12.69	67.79	4	2	1	80000.0	

In [24]:
```python
df.isnull().sum()
```

Out[24]:
```
loan_amnt              0
term                   0
int_rate               0
installment            0
grade                  0
emp_length             0
home_ownership         0
annual_inc             0
verification_status    0
loan_status            0
purpose                0
dtype: int64
```

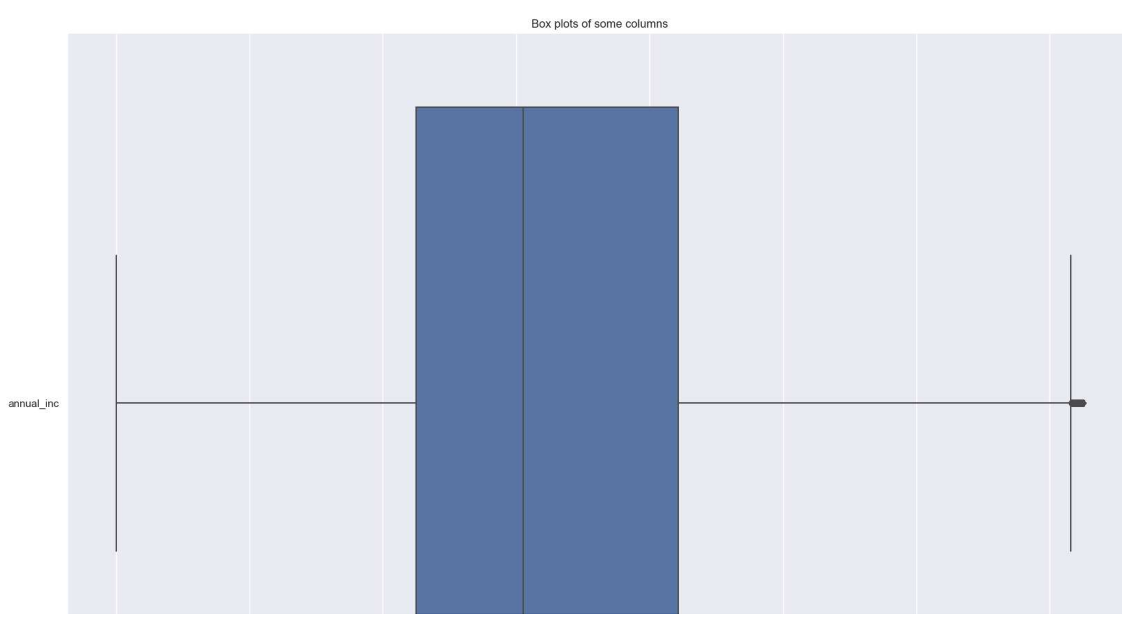
finding outliers

In [20]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, orient='h')
plt.xlabel('Values')
plt.title('Box Plot1 for Outliers (Seaborn)')
plt.show()
```



In [66]:
```
df_columns=["annual_inc"]
for column in df_columns:
  df=remove_outlier(df,df_columns)
plt.figure(figsize=(20,14))
sns.boxplot(data=df[df_columns],orient='h')
plt.title("Box plots of some columns")
plt.xlabel("Values")
```

Out[66]: Text(0.5, 0, 'Values')

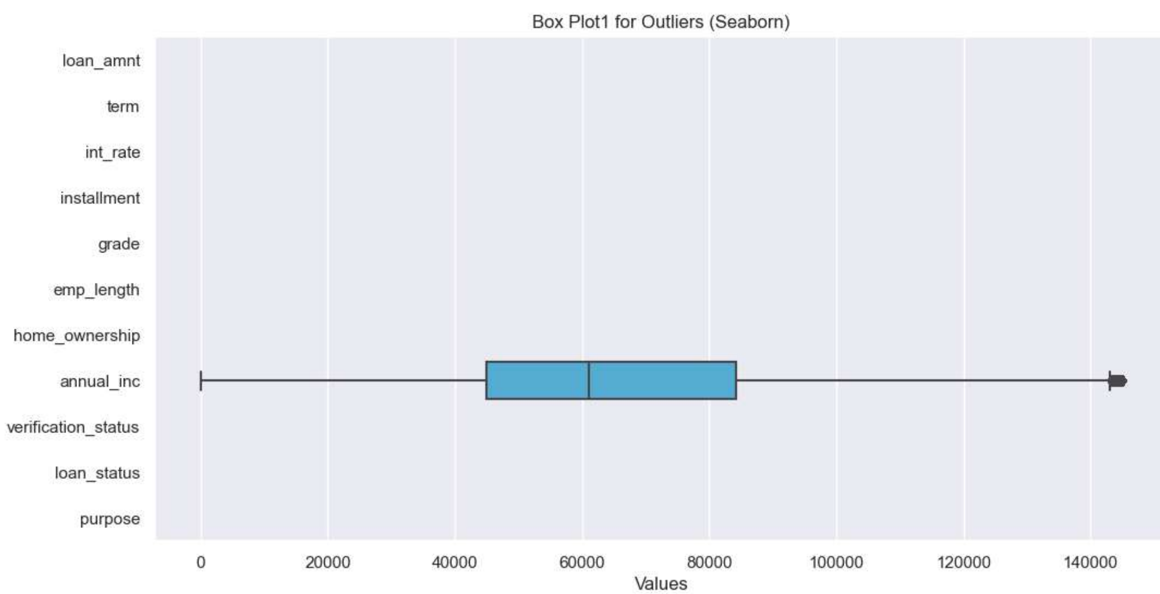


In [ ]:

```
def remove_outlier(df,column,lower_bound=0.25,upper_quartile=0.75):
  q1=df[column].quantile(lower_bound)
  q3=df[column].quantile(upper_quartile)
  iqr=q3-q1
  lower_bound=q1-1.5*iqr
  upper_bound=q3+1.5*iqr
  return df[(df[column]<=upper_bound) & (df[column]>=lower_bound)]
df_columns=["annual_inc"]
for column in df_columns:
  df=remove_outlier(df,df_columns)
plt.subplot(1,2,1)
sns.boxplot(df=df[df_columns],orient='h')
plt.title("Box plots of some columns")
plt.xlabel("Values")
```

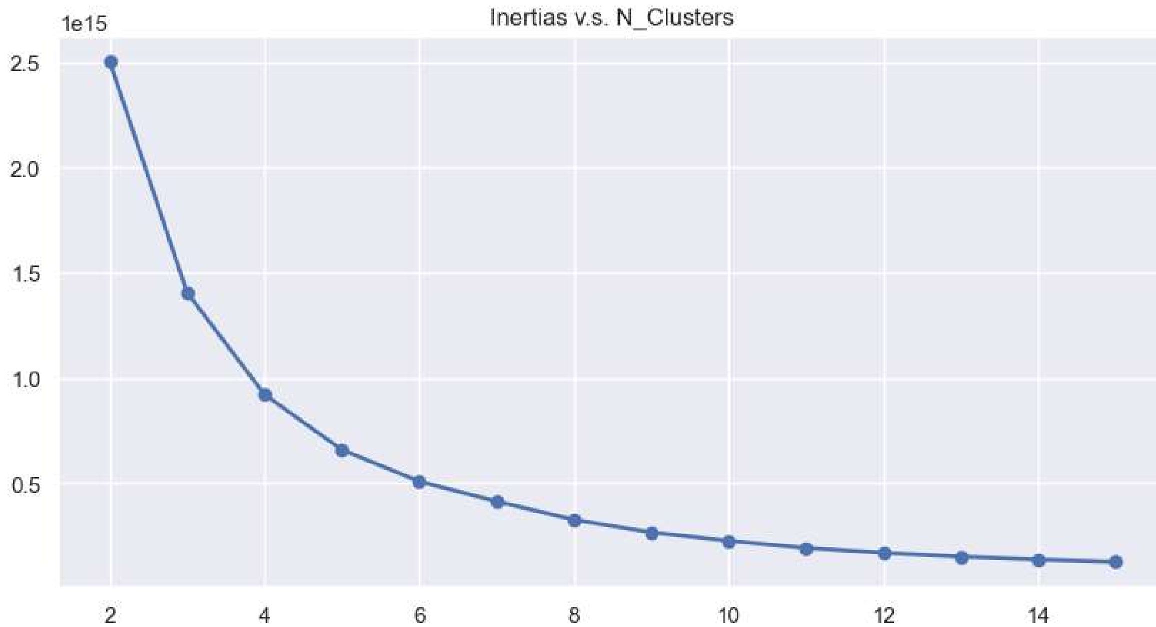
In [67]:

```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, orient='h')
plt.xlabel('Values')
plt.title('Box Plot1 for Outliers (Seaborn)')
plt.show()
```



Clustering

In [25]:
```
from sklearn.cluster import KMeans
inertias = []

for i in range(2,16):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(df)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(10,5))
plt.title('Inertias v.s. N_Clusters')
plt.plot(np.arange(2,16),inertias,marker='o',lw=2);
```



Analysis:

"Elbow" on the chart above is at 4. The number of clusters must be 4. Since the value of k should be odd we will take the value as 5.

KNN Model

In [26]:
```
x=df.drop("loan_status", axis=1)
y=df["loan_status"]
x.shape
```

Out[26]: (887379, 10)

In [27]:
```
y.shape
```
Out[27]:
```
(887379,)
```
In [28]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state
```
In [29]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=5, metric="euclidean")
classifier.fit(x_train,y_train)
```
Out[29]:
```
KNeighborsClassifier(metric='euclidean')
```
In [30]:
```
y_pred=classifier.predict(x_test)
y_pred
```
```
C:\Users\Praveena p\anaconda3\lib\site-packages\sklearn\neighbors\_classific
ation.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`,
`kurtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value of
`keepdims` will become False, the `axis` over which the statistic is taken w
ill be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```
Out[30]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```
In [31]:
```
from sklearn.metrics import confusion_matrix,f1_score, classification_report,
print(confusion_matrix(y_pred,y_test))
```
```
[[200990  18757]
 [  1757    341]]
```
In [32]:
```
print(accuracy_score(y_pred,y_test))
```
```
0.90753003222971
```
In [33]:
```
print(classification_report(y_pred,y_test))
```
```
              precision    recall  f1-score   support

           0       0.99      0.91      0.95    219747
           1       0.02      0.16      0.03      2098

    accuracy                           0.91    221845
   macro avg       0.50      0.54      0.49    221845
weighted avg       0.98      0.91      0.94    221845
```

Gaussian Naive Bayes Model

In [34]:
```
P=df.drop("loan_status", axis=1)
Q=df["loan_status"]
```

In [36]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(P,Q,test_size=0.25,random_state
```

In [37]:
```
from sklearn.naive_bayes import GaussianNB

nb= GaussianNB()
nb.fit(X_train,Y_train)
```

Out[37]:
```
GaussianNB()
```

In [38]:
```
Y_predict=nb.predict(X_test)
Y_predict
```

Out[38]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [43]:
```
print(accuracy_score(Y_predict,Y_test))
```

```
0.913146566296288
```

In [44]:
```
print(classification_report(Y_predict,Y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.91      0.95    221531
           1       0.00      0.23      0.01       314

    accuracy                           0.91    221845
   macro avg       0.50      0.57      0.48    221845
weighted avg       1.00      0.91      0.95    221845
```

In [45]:
```
from sklearn.metrics import mean_squared_error
```

In [46]:
```
print(mean_squared_error(Y_predict,Y_test))
```

```
0.08685343370371205
```

Random Forest

In [47]:
```
A=df.drop("loan_status", axis=1)
B=df["loan_status"]
```

In [55]:
```
from sklearn.model_selection import train_test_split
x1_train,x2_test,y1_train,y2_test = train_test_split(A,B,test_size = 0.25, ran
```

In [56]:
```
# n_estimators: the number of trees in the forest. Usually the higher the numb
# max_depth: The number of splits that each decision tree is allowed to make.
# OOB (out-of-bag) score is a performance metric for a machine learning model,
# n_jobs: integer,optional (default=1) The number of jobs to run in parallel f

from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5,n_estimato
```

In [57]:
```
rf.fit(x1_train, y1_train)
```

Out[57]:
```
RandomForestClassifier(max_depth=5, n_jobs=-1, oob_score=True, random_state=
42)
```

In [59]:
```
Y1_predict=rf.predict(x2_test)
Y1_predict
```

Out[59]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [60]:
```
from sklearn.metrics import classification_report, accuracy_score
print(accuracy_score(Y1_predict,y2_test))
```

```
0.9139128670918885
```

In [61]:
```
print(classification_report(Y1_predict,y2_test))
```

```
C:\Users\Praveena p\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined and
being set to 0.0 in labels with no true samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           0       1.00      0.91      0.96    221845
           1       0.00      0.00      0.00         0

    accuracy                           0.91    221845
   macro avg       0.50      0.46      0.48    221845
weighted avg       1.00      0.91      0.96    221845


C:\Users\Praveena p\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined and
being set to 0.0 in labels with no true samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Praveena p\anaconda3\lib\site-packages\sklearn\metrics\_classificat
ion.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined and
being set to 0.0 in labels with no true samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [62]:
```
from sklearn.metrics import mean_squared_error
```

In [63]:
```
print(mean_squared_error(Y1_predict,y2_test))
```

```
0.08608713290811151
```

LEARNING FROM OUR PROJECT

Data Collection and Preprocessing:Gathered and cleaned the dataset relevant to credit risk modeling. This includes dealing with missing values, outliers, and noisy data.

Feature Engineering: Transformed raw data into meaningful features.

Data Exploration: Explored different relationships between the variables to understand its characteristics using EXPLORATORY DATA ANALYSIS(EDA)

Model Selection: Performed different machine learning and statistical models suitable for credit risk modeling, such as KNN, Gaussian Naive Bayes, Clustering and random forest.

Model Evaluation: Evaluated the performance of our models using various metrics like accuracy, precision, recall, F1-score,mean square error.

Communication:Effectively comunicated our findings, conlusions as well as future works.

CONCLUSION

Our credit risk modeling project has successfully developed a robust and accurate predictive model to assess the creditworthiness of individuals or entities.

Through rigorous data analysis, feature engineering, and machine learning techniques, we have created a tool that effectively identifies potential credit risks.

This model can enhance decision-making processes, reduce financial losses, and ultimately contribute to a more secure and profitable lending portfolio for our organization.

Ongoing monitoring and refinement will be essential to ensure the model's continued effectiveness in managing credit risk.

FUTURE SCOPE

Customization: Tailoring credit risk models to specific industries, regions, or customer segments will become common. Customized models will provide more accurate risk assessments.

Collaboration: Collaboration between financial institutions, fintech companies, and data providers will lead to the development of more robust and comprehensive credit risk models.

Scenario Analysis: Scenario-based modeling will gain prominence for assessing credit risk under different economic conditions. This will help institutions prepare for unexpected economic events.

Different models like Decision tree,XGBoost,etc can be used to improve the accuracies

Credit risk modeling will become more dynamic, data-driven, and adaptable to changing economic conditions

THANK YOU!