In this work I carried out a small analysis of loans from people who paid and did not pay the loan.I used 7 machine learning algorithm for analysis.

- Detailed data description of Credit Risk dataset:
  - person_age : Age of person
  - person_income Annual : Income per year
  - person_home_ownership : Home ownership
  - person_emp_length : Employment length (in years)
  - loan_intent : intention of loan
  - loan_grade : Loan grade
  - loan_amnt : Loan amount
  - loan_int_rate : Interest rate
  - loan_status : Loan status (0 is non default(payer) ,1 is default(non-payer)
  - loan_percent_income : Percent income
  - cb_person_default_on_file : Historical default
  - cb_preson_cred_hist_length : Credit history length

- Target Variable : loan_status

In [2]:
```
# import necessary liberaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# read the data
df = pd.read_csv("credit_risk_dataset.csv")
df.head()
```

Out[3]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_sta
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	

In [4]:
```
df.shape
```

Out[4]: (32581, 12)

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   person_age                  32581 non-null  int64  
 1   person_income               32581 non-null  int64  
 2   person_home_ownership       32581 non-null  object 
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  object 
 5   loan_grade                  32581 non-null  object 
 6   loan_amnt                   32581 non-null  int64  
 7   loan_int_rate               29465 non-null  float64
 8   loan_status                 32581 non-null  int64  
 9   loan_percent_income         32581 non-null  float64
 10  cb_person_default_on_file   32581 non-null  object 
 11  cb_person_cred_hist_length  32581 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

Data Cleaning

In [6]:
```
# check null values
df.isna().sum()
```

Out[6]:
```
person_age                      0
person_income                   0
person_home_ownership           0
person_emp_length             895
loan_intent                     0
loan_grade                      0
loan_amnt                       0
loan_int_rate                3116
loan_status                     0
loan_percent_income             0
cb_person_default_on_file       0
cb_person_cred_hist_length      0
dtype: int64
```

In [7]:
```
#check the percentage of missing values
(df.isna().sum()/len(df))*100
```

Out[7]:
```
person_age                    0.000000
person_income                 0.000000
person_home_ownership         0.000000
person_emp_length             2.747000
loan_intent                   0.000000
loan_grade                    0.000000
loan_amnt                     0.000000
loan_int_rate                 9.563856
loan_status                   0.000000
loan_percent_income           0.000000
cb_person_default_on_file     0.000000
cb_person_cred_hist_length    0.000000
dtype: float64
```

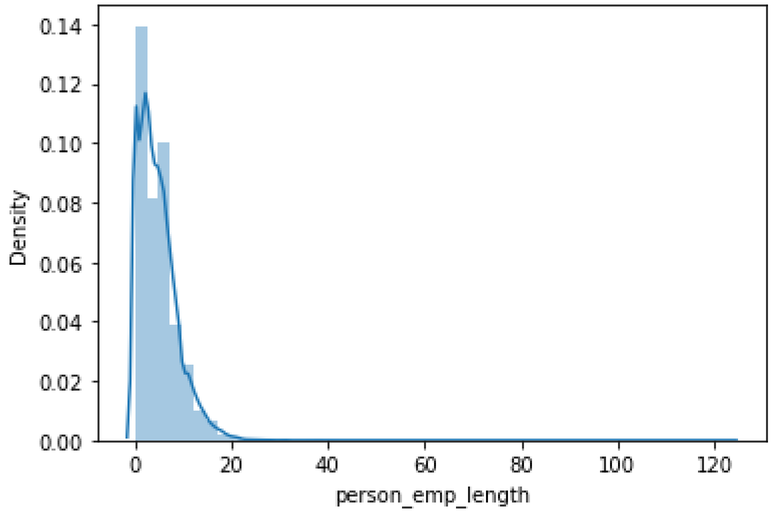
Percantage of missing values in both the columns are under 30, so can be treated with appropriate technique, after exploring the distribution

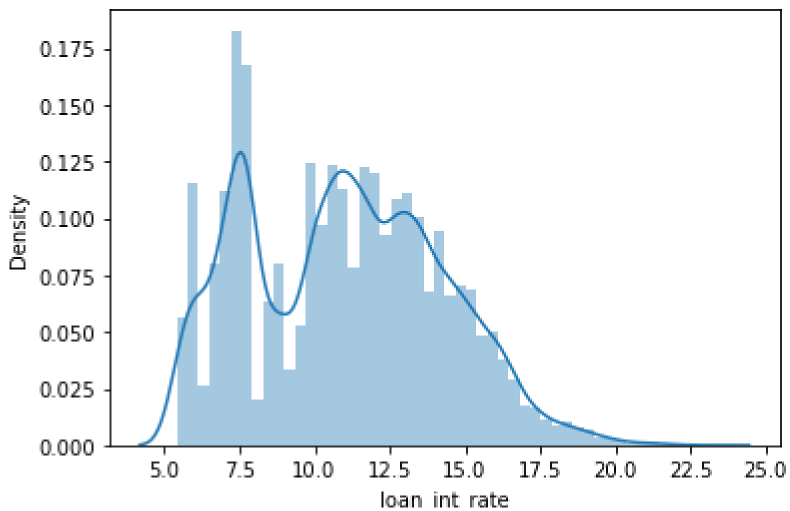
In [8]:

```
#Check distribution of both columns
def dist_plot(data,col):
     plt.figure()
     sns.distplot(data[col].dropna())
```

In [9]:
```
dist_plot(df,'person_emp_length'), dist_plot(df,'loan_int_rate')
```

Out[9]: (None, None)





Both columns are not following a normal distribution. So preferably we can use 'median' to replace the missing values

In [10]:
```
# fill null value by median
df['person_emp_length'].median()
```

Out[10]:
```
4.0
```

In [11]:
```
df['person_emp_length'].fillna('4.0',inplace=True)
```

In [12]:
```
df['loan_int_rate'].median()
```

Out[12]:
```
10.99
```

In [13]:
```
df['loan_int_rate'].fillna('10.99',inplace=True)
```

In [14]:
```
print('Null values : ',df.isna().sum().sum())
```

```
Null values :  0
```

Here we fill the null values by median bt it may affect on the accuracy of model, to see the differece we will check the accuaracy of model in both ways. by filling the values or by dropping the values

In [15]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [16]:
```
# check multicolinearity
def calculate_vif(X):
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return(vif)
```

In [17]:
```
X = df.select_dtypes(exclude='object').dropna()
calculate_vif(X)
```

Out[17]:

	variables	VIF
0	person_age	13.446412
1	person_income	3.198992
2	loan_amnt	6.873545
3	loan_status	1.528453
4	loan_percent_income	7.904940
5	cb_person_cred_hist_length	6.309861

Age has high value for VIF, which indicates that it can be predicted by other independent variables in the dataset,So the column 'age' can be dropped to reduce the feature set

https://www.investopedia.com/terms/v/variance-inflation-factor.asp# (https://www.investopedia.com/terms/v/variance-inflation-factor.asp#)

In [18]:
```
# check outliers in data
df['person_age'].max()
```

Out[18]: 144

In [19]:
```
age = df[df['person_age'] > 100]
age
```

Out[19]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loar
81	144	250000	RENT	4.0	VENTURE	C	4800	13.57	
183	144	200000	MORTGAGE	4.0	EDUCATION	B	6000	11.86	
575	123	80004	RENT	2.0	EDUCATION	B	20400	10.25	
747	123	78000	RENT	7.0	VENTURE	B	20000	10.99	
32297	144	6000000	MORTGAGE	12.0	PERSONAL	C	5000	12.73	

In [20]:
```
#drop the outliers
df = df[df['person_age'] < 100].reset_index(drop=True)
```

In [21]:
```
df
```

Out[21]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_
0	22	59000	RENT	123.0	PERSONAL	D	35000	1
1	21	9600	OWN	5.0	EDUCATION	B	1000	1
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	1
3	23	65500	RENT	4.0	MEDICAL	C	35000	1
4	24	54400	RENT	8.0	MEDICAL	C	35000	1
...	...	...	...	...	...	...	...	
32571	57	53000	MORTGAGE	1.0	PERSONAL	C	5800	1
32572	54	120000	MORTGAGE	4.0	PERSONAL	A	17625	
32573	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000	1
32574	56	150000	MORTGAGE	5.0	PERSONAL	B	15000	1
32575	66	42000	RENT	2.0	MEDICAL	B	6475	

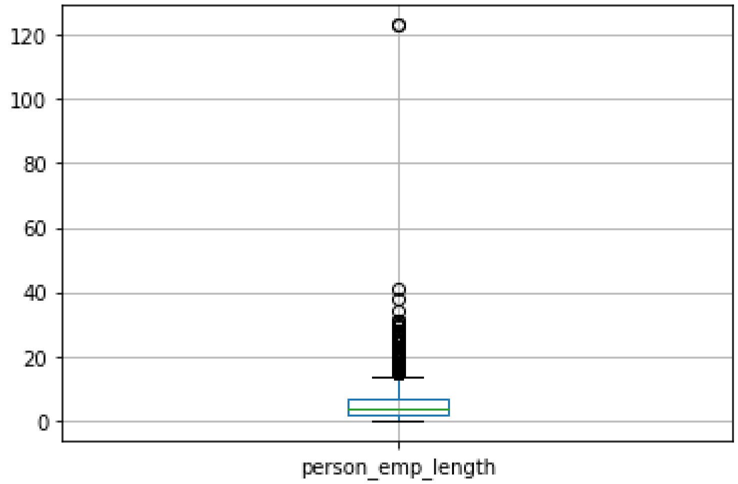
32576 rows × 12 columns

Five rows of outliers are dropped

In [22]:
```
# df[df['person_emp_length'] > 100]
```

In [23]:
```
# check outlier using box plot
df['person_emp_length'] = df.person_emp_length.astype("float")
```

In [24]:
```
df.boxplot(column='person_emp_length')
plt.show()
```



We can clearly see that there is outlier, we should remove that outlier for better accuracy result.

```python
In [25]: df = df[df['person_emp_length'] < 100].reset_index(drop=True)
df
```

Out[25]:

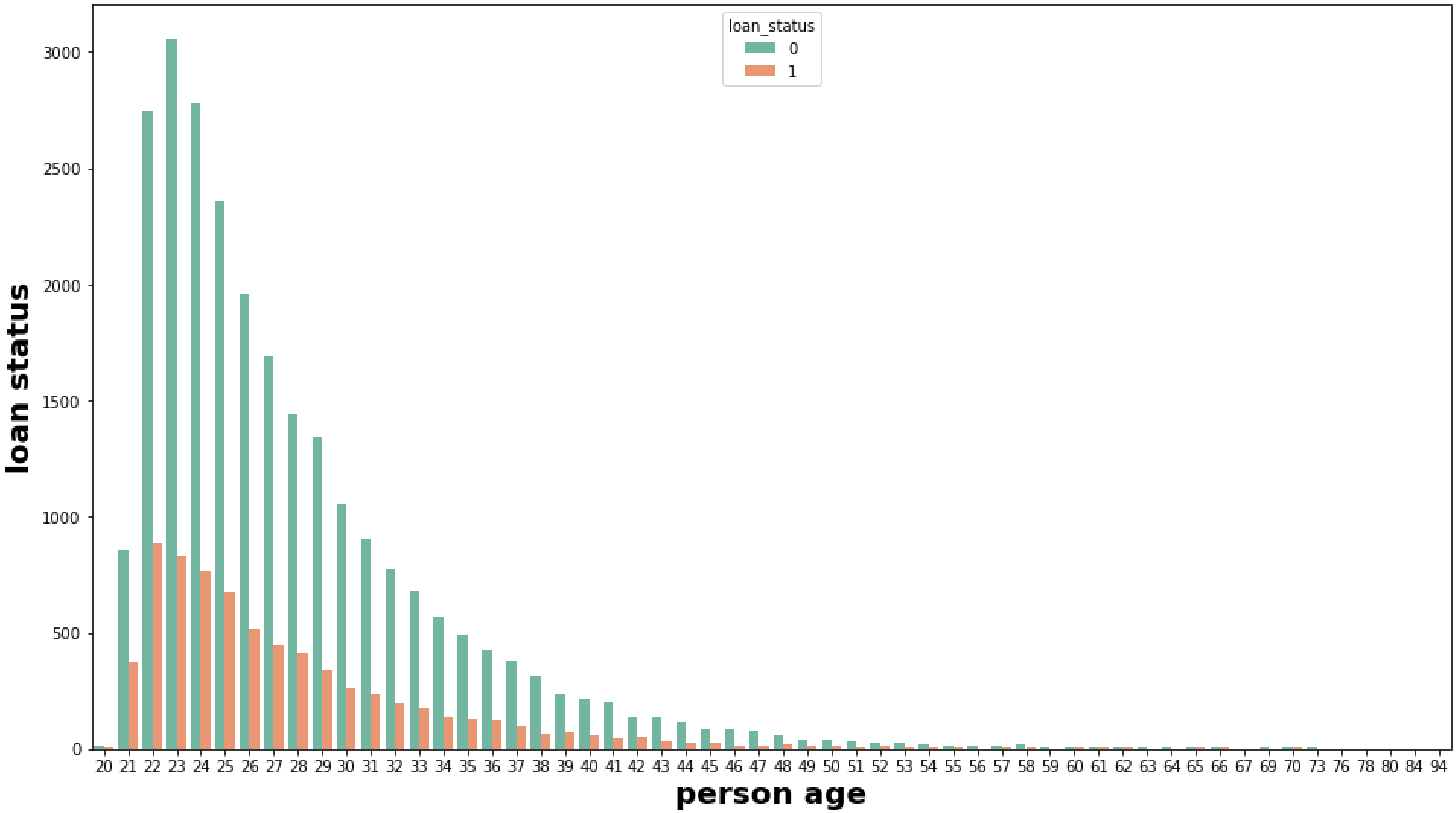
	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_
0	21	9600	OWN	5.0	EDUCATION	B	1000	1
1	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	1
2	23	65500	RENT	4.0	MEDICAL	C	35000	1
3	24	54400	RENT	8.0	MEDICAL	C	35000	1
4	21	9900	OWN	2.0	VENTURE	A	2500	
...	...	...	...	...	...	...	...	
32569	57	53000	MORTGAGE	1.0	PERSONAL	C	5800	1
32570	54	120000	MORTGAGE	4.0	PERSONAL	A	17625	
32571	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000	1
32572	56	150000	MORTGAGE	5.0	PERSONAL	B	15000	1
32573	66	42000	RENT	2.0	MEDICAL	B	6475	

32574 rows × 12 columns

Two rows of outlier dropped

Exploratory Data Analysis

In [26]:
```
plt.figure(figsize=[16,9])
sns.countplot(x = 'person_age', hue= 'loan_status', data=df, palette='Set2' )
plt.xlabel("person age", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.ylabel("loan status", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.show()
```



- 0 : non-default
- 1 : default

We can observe that people who are younger have a tendency of not to pay the loan, The greatest default is among the youngest.

In [27]:
```
import plotly.express as px
```

- https://www.analyticsvidhya.com/blog/2021/10/interactive-plots-in-python-with-plotly-a-complete-guide/ (https://www.analyticsvidhya.com/blog/2021/10/interactive-plots-in-python-with-plotly-a-complete-guide/)

In [28]:
```
defaulter = df[df['loan_status'] == 1]
non_defaulter = df[df['loan_status'] == 0]
```

In [29]:
```
# Loan intention of loan defaulter
fig_1 = px.histogram(defaulter, x='loan_intent',color='loan_intent',template='plotly_dark')
fig_1.show()
```

- An interesting fact is that the people most likely to default on the loan are the youngest, and the biggest expense on loans is for medical expenses.One of the reasons may be that many do not have health insurance and, in an emergency, end up borrowing money.

- Debt consolidation is the act of taking out a single loan to pay off multiple debts.
- https://www.investopedia.com/terms/d/debtconsolidation.asp (https://www.investopedia.com/terms/d/debtconsolidation.asp)

In [30]:
```
# Loan intention of non-defaulters
fig_2 = px.histogram(non_defaulter, x='loan_intent',color='loan_intent', template='plotly_dark')
fig_2.show()
```

- In this graph we see that non-defaulter used the amount for education purpose, the education factor ends up being an

interesting factor for the payment

- Venture debt refers to a financing arrangement wherein companies in their start-up, or early phase are backed by venture capital.

- [https://corporatefinanceinstitute.com/resources/knowledge/finance/venture-debt/ (https://corporatefinanceinstitute.com/resources/knowledge/finance/venture-debt/)](https://corporatefinanceinstitute.com/resources/knowledge/finance/venture-debt/)

In [31]:
```
fig_3 = px.histogram(defaulter,x='person_home_ownership',color='person_home_ownership',template='plotly_dark')
fig_3.show()
```

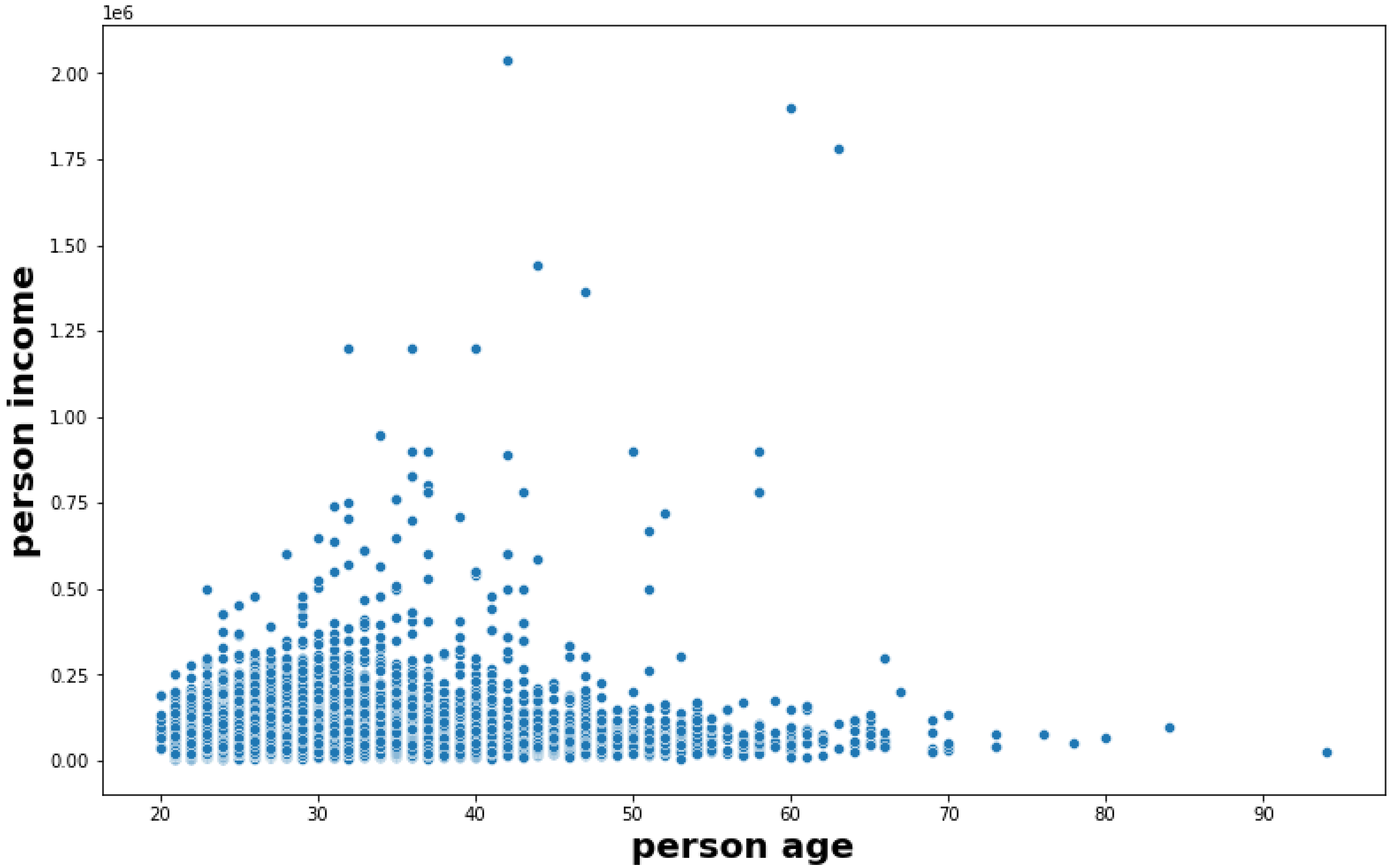
In [32]:
```
fig_4 = px.histogram(non_defaulter,x='person_home_ownership',color='person_home_ownership',template='plotly_dark
fig_4.show()
```

In both of the cases number of renters are more

The number of mortgage is much higher in non-defaulter when related to rent.

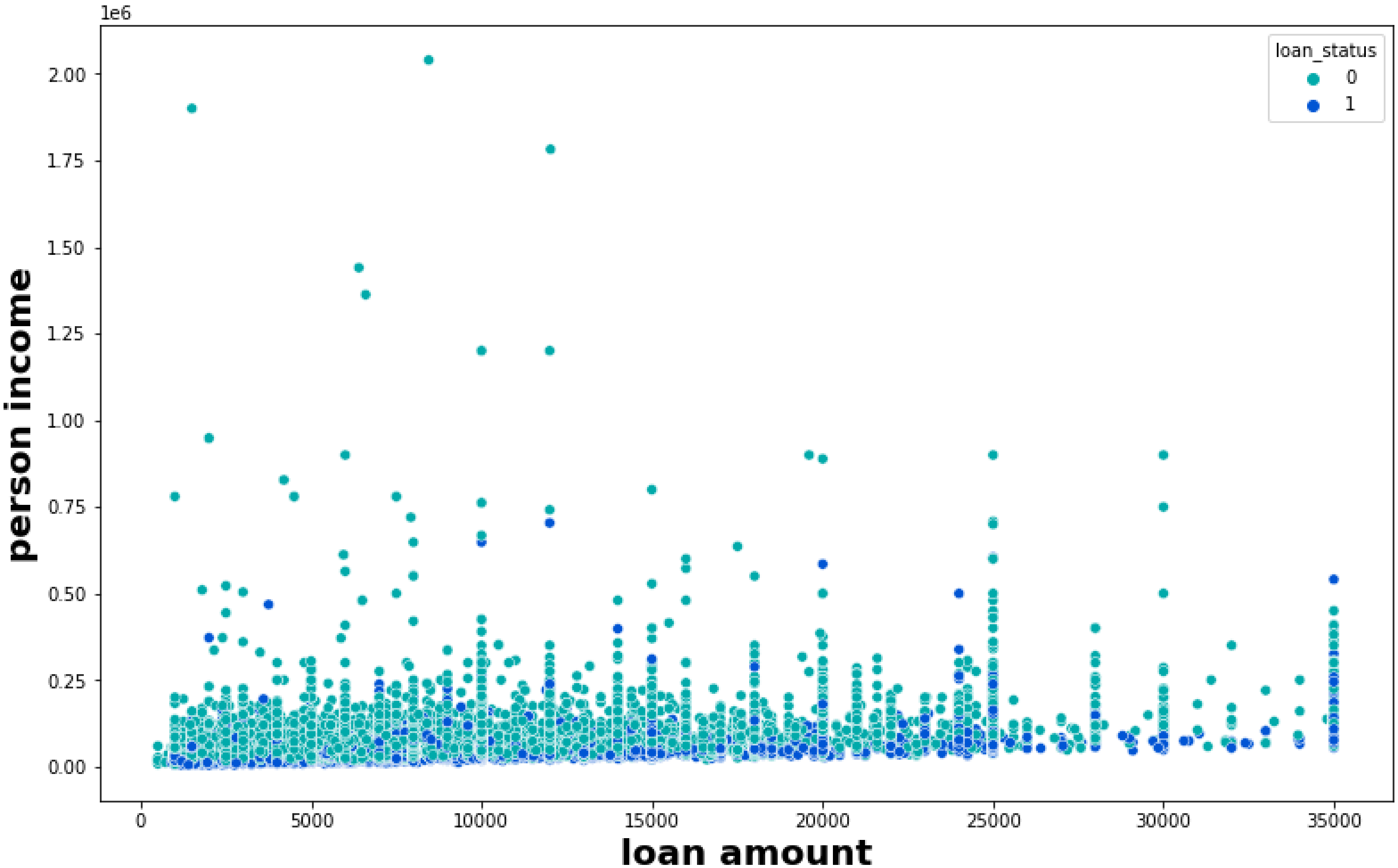
- [https://www.rocketmortgage.com/learn/what-is-a-mortgage (https://www.rocketmortgage.com/learn/what-is-a-mortgage)](https://www.rocketmortgage.com/learn/what-is-a-mortgage)

In [33]:
```
plt.figure(figsize=(13,8))
sns.scatterplot(data=df,x='person_age',y='person_income',)
plt.xlabel("person age", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.ylabel("person income", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.show()
```



In [34]:

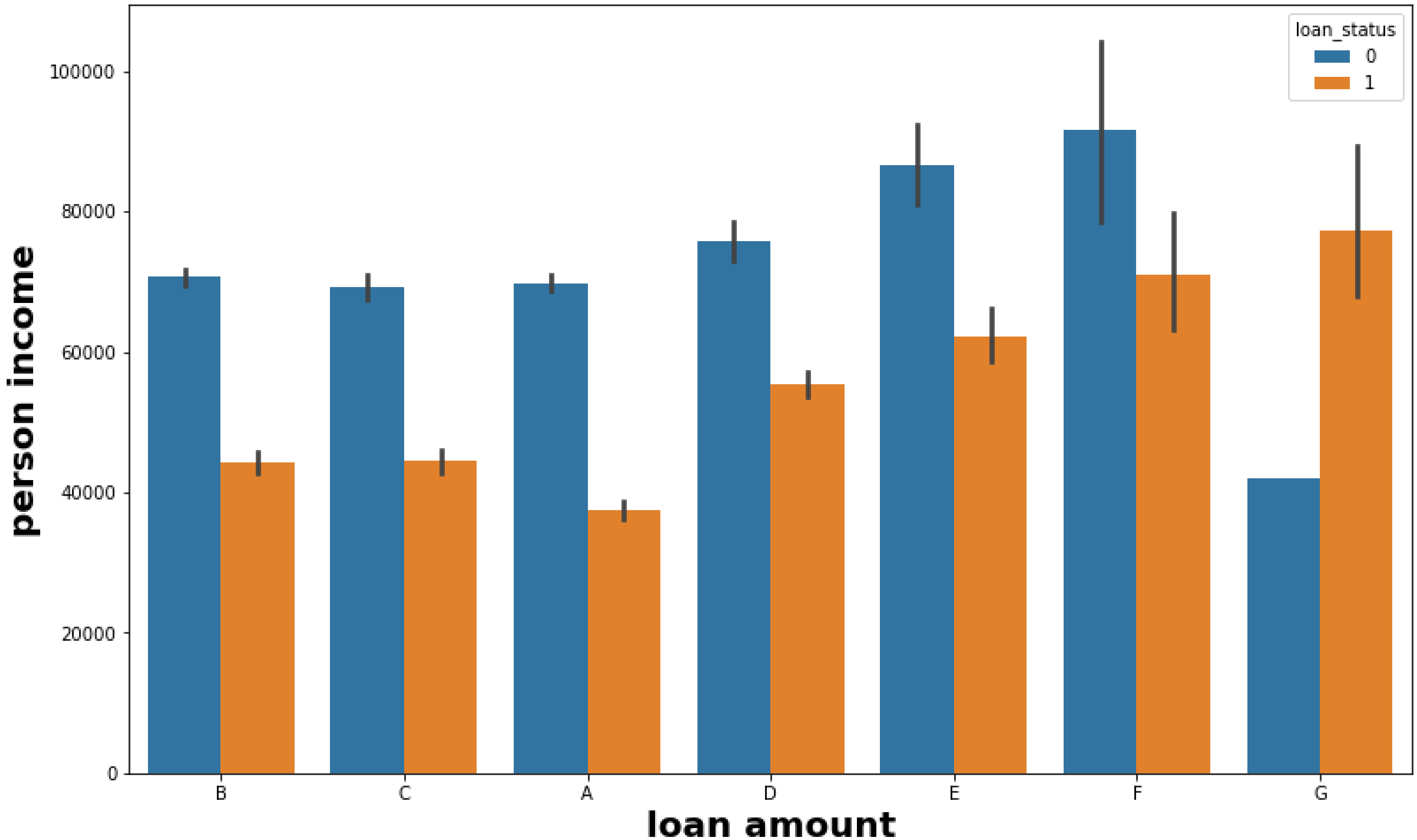
```
plt.figure(figsize=(13,8))
sns.scatterplot(data=df,x='loan_amnt',y='person_income',hue='loan_status',palette='winter_r')
plt.xlabel("loan amount", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.ylabel("person income", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.show()
```



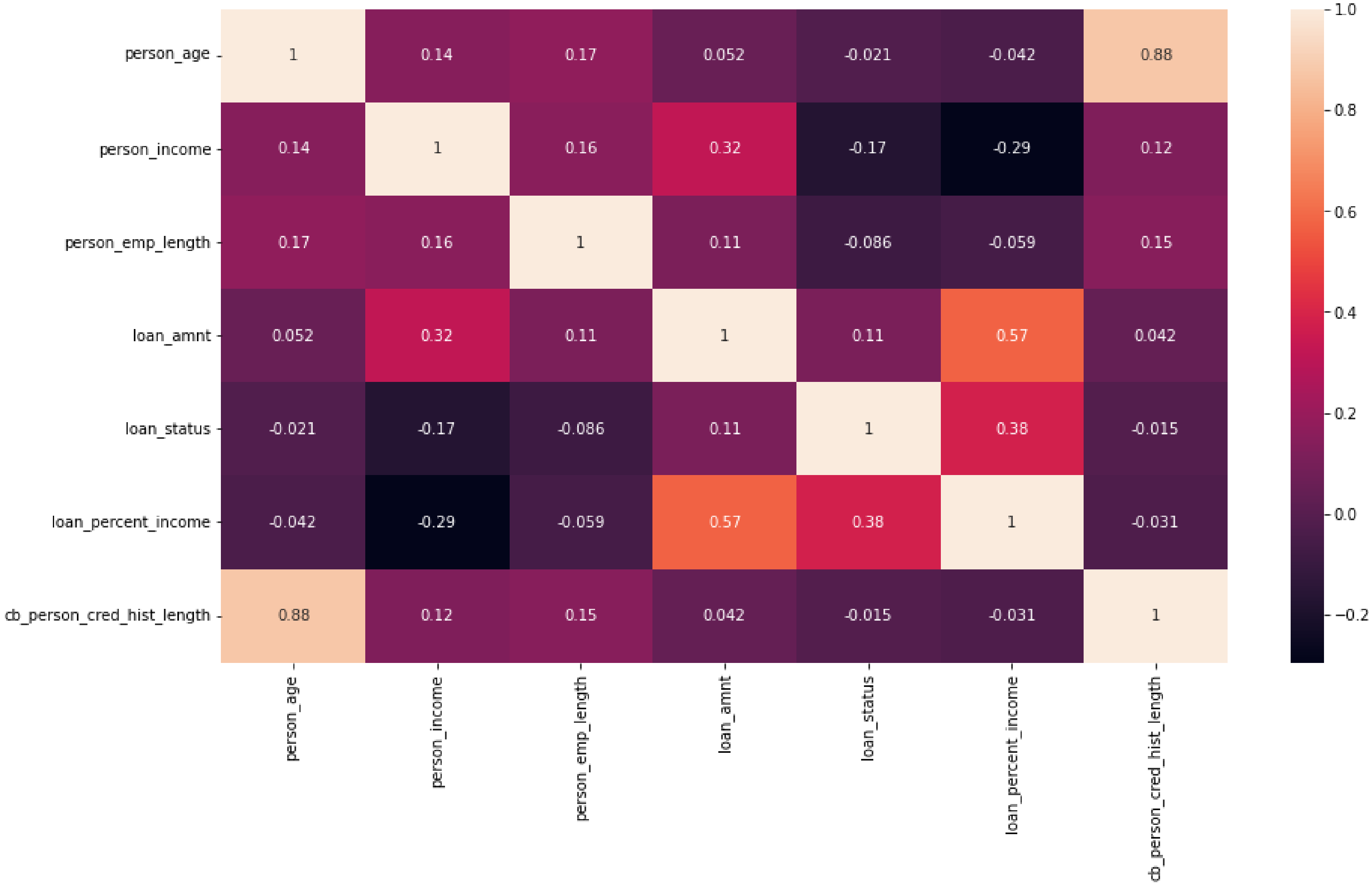
- inference:
  - here, we can see that the person whoes income is more than 75k are non defaulters.
  - the person whoes income is less than 25k have more number of defaulter

In [35]:

```
plt.figure(figsize=(13,8))
sns.barplot(data=df,y='person_income',x='loan_grade',hue='loan_status')
plt.xlabel("loan amount", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.ylabel("person income", fontdict={'fontsize':20, 'color':'Black', 'fontweight':'bold'})
plt.show()
```



In [36]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(), annot= True)
plt.show()
```



Model Building

In [37]:
```
# Splitting of data
X = df.drop(columns='loan_status')
y= df['loan_status']
```

In [38]:
```
# encode the categorical data
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
one_hot_encoder = ColumnTransformer(transformers=[('OneHot', OneHotEncoder(), [0,2,4,5,9,10])], remainder='passt
X = one_hot_encoder.fit_transform(X).toarray()
```

In [39]:
```
# Escalation of values
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [40]:
```
# Division of data into training and testing
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 = 0)
```

In [41]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(24430, 109)
(8144, 109)
(24430,)
(8144,)
```

Naive Bayes

In [42]:
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
predictions = naive_bayes.predict(X_test)
predictions
```

Out[42]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [43]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print('accuracy of naive bayes algorithm :%.2f%%' % (accuracy_score(y_test, predictions)*100))
```

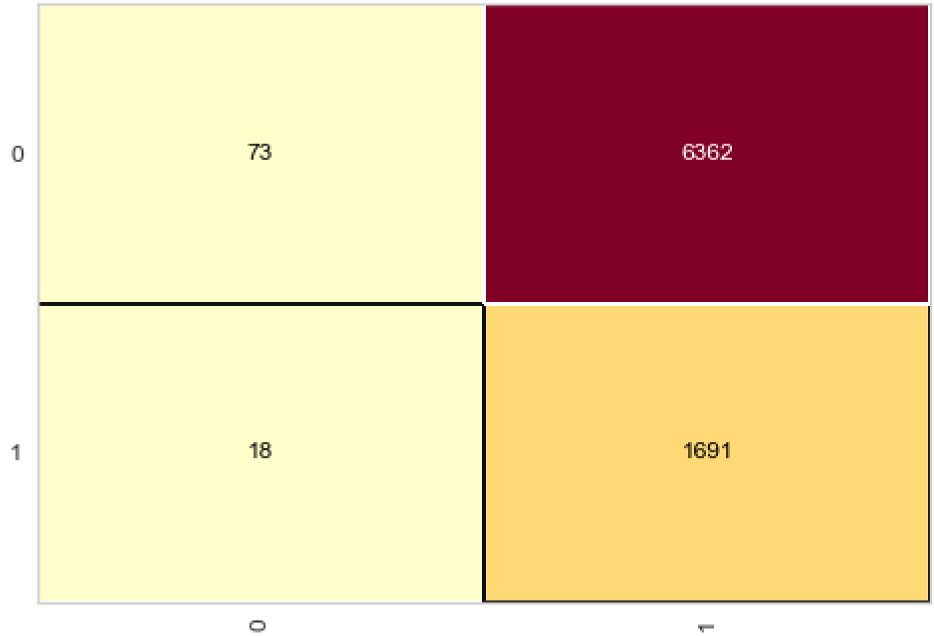
```
accuracy of naive bayes algorithm :21.66%
```

In [44]:
```
confusion_matrix(y_test, predictions)
```

Out[44]:
```
array([[  73, 6362],
       [  18, 1691]], dtype=int64)
```

In [45]:
```python
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(naive_bayes)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[45]: 0.21660117878192534



In [46]:
```python
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.80      0.01      0.02      6435
           1       0.21      0.99      0.35      1709

    accuracy                           0.22      8144
   macro avg       0.51      0.50      0.18      8144
weighted avg       0.68      0.22      0.09      8144
```

One of the reasons for the bad result of this algorithm is, the fact that the combination of characteristics, each pair is independent, which is not always true, contributing to the bad performance of this algorithm.

Decision Trees

In [47]:
```
from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier(criterion='entropy')
dec_tree.fit(X_train, y_train)
```

Out[47]:
```
DecisionTreeClassifier(criterion='entropy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

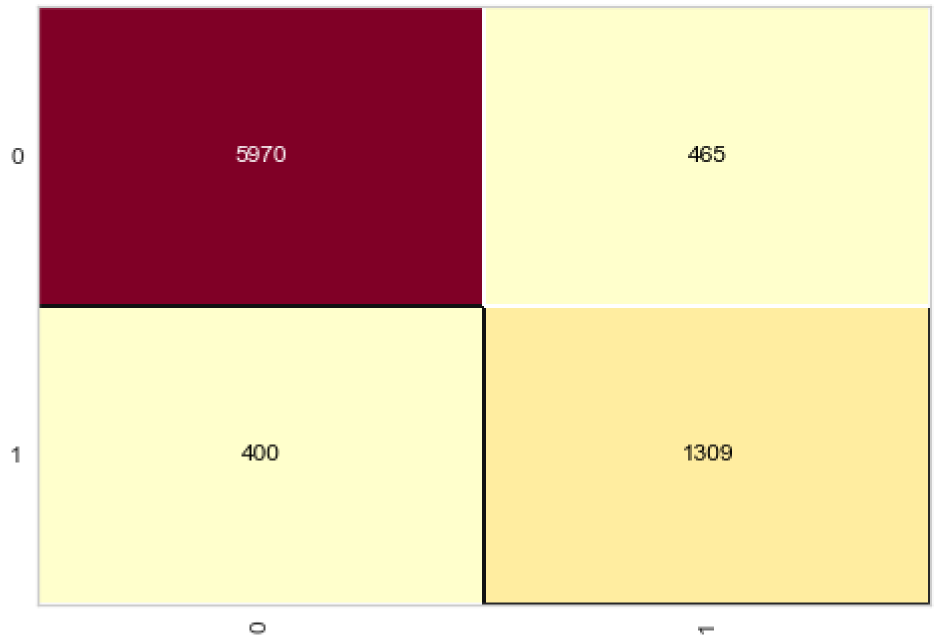
In [48]:
```
y_pred = dec_tree.predict(X_test)
```

In [49]:
```
print('accuracy of decision tree algorithm :%.2f%%' % (accuracy_score(y_test, y_pred)*100))
```

```
accuracy of decision tree algorithm :89.38%
```

In [50]:
```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(dec_tree)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[50]: 0.893786836935167



In [51]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.93      0.93      6435
           1       0.74      0.77      0.75      1709

    accuracy                           0.89      8144
   macro avg       0.84      0.85      0.84      8144
weighted avg       0.90      0.89      0.89      8144
```

XGBoost

In [52]:
```
from xgboost import XGBClassifier
```

In [53]:
```
xgboost = XGBClassifier(n_estimators=1000, max_depth=15, learning_rate=0.05, objective='binary:logistic', random
xgboost.fit(X_train, y_train)
```

Out[53]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=15, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=1000,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=10,
              reg_alpha=0, reg_lambda=1, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

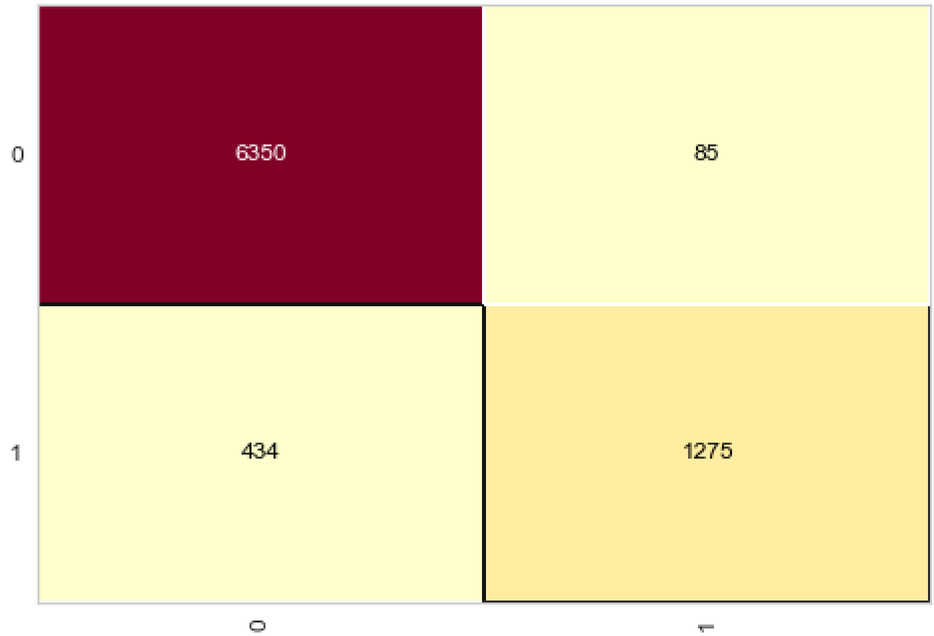
In [54]:
```
y_pred1 = xgboost.predict(X_test)
```

In [55]:
```
print('accuracy of xgboost algorithm :%.2f%%' % (accuracy_score(y_test, y_pred1)*100))
```

```
accuracy of xgboost algorithm :93.63%
```

In [56]:
```
cm = ConfusionMatrix(xgboost)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[56]: 0.9362721021611002



In [57]:
```
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96      6435
           1       0.94      0.75      0.83      1709

    accuracy                           0.94      8144
   macro avg       0.94      0.87      0.90      8144
weighted avg       0.94      0.94      0.93      8144
```

Random Forest

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=40, criterion='entropy', random_state = 0)
random_forest.fit(X_train, y_train)
```

Out[58]:
```
RandomForestClassifier(criterion='entropy', n_estimators=40, random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

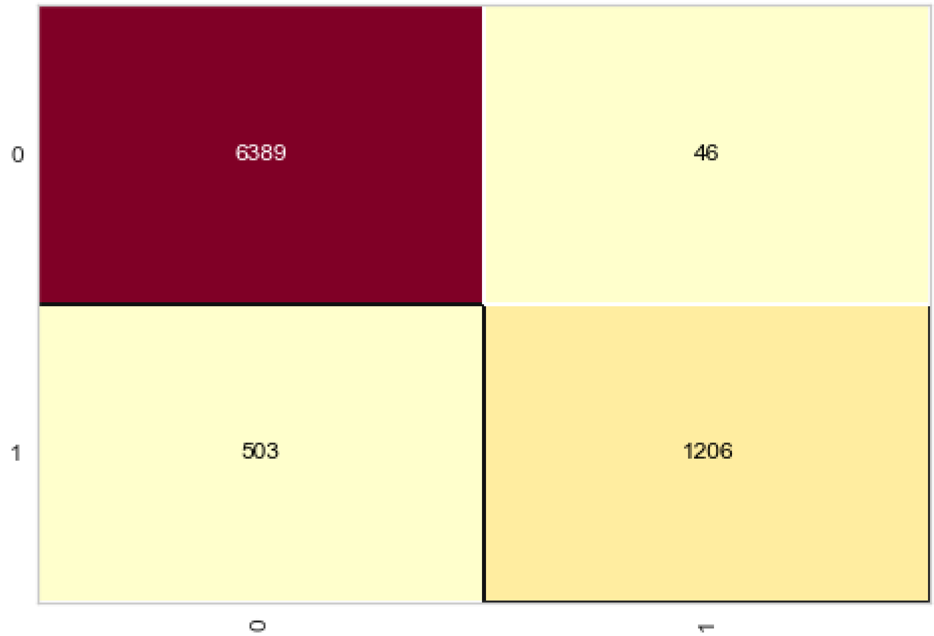
In [59]:
```
y_pred_rf = random_forest.predict(X_test)
```

In [60]:
```
print('accuracy of random forest algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_rf)*100))
```

```
accuracy of random forest algorithm :93.26%
```

In [61]:
```
cm = ConfusionMatrix(random_forest)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[61]:
```
0.9325884086444007
```



In [62]:
```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      6435
           1       0.96      0.71      0.81      1709

    accuracy                           0.93      8144
   macro avg       0.95      0.85      0.89      8144
weighted avg       0.93      0.93      0.93      8144
```

Instance-based learning - knn

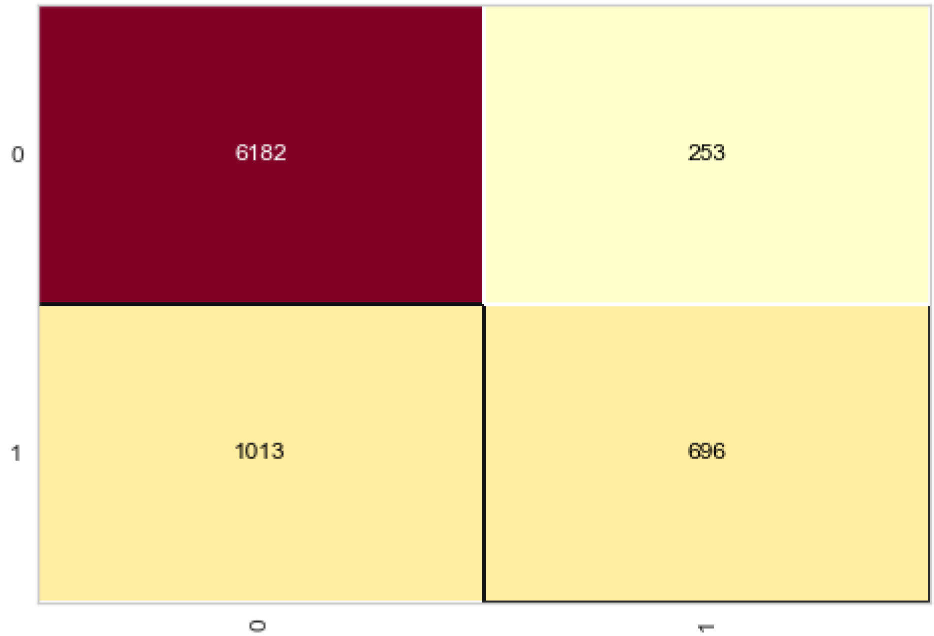
In [63]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p = 2)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

In [64]:
```
print('accuracy of knn algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_knn)*100))
```

```
accuracy of knn algorithm :84.45%
```

In [65]:
```
cm = ConfusionMatrix(knn)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[65]: 0.8445481335952849



In [66]:
```
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      6435
           1       0.73      0.41      0.52      1709

    accuracy                           0.84      8144
   macro avg       0.80      0.68      0.72      8144
weighted avg       0.83      0.84      0.83      8144
```

SVM

In [67]:
```
from sklearn.svm import SVC
svm = SVC(kernel='rbf', random_state=1, C = 2.0)
svm.fit(X_train, y_train)
```

Out[67]:
```
SVC(C=2.0, random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

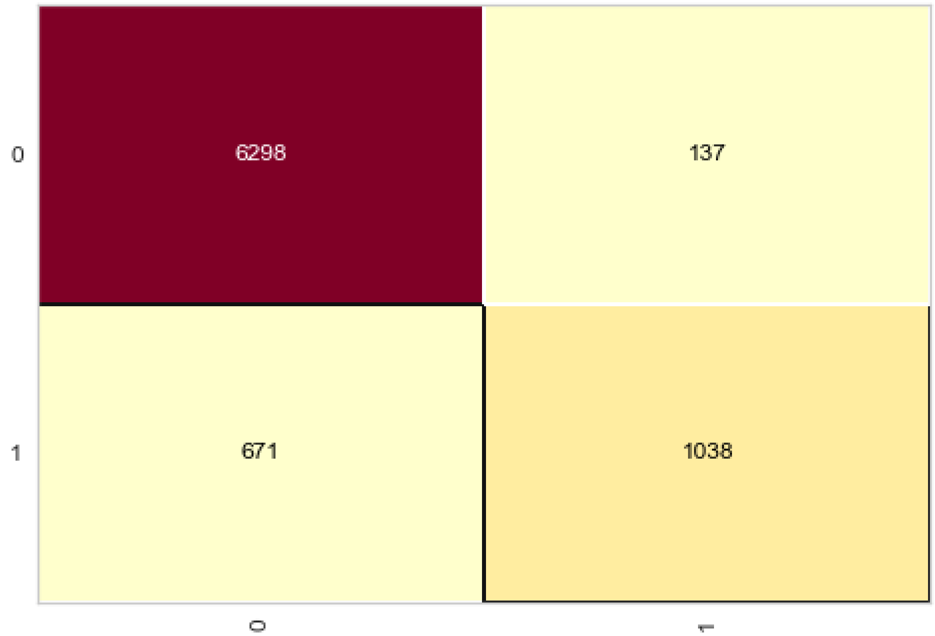
In [68]:
```
y_pred_svm = svm.predict(X_test)
```

In [69]:
```
print('accuracy of svm algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_svm)*100))
```

```
accuracy of svm algorithm :90.08%
```

In [70]:
```
cm = ConfusionMatrix(svm)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[70]: 0.9007858546168959



In [71]:
```
print(classification_report(y_test, y_pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      6435
           1       0.88      0.61      0.72      1709

    accuracy                           0.90      8144
   macro avg       0.89      0.79      0.83      8144
weighted avg       0.90      0.90      0.89      8144
```

Logistic Regression

In [72]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[72]:
```
LogisticRegression(random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

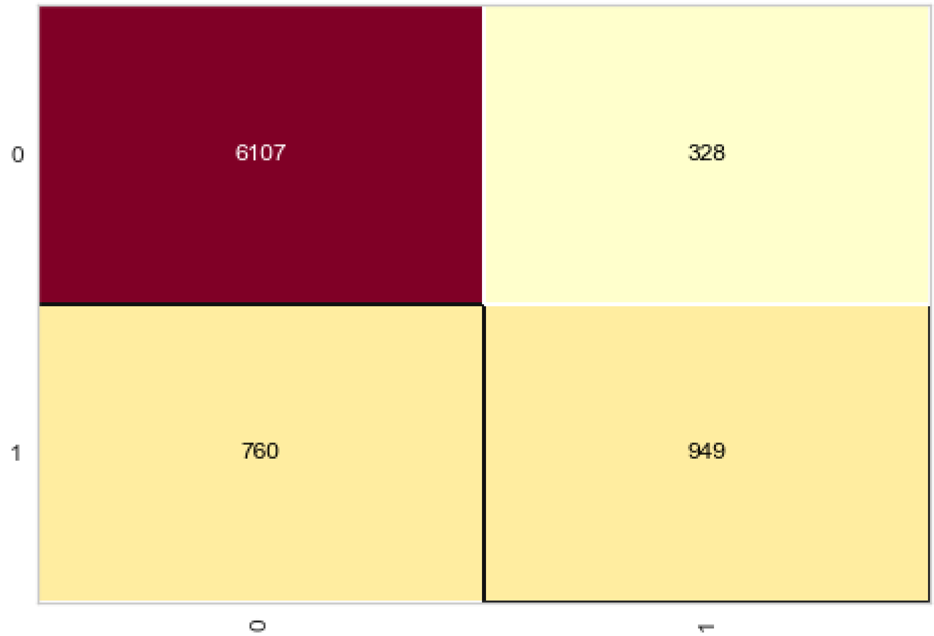
In [73]:
```
y_pred_reg = classifier.predict(X_test)
```

In [74]:
```
print('accuracy of logistic regression algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_reg)*100))
```

```
accuracy of logistic regression algorithm :86.64%
```

In [75]:
```
cm = ConfusionMatrix(classifier)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Out[75]: 0.8664047151277013



In [76]:
```
print(classification_report(y_test, y_pred_reg))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      6435
           1       0.74      0.56      0.64      1709

    accuracy                           0.87      8144
   macro avg       0.82      0.75      0.78      8144
weighted avg       0.86      0.87      0.86      8144
```

Check the accuracy by dropping the null values

We are performing this operation for our better understanding

In [77]:
```
# read the data
df = pd.read_csv("credit_risk_dataset.csv")
df.head()
```

Out[77]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_sta
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	

In [78]:
```
df.shape
```

Out[78]: (32581, 12)

In [79]:
```python
print('Total null values : ',df.isna().sum().sum())
```

```
Total null values :  4011
```

In [80]:
```python
# dropping null values
df1 = df.dropna(inplace=True)
```

In [81]:
```python
df
```

Out[81]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_
0	22	59000	RENT	123.0	PERSONAL	D	35000	1
1	21	9600	OWN	5.0	EDUCATION	B	1000	1
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	1
3	23	65500	RENT	4.0	MEDICAL	C	35000	1
4	24	54400	RENT	8.0	MEDICAL	C	35000	1
...	...	...	...	...	...	...	...	
32576	57	53000	MORTGAGE	1.0	PERSONAL	C	5800	1
32577	54	120000	MORTGAGE	4.0	PERSONAL	A	17625	
32578	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000	1
32579	56	150000	MORTGAGE	5.0	PERSONAL	B	15000	1
32580	66	42000	RENT	2.0	MEDICAL	B	6475	

28638 rows × 12 columns

In [82]:
```python
# dropping outliers
df = df[df['person_age'] < 100].reset_index(drop=True)
```

In [83]:
```
df = df[df['person_emp_length'] < 100].reset_index(drop=True)
df
```

Out[83]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_
0	21	9600	OWN	5.0	EDUCATION	B	1000	1
1	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	1
2	23	65500	RENT	4.0	MEDICAL	C	35000	1
3	24	54400	RENT	8.0	MEDICAL	C	35000	1
4	21	9900	OWN	2.0	VENTURE	A	2500	
...	...	...	...	...	...	...	...	
28627	57	53000	MORTGAGE	1.0	PERSONAL	C	5800	1
28628	54	120000	MORTGAGE	4.0	PERSONAL	A	17625	
28629	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000	1
28630	56	150000	MORTGAGE	5.0	PERSONAL	B	15000	1
28631	66	42000	RENT	2.0	MEDICAL	B	6475	

28632 rows × 12 columns

In [84]:
```
# Splitting of data
X = df.drop(columns='loan_status')
y= df['loan_status']
```

In [85]:
```
# encode the categorical data
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
one_hot_encoder = ColumnTransformer(transformers=[('OneHot', OneHotEncoder(), [0,2,4,5,9,10])], remainder='passt
X = one_hot_encoder.fit_transform(X).toarray()
```

In [86]:
```
# Escalation of values
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [87]:
```
# Division of data into training and testing
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 = 0)
```

In [88]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(21474, 108)
(7158, 108)
(21474,)
(7158,)
```

Naive Bayes

In [89]:
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
predictions = naive_bayes.predict(X_test)

predictions = naive_bayes.predict(X_test)

print('accuracy of naive bayes algorithm :%.2f%%' % (accuracy_score(y_test, predictions)*100))


print(classification_report(y_test, predictions))
```

```
accuracy of naive bayes algorithm :21.63%
              precision    recall  f1-score   support

           0       0.78      0.01      0.02      5651
           1       0.21      0.99      0.35      1507

    accuracy                           0.22      7158
   macro avg       0.50      0.50      0.18      7158
weighted avg       0.66      0.22      0.09      7158
```

Decision Tree

In [90]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier(criterion='entropy')
dec_tree.fit(X_train, y_train)

y_pred = dec_tree.predict(X_test)

print('accuracy of decision tree algorithm :%.2f%%' % (accuracy_score(y_test, y_pred)*100))

print(classification_report(y_test, y_pred))
```

```
accuracy of decision tree algorithm :90.16%
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      5651
           1       0.76      0.77      0.77      1507

    accuracy                           0.90      7158
   macro avg       0.85      0.85      0.85      7158
weighted avg       0.90      0.90      0.90      7158
```

Radom Forest

In [91]:
```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=40, criterion='entropy', random_state = 0)
random_forest.fit(X_train, y_train)

y_pred_rf = random_forest.predict(X_test)

print('accuracy of random forest algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_rf)*100))
print(classification_report(y_test, y_pred_rf))
```

```
accuracy of random forest algorithm :93.55%
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      5651
           1       0.97      0.72      0.82      1507

    accuracy                           0.94      7158
   macro avg       0.95      0.86      0.89      7158
weighted avg       0.94      0.94      0.93      7158
```

Knn

In [92]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p = 2)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

print('accuracy of knn algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_knn)*100))

print(classification_report(y_test, y_pred_knn))
```

```
accuracy of knn algorithm :84.51%
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      5651
           1       0.73      0.42      0.53      1507

    accuracy                           0.85      7158
   macro avg       0.80      0.69      0.72      7158
weighted avg       0.83      0.85      0.83      7158
```

SVM

In [93]:
```
from sklearn.svm import SVC
svm = SVC(kernel='rbf', random_state=1, C = 2.0)
svm.fit(X_train, y_train)

y_pred_svm = svm.predict(X_test)

print('accuracy of svm algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_svm)*100))

print(classification_report(y_test, y_pred_svm))
```

```
accuracy of svm algorithm :89.98%
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      5651
           1       0.89      0.60      0.71      1507

    accuracy                           0.90      7158
   macro avg       0.90      0.79      0.83      7158
weighted avg       0.90      0.90      0.89      7158
```

Logistic Regression

In [94]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

y_pred_reg = classifier.predict(X_test)

print('accuracy of logistic regression algorithm :%.2f%%' % (accuracy_score(y_test, y_pred_reg)*100))

print(classification_report(y_test, y_pred_reg))
```

```
accuracy of logistic regression algorithm :86.78%
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      5651
           1       0.75      0.56      0.64      1507

    accuracy                           0.87      7158
   macro avg       0.82      0.75      0.78      7158
weighted avg       0.86      0.87      0.86      7158
```

XGBoost

In [95]:
```
from xgboost import XGBClassifier
xgboost = XGBClassifier(n_estimators=1000, max_depth=15, learning_rate=0.05, objective='binary:logistic', random
xgboost.fit(X_train, y_train)

y_pred1 = xgboost.predict(X_test)

print('accuracy of xgboost algorithm :%.2f%%' % (accuracy_score(y_test, y_pred1)*100))

print(classification_report(y_test, y_pred1))
```

```
accuracy of xgboost algorithm :93.99%
              precision    recall  f1-score   support

           0       0.94      0.99      0.96      5651
           1       0.95      0.76      0.84      1507

    accuracy                           0.94      7158
   macro avg       0.94      0.87      0.90      7158
weighted avg       0.94      0.94      0.94      7158
```

In [97]:
```python
# Create the comaprison table of both method
from tabulate import tabulate
info = {'algorithm': ['Naive bayes', 'Decision tree', 'Random forest','Knn','SVM','Logistic regression','XGboost
            'accuracy by filing null values ': ['21.66%', '89.38%', '93.26%','84.45%','90.08%','86.64%','93.63%'
            'accuracy by dropping null values': ['21.63%', '90.16%', '93.55%','84.51%','89.98%','86.78%','93.99%
print(tabulate(info, headers='keys', showindex=True, tablefmt='fancy_grid'))
```

	algorithm	accuracy by filing null values	accuracy by dropping null values
0	Naive bayes	21.66%	21.63%
1	Decision tree	89.38%	90.16%
2	Random forest	93.26%	93.55%
3	Knn	84.45%	84.51%
4	SVM	90.08%	89.98%
5	Logistic regression	86.64%	86.78%
6	XGboost	93.63%	93.99%

- Here we can see that, there is slightly difference between the accuracy score of both.so, we can say that filing or dropping the null values doesn't affect more on the accuracy of model.
- we check the accuracy of model in both way by filling and by dropping null values only for our better understanding.

Insight

- The focus of this work was only to analyze and compare the results of the most used algorithms when it comes to credit analysis with a focus on the risk of its concession to non-payers.
- In this work,XGBoost and Random Forest algorithm performed best among the algorithm
- There is slight difference in both of the algorithms ,among them XGBoost perform much better.
- https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/ (https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/)

- https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/ (https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/)

NOTE: These percentages are from data performed at the time of creation of the algorithm, when we save the algorithm it compiles the code again, which may have a slight variation in the results for more or less.