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

In [2]:
```
# import data
loan_default = pd.read_csv("loan_default.csv")
loan_default.head()
```

	ID	year	loan_limit	Gender	approv_in_adv	loan_type	loan_purpose	Credit_Worthiness	open_credit	business_or_commercial	...	credit_type	Credit_Score	co-applicant_credit_type	age	submission_of_application	LT
0	24890	2019	cf	Sex Not Available	nopre	type1	p3	l1	nopc	nobcc	...	EXP	758	CIB	25-34	to_inst	98.1565
1	24891	2019	cf	Male	nopre	type2	p4	l1	nopc	bcc	...	EQUI	552	EXP	55-64	to_inst	Na
2	24892	2019	cf	Male	pre	type1	p3	l1	nopc	nobcc	...	EXP	834	CIB	35-44	to_inst	80.0280
3	24893	2019	cf	Male	nopre	type1	p4	l1	nopc	nobcc	...	EXP	587	CIB	45-54	no_inst	69.3706
4	24894	2019	cf	Joint	pre	type1	p3	l1	nopc	nobcc	...	CRIF	602	EXP	25-34	not_inst	91.8966

5 rows × 34 columns

clean data

In [3]:
```
loan_default.columns
```

Out[3]:
```
Index(['ID', 'year', 'loan_limit', 'Gender', 'approv_in_adv', 'loan_type',
       'loan_purpose', 'Credit_Worthiness', 'open_credit',
       'business_or_commercial', 'loan_amount', 'rate_of_interest',
       'Interest_rate_spread', 'Upfront_charges', 'term', 'Neg_ammortization',
       'interest_only', 'lump_sum', 'property_value',
       'construction_type', 'occupancy_type', 'Secured_by', 'total_units',
       'income', 'credit_type', 'Credit_Score', 'co-applicant_credit_type',
       'age', 'submission_of_application', 'LTV', 'Region', 'Security_type',
       'Status', 'dtir1',
       'dtypes-object'],
      dtype='object')
```

In [4]:
```
# remove columns not needed
col = ['year', 'Interest_rate_spread', 'Upfront_charges', 'term', 'Neg_ammortization',
       'interest_only', 'lump_sum_payment', 'property_value',
       'construction_type', 'occupancy_type', 'Secured_by', 'total_units',
       'income', 'credit_type', 'Credit_Score', 'co-applicant_credit_type',
       'age', 'submission_of_application', 'LTV', 'Security_type', 'dtir1', 'approv_in_adv', 'ID', 'open_credit']
loan_default.drop(columns = col, inplace = True)
```

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

Out[5]:
```
Index(['loan_limit', 'Gender', 'loan_type', 'loan_purpose',
       'Credit_Worthiness', 'business_or_commercial', 'loan_amount',
       'rate_of_interest', 'Region', 'Status',
       'dtypes-object'],
      dtype='object')
```

In [6]:
```
loan_default.head()
```

	loan_limit	Gender	loan_type	loan_purpose	Credit_Worthiness	business_or_commercial	loan_amount	rate_of_interest	Region	Status
0	cf	Sex Not Available	type1	p3	l1	nobcc	116500	NaN	south	1
1	cf	Male	type2	p4	l1	nobcc	206000	NaN	North	1
2	cf	Male	type1	p3	l1	bcc	406000	4.56	south	0
3	cf	Male	type1	p4	l1	nobcc	456500	4.25	North	0
4	cf	Joint	type1	p3	l1	nobcc	696500	4.00	North	0

In [7]:
```
loan_default.isna().sum()
```

Out[7]:
```
loan_limit                3344
Gender                       0
loan_type                    0
loan_purpose               134
Credit_Worthiness            0
business_or_commercial       0
loan_amount                  0
rate_of_interest         36439
Region                       0
Status                       0
dtypes-object            31154
dtype: int64
```

In [8]:
```
loan_default.size
```

Out[8]:
```
1489700
```

In [9]:
```
loan_default.dtypes
```

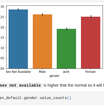
Out[9]:
```
loan_limit                object
Gender                    object
loan_type                 object
loan_purpose              object
Credit_Worthiness         object
business_or_commercial    object
loan_amount                int64
rate_of_interest         float64
Region                    object
Status                     int64
dtypes-object             object
dtype: object
```

In [10]:
```
# set all data to lower case
loan_default.columns = map(str.lower, loan_default.columns)
```

In [11]:
```
loan_default.head()
```

	loan_limit	gender	loan_type	loan_purpose	credit_worthiness	business_or_commercial	loan_amount	rate_of_interest	region	status
0	cf	Sex Not Available	type1	p3	l1	nobcc	116500	NaN	south	1
1	cf	Male	type2	p4	l1	bcc	206000	NaN	North	1
2	cf	Male	type1	p3	l1	nobcc	406000	4.56	south	0
3	cf	Male	type1	p4	l1	nobcc	456500	4.25	North	0
4	cf	Joint	type1	p3	l1	nobcc	696500	4.00	North	0

In [12]:
```
# plot using seaborn (gender and status)
sns.barplot(x = 'gender', y = 'status', data = loan_default)
plt.show()
```



the **sex not available** is higher that the normal so it will be dropped to avoid incorrect prediction

In [13]:
```
loan_default.gender.value_counts()
```

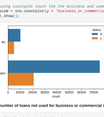
Out[13]:
```
Male                 42346
Joint                37565
Sex Not Available    15783
Female               14454
Name: gender, dtype: int64
```

In [14]:
```
# drop sex not available
loan_default.drop(loan_default.loc[loan_default['gender'] == 'Sex Not Available'].index, inplace = True)
```

In [15]:
```
loan_default.gender.value_counts()
```

Out[15]:
```
Male    42346
Joint   37565
Female  14454
Name: gender, dtype: int64
```

In [16]:
```
# count the numbers of male and female that pays the loan
gen_pay = sns.countplot(x = 'gender', hue = 'status', data = loan_default)
```



both gender default payment of loan with the male higher than female

In [17]:
```
# using countplot count the the business and commercial
bcscom = sns.countplot(y = 'business_or_commercial', hue = 'status', data = loan_default)
plt.show()
```

the number of loans not used for business or commercial is higher than the no of loans used for business or commercial

In [18]:
```
# split the dataset into X and y
X = loan_default.drop('status', axis = 1)
y = loan_default.status
```

In [19]:
```
X.dtypes
```

Out[19]:
```
loan_limit                object
gender                    object
loan_type                 object
loan_purpose              object
credit_worthiness         object
business_or_commercial    object
loan_amount                int64
rate_of_interest         float64
region                    object
dtypes-object             object
dtype: object
```

In [20]:
```
# replace the NaN
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# fill non-numerical values with missing and numerical values with mean
cat_imp = SimpleImputer(strategy='constant', fill_value='missing')
num_imp = SimpleImputer(strategy='mean')

# define columns
cat_features = ['loan_limit', 'gender', 'loan_type',
                'loan_purpose', 'credit_worthiness', 'business_or_commercial', 'region']
num_features = ['loan_amount', 'rate_of_interest']

# create a imputer to fills the NaN values
imputer = ColumnTransformer(
    [('cat_imp', cat_imp, cat_features),
     ('num_imp', num_imp, num_features)]
    )

# transform the data
filled_X = imputer.fit_transform(X)
filled_X
```

Out[20]:
```
array([['cf', 'Male', 'type2', ..., 'North', 206000.0,
        4.56179094519453851],
       ['cf', 'Male', 'type1', ..., 'south', 406000.0, 4.56],
       ['cf', 'Male', 'type1', ..., 'North', 456500.0, 4.25],
       ...,
       ['cf', 'Male', 'type1', ..., 'south', 206500.0, 3.125],
       ['cf', 'Female', 'type1', ..., 'North', 196000.0, 3.5],
       ['cf', 'Female', 'type1', ..., 'North', 406500.0, 4.374]],
      dtype=object)
```

In [21]:
```
col = ['loan_limit', 'gender', 'loan_type', 'loan_purpose', 'credit_worthiness',
       'business_or_commercial', 'loan_amount', 'rate_of_interest']
filled_X = pd.DataFrame(filled_X, columns = col)
filled_X.head()
```

	loan_limit	gender	loan_type	loan_purpose	credit_worthiness	business_or_commercial	region	loan_amount	rate_of_interest
0	cf	Male	type2	p4	l1	nobcc	North	206000.0	4.56179
1	cf	Male	type1	p3	l1	nobcc	south	406000.0	4.56
2	cf	Male	type1	p4	l1	nobcc	North	456500.0	4.25
3	cf	Joint	type1	p3	l1	nobcc	North	696500.0	4.0
4	cf	Joint	type1	p3	l1	nobcc	North	706500.0	3.99

In [22]:
```
# convert non-numerical to numbers using sklearn
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

cat_features = ['loan_limit', 'gender', 'loan_type', 'loan_purpose', 'credit_worthiness',
                'business_or_commercial', 'region']
ohe_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot',
                                  ohe_hot,
                                  cat_features)],
                                remainder='passthrough')
transformed_X = transformer.fit_transform(filled_X)
transformed_X
```

Out[22]:
```
array([[1.0, 0.0, 0.0, ..., 0.0, 206500.0, 4.9417509601045991],
       [1.0, 0.0, 0.0, ..., 0.0, 406000.0, 4.56],
       [1.0, 0.0, 0.0, ..., 0.0, 456500.0, 4.25],
       ...,
       [1.0, 0.0, 0.0, ..., 0.0, 406000.0, 3.125],
       [1.0, 0.0, 0.0, ..., 0.0, 196500.0, 3.5],
       [1.0, 0.0, 0.0, ..., 0.0, 406500.0, 4.375]], dtype=object)
```

In [23]:
```
transformed_X.shape
```

Out[23]:
```
(115011, 24)
```

This is a classification problem as such, according to the map the data is greater than 100k so we will use SGDClassifier

In [24]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split

# set up the random seed
np.random.seed(42)

# the data has been created (transformed_X, and y)

# split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(transformed_X, y, test_size = 0.2)

clf = SGDClassifier()
clf.fit(X_train, y_train)
```

Out[24]:
```
SGDClassifier()
```

In [25]:
```
X_train.shape, X_test.shape
```

Out[25]:
```
((108000, 24), (22203, 24))
```

In [26]:
```
clf.score(X_test, y_test)
```

Out[26]:
```
0.23816061380176554
```

the model did not work let's try another: according to the map next is RBFSampler:

In [27]:
```
from sklearn.kernel_approximation import RBFSampler
from sklearn.linear_model import SGDClassifier

# set up the random seed
np.random.seed(42)

# the data has been created (transformed_X, and y)

# split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(transformed_X, y, test_size = 0.2)

clf = RBFSampler()
X_features = clf.fit_transform(transformed_X)
clf = SGDClassifier()
clf.fit(X_train, y_train)
```

Out[27]:
```
SGDClassifier()
```

In [28]:
```
clf.score(X_test, y_test)
```

Out[28]:
```
0.7661376622881616
```

In [29]:
```
clf.score(X_train, y_train)
```

Out[29]:
```
0.7670370546479588
```

did not work as well so i used RandomForestClassifier:

In [30]:
```
from sklearn.ensemble import RandomForestClassifier

# set up the random seed
np.random.seed(42)

# the data has been created (transformed_X, and y)

# split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(transformed_X, y, test_size = 0.2)

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
```

Out[30]:
```
RandomForestClassifier()
```

In [31]:
```
clf.score(X_train, y_train)
```

Out[31]:
```
1.0
```

In [32]:
```
clf.score(X_test, y_test)
```

Out[32]:
```
0.9999549461013008
```

In [33]:
```
# make prediction
y_preds = clf.predict(X_test)
```

In [34]:
```
y_preds
```

Out[34]:
```
array([0, 0, 0, ..., 1, 0, 1], dtype=int64)
```

In [35]:
```
# compare prediction against truth
clf_df = pd.DataFrame(data = {'actual_values': y_test,
                              'predicted_values': y_preds})

clf_df
```

Out[35]:

	actual_values	predicted_values
19360	0	0
44602	0	0
87570	0	0
110110	0	0
5651	1	1
...	...	...
66128	0	0
47606	0	0
110013	1	1
88638	0	0
86477	1	1

22203 rows × 2 columns

In [36]:
```
loan_prob = clf.predict_proba(X_test)
loan_prob[:5]
```

Out[36]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.]])
```

there is a low difference between the predicted and the actual values

In [37]:
```
# find the accuracy score
from sklearn.model_selection import cross_val_score
np.random.seed(7)

# normal score
clf_score = clf.score(X_test, y_test)

# mean of cross_val_score
clf_cro_score = np.mean(cross_val_score(clf, transformed_X, y))

clf_score, clf_cro_score
```

In [38]:
```
(0.9999549461013008, 0.9999549461013215)
```

In [39]:
```
# using confusion_matrix, evaluate the difference the dataset
from sklearn.metrics import confusion_matrix
confusion_matrix(y_preds, y_test)
```

Out[39]:
```
array([[10014,     0],
       [    1,  12188]], dtype=int64)
```

In [40]:
```
# visualize confusion_matrix using seaborn
loan_mat = confusion_matrix(y_preds, y_test)

def loan_plt(loan_mat):
    fig, ax = plt.subplots(figsize = (6, 5))
    ax = sns.heatmap(loan_mat,
                     annot=True,
                     cmap='Greens', # change colors
                     cbar = False, fmt = 'g')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')

loan_plt(loan_mat)
```

In [41]:
```
# the classification report
from sklearn.metrics import classification_report

loan_class = classification_report(y_preds, y_test)

print(loan_class)
```

Out[41]:
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10014
           1       1.00      1.00      1.00     12188

    accuracy                           1.00     22203
   macro avg       1.00      1.00      1.00     22203
weighted avg       1.00      1.00      1.00     22203
```

the data model is good and predictive

In [42]: