LendingClub Dataset

Dataset Description： The dataset contains the homeloan records released from Lending Club. The problem lenders care is that whether a borrower can repay homeloan and interest on time. In addition, the homeloan interest rate is highly related with the loan credit of borrowers. A borrower with a higher credit can easily get homeloan with a lower rate. The task of the project is to predict whehter a borrower can replay homeloan and interest on time based on a number of features of the borrower. This is a typical classification problem and this notebook demonstrate how to use Lending Club dataset to decide whether a homeloan case should be approved based a borrower' features.

**Dataset Detail Information* The dataset contains 9,578 recrds in total. Every record has 13 features and 1 label which are described as below.

- credit.policy: This is the label. Its value is 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise.
- purpose: The purpose of the loan (takes values "creditcard", "debtconsolidation", "educational", "majorpurchase", "smallbusiness", “home_improvement” and "all_other").
- int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates.
- installment: The monthly installments owed by the borrower if the loan is funded.
- log.annual.inc: The natural log of the self-reported annual income of the borrower.
- dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income).
- fico: The FICO credit score of the borrower.Common FICO scores range from 300 to 850,with higher scores indicating better credit.
- days.with.cr.line: The number of days the borrower has had a credit line.
- revol.bal: The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle).
- revol.util: The borrower's revolving line utilization rate (the amount of the credit

line used relative to total credit available).

- inq.last.6mths: The borrower's number of inquiries by creditors in the last 6 months.
- delinq.2yrs: The number of times the borrower had been 30+ days past due on a payment in the past 2 years.
- pub.rec: The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments
- not.fully.paid： Whether the borrower will be fully paid or not.

Learning taks:

1. Data should be preprocessed and cleaned.
2. Feature selection should be conducted to remove irrelevant features.
3. Train a logistic regression model to predict "credit.policy" by using the other 13 features.
4. The logistic regression model should be evaluated with cross validation by using 5-10 folds.

Import Packages

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Load Dataset and Clean Dataset

The purpose of this part is to guarantee the data used in the notebook is reliable. 。

In [2]:
```
# Import Dataset
Data = pd.read_csv('./data/loan_data.csv')
# Display first 5 rows of dataset
Data.head()

# NOTE: Fist entry has missing data in 'not.fully.paid'
```

Out[2]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	56
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	27
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	26
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	40

After loading the dataset into the notebook, we can further check basic information of the dataset such as data type.

In [3]:

```python
# Check basic info of the dataset
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      9578 non-null   int64  
 1   purpose            9578 non-null   object 
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64  
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64  
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64  
 11  delinq.2yrs        9578 non-null   int64  
 12  pub.rec            9578 non-null   int64  
 13  not.fully.paid     9577 non-null   float64
dtypes: float64(7), int64(6), object(1)
memory usage: 1.0+ MB
```

According to the displayed dataset information, we can conclude that

1. The dataset is complete without missing any record.
2. There are 13 features and 1 label. There are three possible datatypes, which are loat64,int64 and object There are seven data types which are：credit_card, debt_consolidation, educational, major_purchase, small_business, home_improvement 和all_other。 Note that the type of purpose is object, which cannot be analyzed directly. This feature will be converted by OneHotEncoder or OrdinalEncoder.

Now, we can proveed to check basic statistical information of these features such as mean values, standard deviation, maximum and minimum values, etc.

In [4]:
```
# Check basic statistical information
Data.describe()
```

Out[4]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fic
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.00000
mean	0.804970	0.122570	319.089413	10.932117	12.606679	710.84631
std	0.396245	0.027163	207.071301	0.614813	6.883970	37.97053
min	0.000000	-0.146100	15.670000	7.547502	0.000000	612.00000
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.00000
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.00000
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.00000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.00000

Task 1: Based on Data info, please clean the dataset by removing abnormal data points or filling in missing values.

In [5]:
```
# Check for missing values
print(Data.isna().sum())

# NOTE: 1 missing value in 'not.fully.paid'
```

```
credit.policy        0
purpose              0
int.rate             0
installment          0
log.annual.inc       0
dti                  0
fico                 0
days.with.cr.line    0
revol.bal            0
revol.util           0
inq.last.6mths       0
delinq.2yrs          0
pub.rec              0
not.fully.paid       1
dtype: int64
```

In [6]:
```
# Fill missing data entry in 'not.fully.paid' with mean
Data['not.fully.paid'] = Data['not.fully.paid'].fillna(Data['not.fully.pa
Data.head()
```

Out[6]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	471
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4

In [7]:

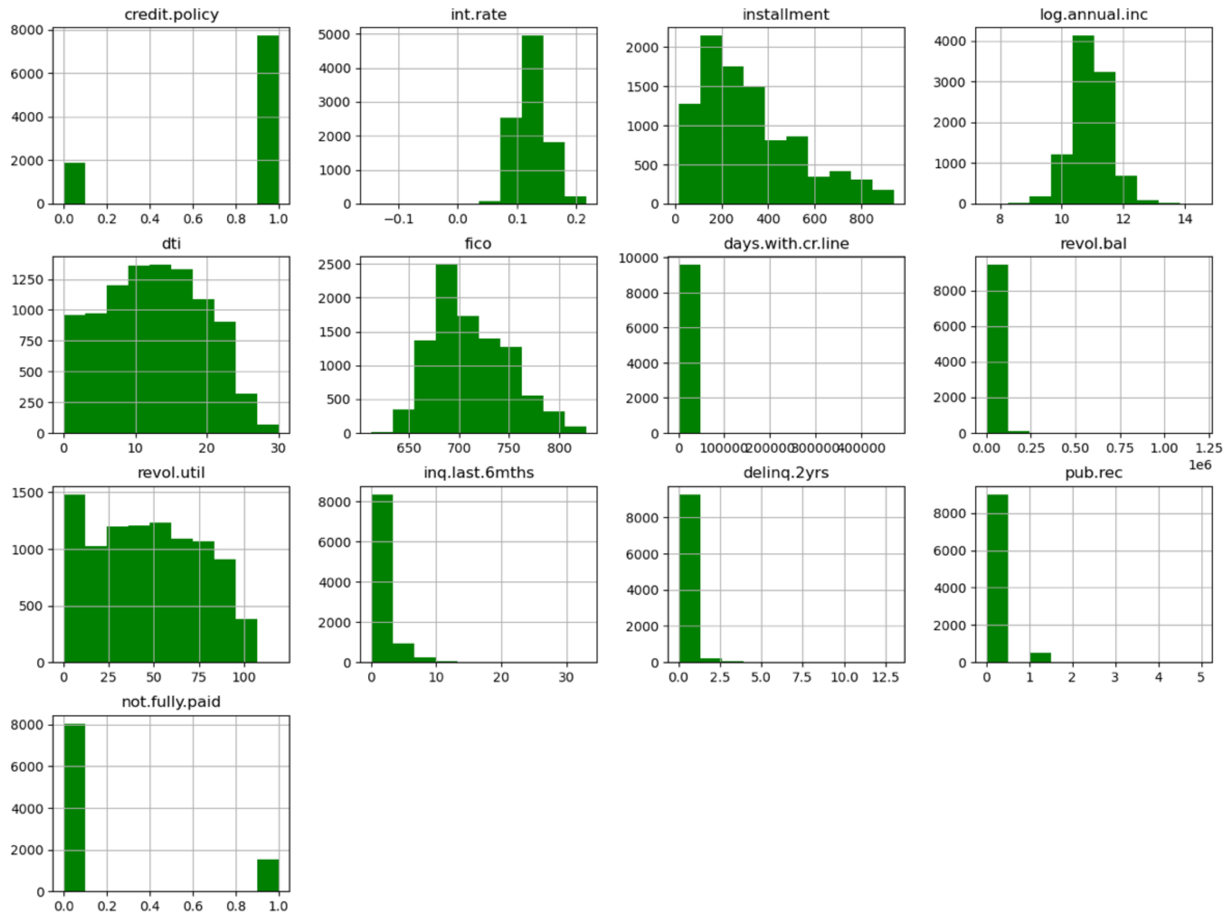
```
# Check for abnormal data points
Data.describe().transpose()
```

Out[7]:

	count	mean	std	min	25%	
credit.policy	9578.0	0.804970	0.396245	0.000000	1.000000	1.00
int.rate	9578.0	0.122570	0.027163	-0.146100	0.103900	0.12
installment	9578.0	319.089413	207.071301	15.670000	163.770000	268.95
log.annual.inc	9578.0	10.932117	0.614813	7.547502	10.558414	10.92
dti	9578.0	12.606679	6.883970	0.000000	7.212500	12.66
fico	9578.0	710.846314	37.970537	612.000000	682.000000	707.00
days.with.cr.line	9578.0	4609.450638	5380.501367	178.958333	2820.000000	4139.95
revol.bal	9578.0	16913.963876	33756.189557	0.000000	3187.000000	8596.00
revol.util	9578.0	46.799236	29.014417	0.000000	22.600000	46.30
inq.last.6mths	9578.0	1.577469	2.200245	0.000000	0.000000	1.00
delinq.2yrs	9578.0	0.163708	0.546215	0.000000	0.000000	0.00
pub.rec	9578.0	0.062122	0.262126	0.000000	0.000000	0.00
not.fully.paid	9578.0	0.160071	0.366672	0.000000	0.000000	0.00

In [8]:

```
# Display histogram of dataset
Data.hist(bins=10 ,figsize=(16,12), color = 'Green')
plt.show()
```



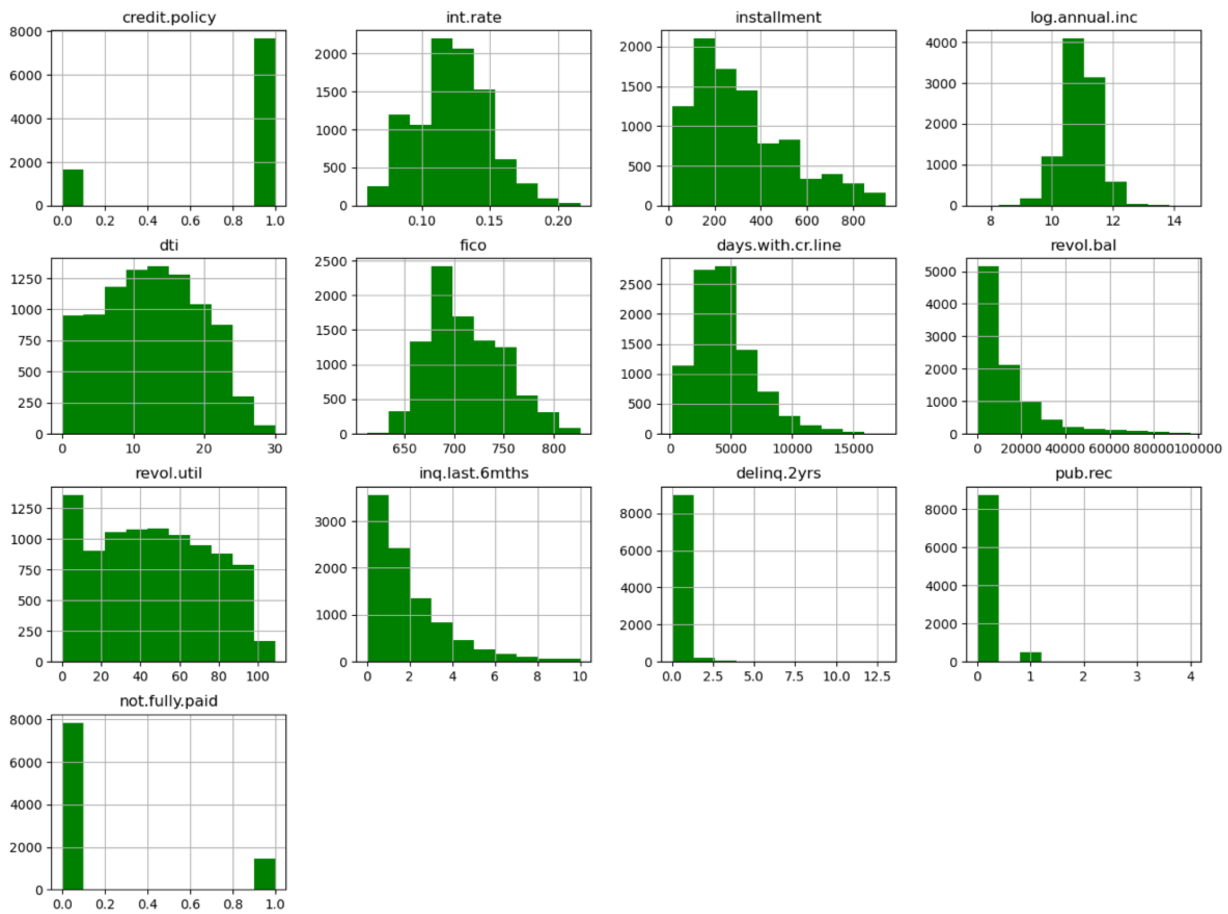
In [9]:
```
# Remove abnormal dat points
Data = Data[(Data['int.rate'] >= 0) & (Data['int.rate'] <= 1)]
Data = Data[(Data['days.with.cr.line'] >= 100) & (Data['days.with.cr.line
Data = Data[(Data['revol.bal'] >= 0) & (Data['revol.bal'] <= 100000)]
Data = Data[(Data['inq.last.6mths'] >= 0) & (Data['inq.last.6mths'] <= 10

# Display the cleaned dataset
Data.describe().transpose()
```

Out[9]:

	count	mean	std	min	25%	
credit.policy	9301.0	0.823890	0.380934	0.000000	1.000000	1.00
int.rate	9301.0	0.122428	0.026790	0.060000	0.102800	0.12
installment	9301.0	315.604349	204.460136	15.670000	163.480000	267.11
log.annual.inc	9301.0	10.909586	0.597767	7.547502	10.545341	10.91
dti	9301.0	12.515017	6.873931	0.000000	7.120000	12.57
fico	9301.0	711.024836	37.892683	612.000000	682.000000	707.00
days.with.cr.line	9301.0	4516.014756	2470.611283	178.958333	2789.958333	4096.00
revol.bal	9301.0	13256.676809	15392.386083	0.000000	3090.000000	8332.00
revol.util	9301.0	46.549483	28.941791	0.000000	22.300000	46.00
inq.last.6mths	9301.0	1.468122	1.804469	0.000000	0.000000	1.00
delinq.2yrs	9301.0	0.163961	0.543481	0.000000	0.000000	0.00
pub.rec	9301.0	0.061606	0.257719	0.000000	0.000000	0.00
not.fully.paid	9301.0	0.155914	0.362774	0.000000	0.000000	0.00

In [10]:

```
# Display histogram of cleaned dataset
Data.hist(bins=10 ,figsize=(16,12), color = 'Green')
plt.show()
```



Task 2: From data feature distributions, please discuss whether we should normalise these features.

In [11]:
```
# We should use normalisation as the ranges of values are very different
# The features have different scales, and some of them have a skewed dist
# such as 'int.rate', 'dti', and 'revol.bal'. Normalizing the data will e
# contributes equally to the analysis.
```

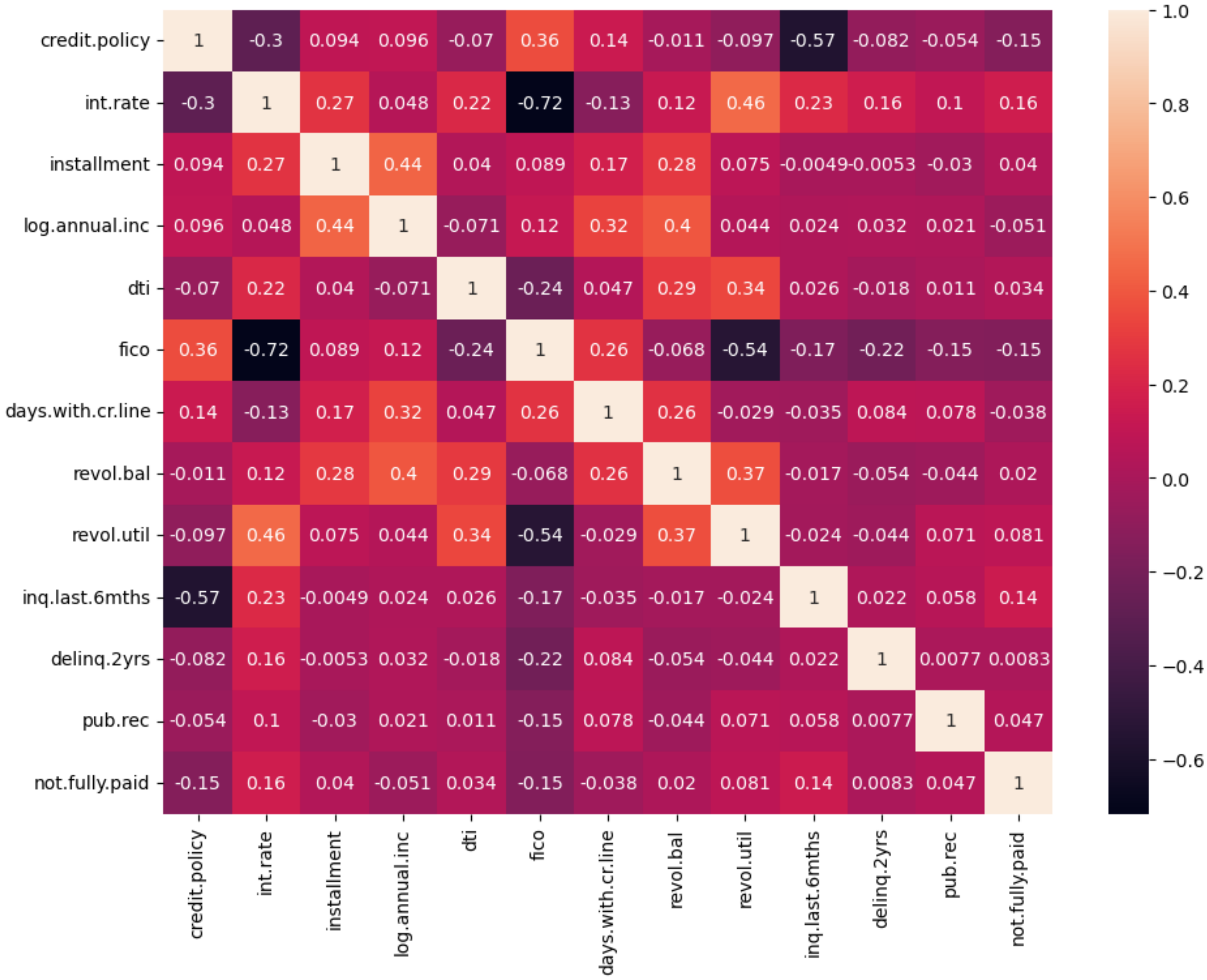
Data Analytics and Classification

Now, our target is to train a logistic regression model to predict 'credit.policy' with 13 features. This is a typical classification problem.

Task 3: It is unnecessary to use all 13 features as input of the logistic regression model. To select relevant features, we can plot the heatmap between two features to filter relevant features as our input.

In [12]:
```
# Create correlation heatmap of all features
plt.figure(figsize=(11,8))
sns.heatmap(Data.corr(), annot=True)

# NOTE: Use this to find what has a positive correlation with 'credit.pol
```

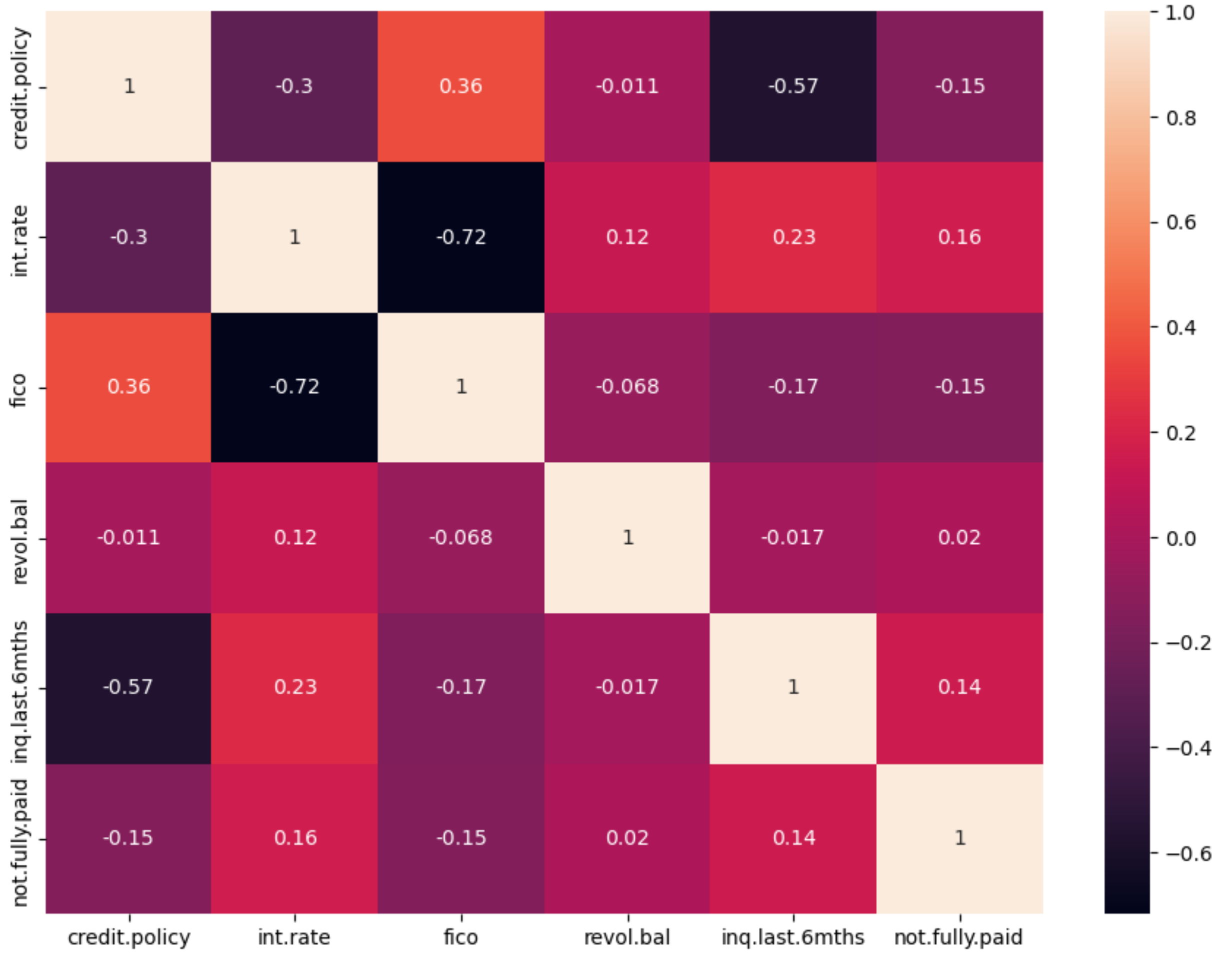
Out[12]:
```
<AxesSubplot:>
```



In [13]:

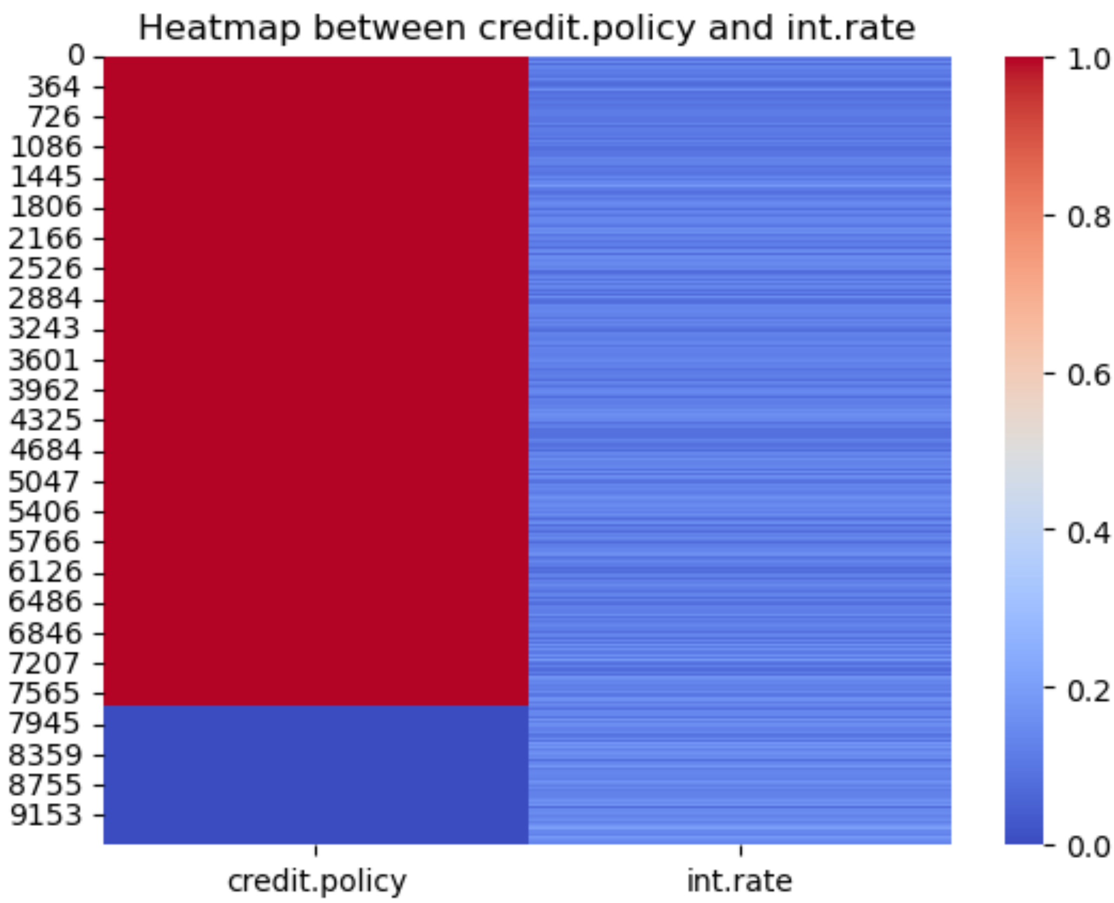
```
# drop uncorellated features
Data = Data.drop(['installment', 'log.annual.inc', 'dti', 'days.with.cr.l

# Create new correlation heatmap with remaining features
plt.figure(figsize=(11,8))
sns.heatmap(Data.corr(), annot=True)
```

Out[13]: <AxesSubplot:>



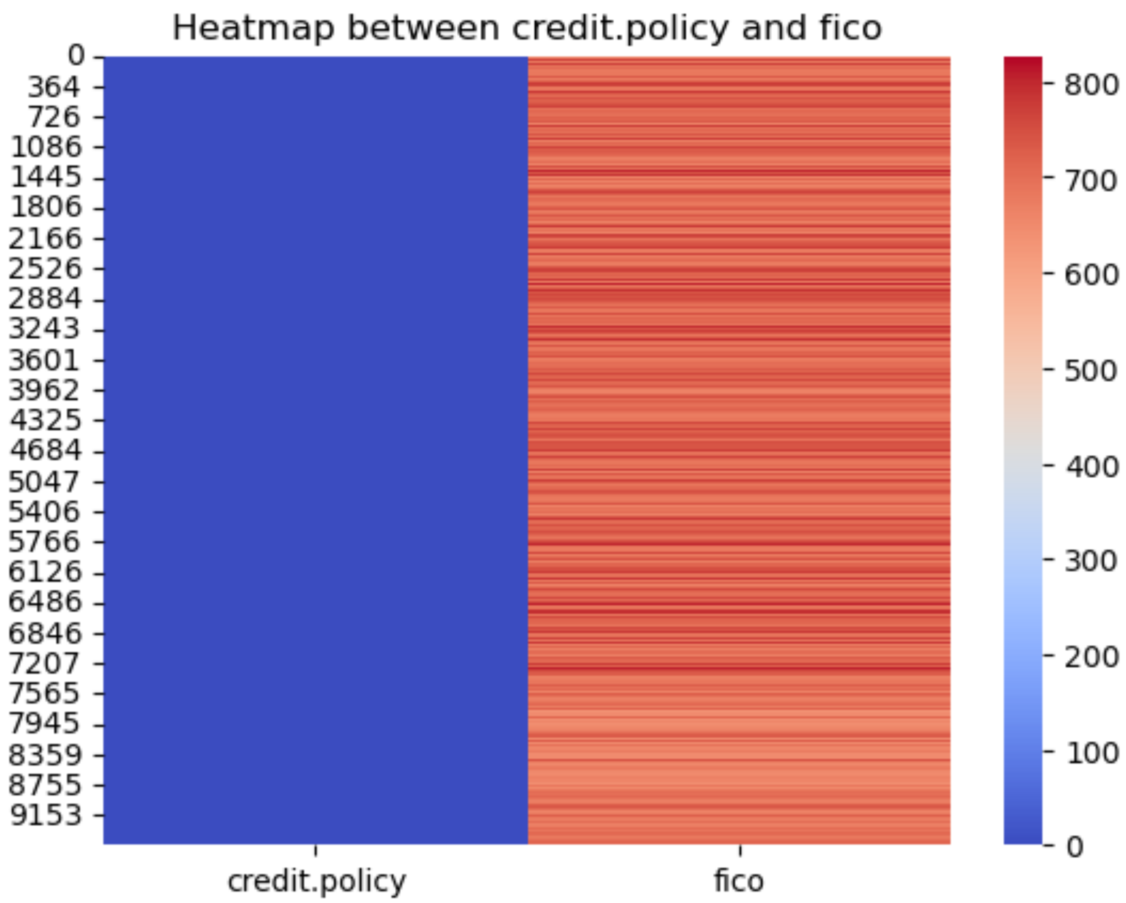
From the heatmaps, we can find different correlations between each feature and 'credit.policy'. We only reserve features that have positive correlations with 'credit.policy' by removing features with a low correlation with credit_policy.

In [14]:
```
# select two features to create heatmap between
feature1 = 'credit.policy'
feature2 = 'int.rate'

# create dataframe containing the two selected features
data_subset = Data[[feature1, feature2]]

# create heatmap
sns.heatmap(data_subset, cmap='coolwarm')

# set title
plt.title(f'Heatmap between {feature1} and {feature2}')

# show plot
plt.show()
```



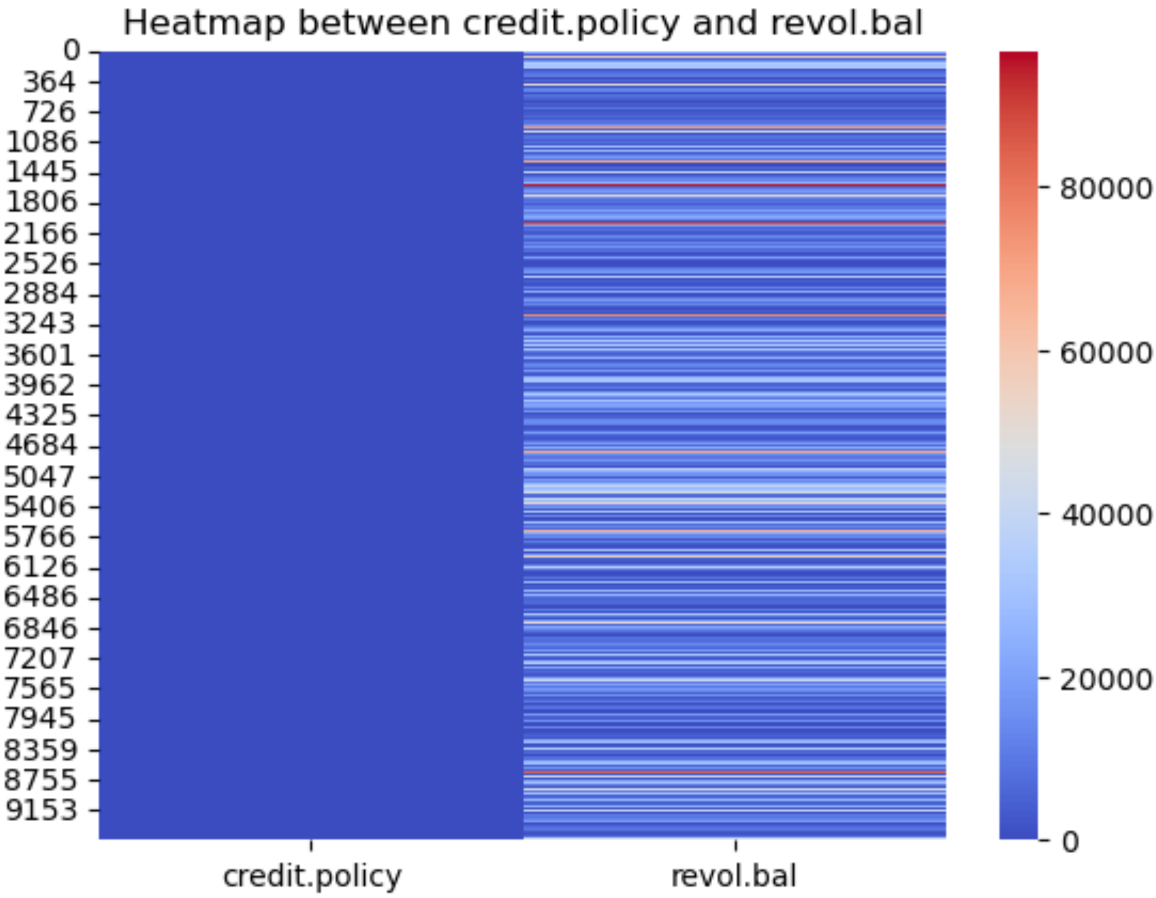
In [15]:

```
# select two features to create heatmap between
feature1 = 'credit.policy'
feature2 = 'fico'

# create dataframe containing the two selected features
data_subset = Data[[feature1, feature2]]

# create heatmap
sns.heatmap(data_subset, cmap='coolwarm')

# set title
plt.title(f'Heatmap between {feature1} and {feature2}')

# show plot
plt.show()
```



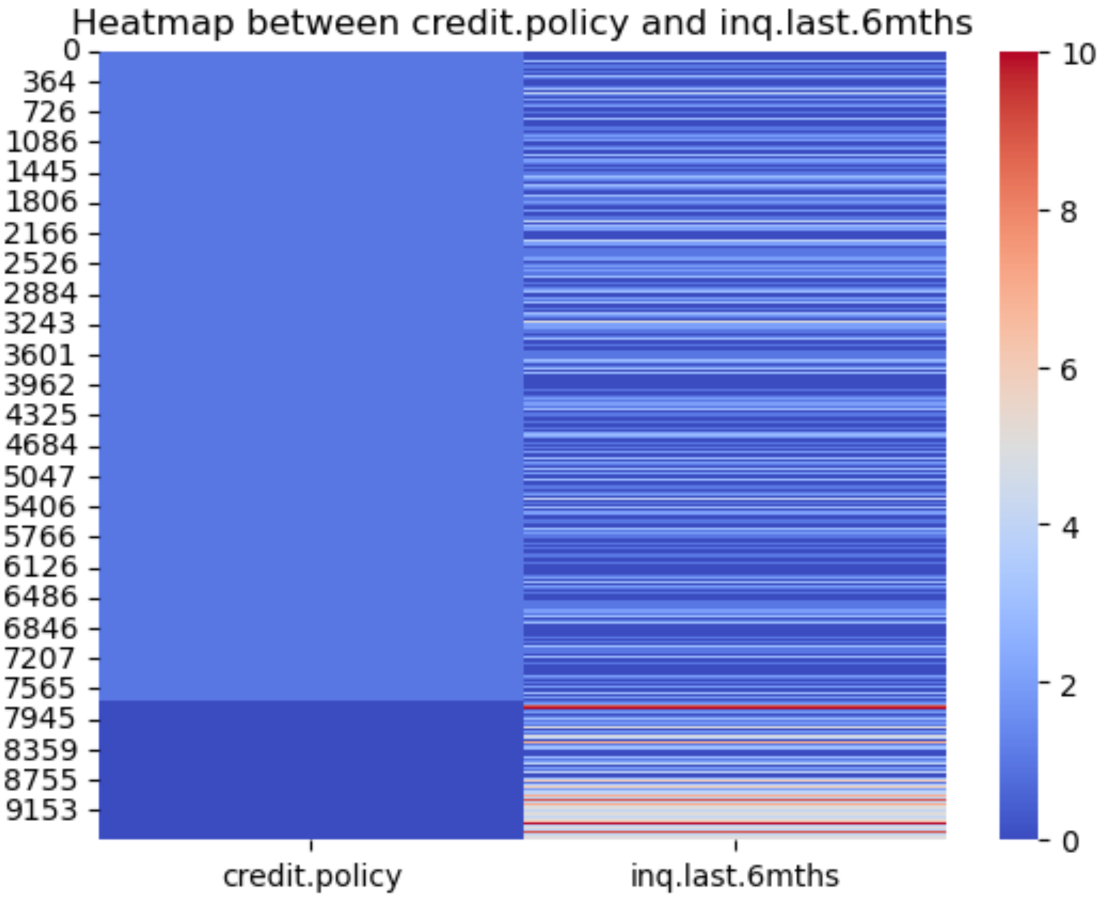
In [16]:

```
# select two features to create heatmap between
feature1 = 'credit.policy'
feature2 = 'revol.bal'

# create dataframe containing the two selected features
data_subset = Data[[feature1, feature2]]

# create heatmap
sns.heatmap(data_subset, cmap='coolwarm')

# set title
plt.title(f'Heatmap between {feature1} and {feature2}')

# show plot
plt.show()
```



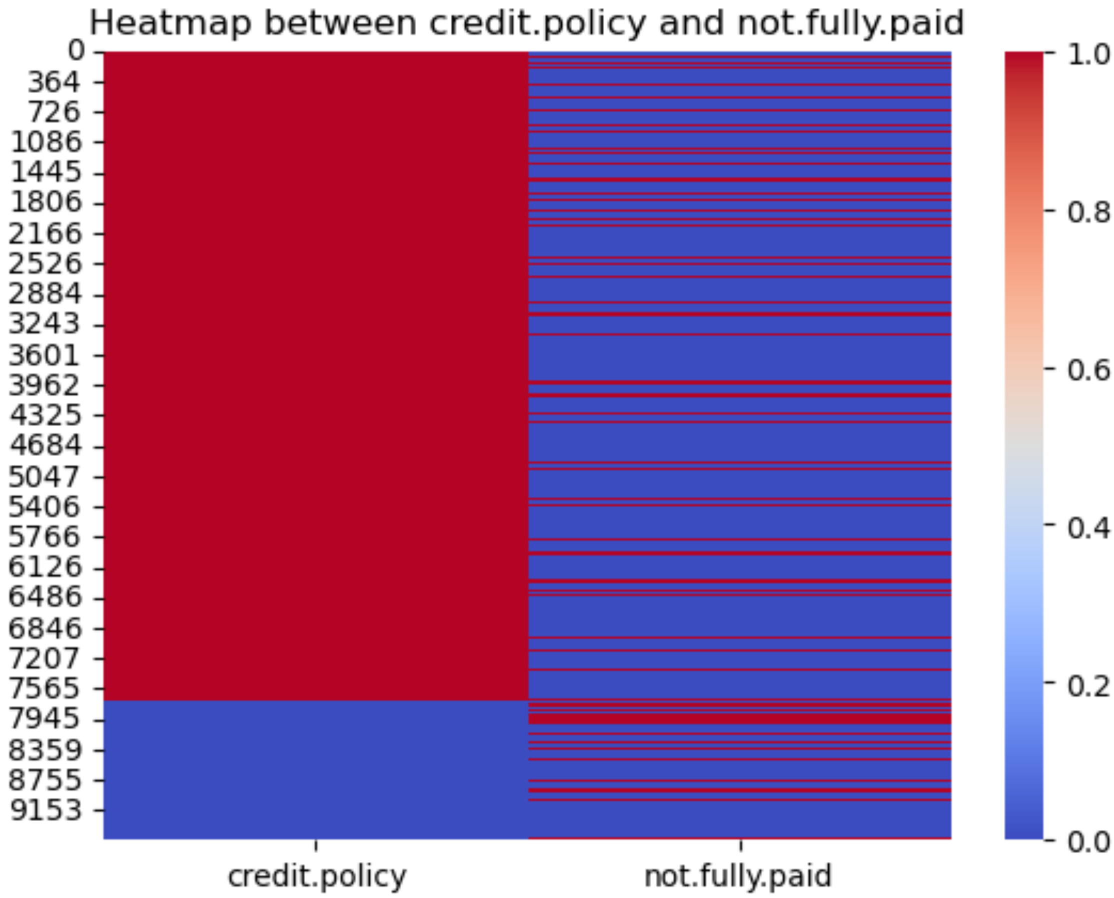
In [17]:

```
# select two features to create heatmap between
feature1 = 'credit.policy'
feature2 = 'inq.last.6mths'

# create dataframe containing the two selected features
data_subset = Data[[feature1, feature2]]

# create heatmap
sns.heatmap(data_subset, cmap='coolwarm')

# set title
plt.title(f'Heatmap between {feature1} and {feature2}')

# show plot
plt.show()
```



In [18]:

```
# select two features to create heatmap between
feature1 = 'credit.policy'
feature2 = 'not.fully.paid'

# create dataframe containing the two selected features
data_subset = Data[[feature1, feature2]]

# create heatmap
sns.heatmap(data_subset, cmap='coolwarm')

# set title
plt.title(f'Heatmap between {feature1} and {feature2}')

# show plot
plt.show()
```



Data Preprocess

Process of Object data type

The logistic regression model cannot well process the object data type. We convert this data type with OneHotEncoder such that this feature can be handled by the logistic regression model.

In [19]:
```
dummy_purpose = pd.get_dummies(Data['purpose'])
dummy_purpose.head()
New_Data = pd.concat((Data.iloc[:,0], dummy_purpose, Data.iloc[:,2:]), ax
New_Data.head()
```

Out[19]:

	credit.policy	all_other	credit_card	debt_consolidation	educational	home_improveme
0	1	0	0	1	0	
1	1	0	1	0	0	
3	1	0	0	1	0	
4	1	0	1	0	0	
5	1	0	1	0	0	

Dataset classification

We classify all data records into training set (80%), validation set (10%) and test set (10%) so that we can determine hyper-parameters with k-cross validation.

Randomly select 90% as the training + validation sets. The rest 10% will be used as the test set.

In [20]:
```
from sklearn.model_selection import train_test_split
print(New_Data['credit.policy'].value_counts())
```

```
1    7663
0    1638
Name: credit.policy, dtype: int64
```

We complete dataset classification as below.

In [21]:
```
x_ex1 = New_Data.copy().drop(columns=['credit.policy', 'int.rate','revol.
y_ex1 = New_Data.copy()['credit.policy']
x_ex1_array = x_ex1.values
y_ex1_array = y_ex1.values
x_ex1_train = x_ex1_array[0:int((len(y_ex1_array)+1)*0.9),:]
x_ex1_test = x_ex1_array[int((len(y_ex1_array)+1)*0.9):,:]
y_ex1_train = y_ex1_array[0:int((len(y_ex1_array)+1)*0.9)]
y_ex1_test = y_ex1_array[int((len(y_ex1_array)+1)*0.9):]
```

In [22]:
```
x_ex1_train.shape
```

Out[22]:
```
(8371, 8)
```

In [23]:
```
y_ex1_train
```

Out[23]:
```
array([1, 1, 1, ..., 0, 0, 0])
```

In [24]:
```
# Randomly select 90% as the training + validation sets. The rest 10% wil
x_ex1_train, x_ex1_test, y_ex1_train, y_ex1_test = train_test_split(x_ex1
```

In [25]:
```
# Confirm split
x_ex1_train
```

Out[25]:
```
array([[  0,   0,   1, ...,   0,   0, 742],
       [  0,   0,   1, ...,   0,   0, 652],
       [  1,   0,   0, ...,   0,   0, 682],
       ...,
       [  0,   0,   1, ...,   0,   0, 722],
       [  0,   0,   1, ...,   0,   0, 692],
       [  0,   0,   1, ...,   0,   0, 667]])
```

Data normalisation

Task 4: Recall that we have observed large value discrepancies between these features. It is necessary to normalise these features before we use them to train our models. Here, we emply standardization method to normalise our dataset as below.

In [26]:
```
from sklearn.preprocessing import StandardScaler

obje_ss = StandardScaler()
x_ex1_train = obje_ss.fit_transform(x_ex1_train)
x_ex1_test = obje_ss.fit_transform(x_ex1_test)

x_ex1_train
```

Out[26]:
```
array([[-0.5728436 , -0.3867394 ,  1.18861225, ..., -0.22136765,
        -0.25366089,  0.82202916],
       [-0.5728436 , -0.3867394 ,  1.18861225, ..., -0.22136765,
        -0.25366089, -1.54906353],
       [ 1.74567717, -0.3867394 , -0.84131726, ..., -0.22136765,
        -0.25366089, -0.7586993 ],
       ...,
       [-0.5728436 , -0.3867394 ,  1.18861225, ..., -0.22136765,
        -0.25366089,  0.29511967],
       [-0.5728436 , -0.3867394 ,  1.18861225, ..., -0.22136765,
        -0.25366089, -0.49524456],
       [-0.5728436 , -0.3867394 ,  1.18861225, ..., -0.22136765,
        -0.25366089, -1.15388142]])
```

Model Evaluation

In this stage, we are going to train a logistic regression model. Cross validation will be used to determine hyper-parameters and evaluate model performance.

Logistic Regression Model

Task 5: Train a Logistic Regression Model with training dataset.

In [27]:
```
from sklearn.linear_model import LogisticRegression
model_log = LogisticRegression()

model_log.fit(x_ex1_train, y_ex1_train)
```

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

In [28]:
```
# Display training dataset score
training_score = model_log.score(x_ex1_train, y_ex1_train)
training_score
```

Out[28]:
```
0.8524492234169654
```

In [29]:
```
from sklearn.metrics import accuracy_score

predictOnTest = model_log.predict(x_ex1_test)
# Evaluate against test data
accuracy_score(y_ex1_test, predictOnTest)
```

Out[29]:
```
0.8549946294307197
```

Task 6: To better understand our result, we visualize the performance evluation by comparing the model accuracy of the Logistic Regression model on the training dataset and each validation dataset.

In [30]:
```
from sklearn.model_selection import cross_validate

cross_validate(model_log, X=x_ex1_train, y=y_ex1_train, scoring='accuracy
```

Out[30]:
```
{'fit_time': array([0.00943899, 0.00697613, 0.00720882, 0.00626731, 0.005
90491,
        0.00606012, 0.00577998, 0.00820112, 0.00732923]),
 'score_time': array([0.00039697, 0.00030994, 0.00023007, 0.00019979, 0.0
002501 ,
        0.00022602, 0.00026011, 0.00022674, 0.00024414]),
 'test_score': array([0.8516129 , 0.85053763, 0.84623656, 0.84731183, 0.8
5376344,
        0.86344086, 0.85376344, 0.84946237, 0.84408602]),
 'train_score': array([0.85255376, 0.85094086, 0.85322581, 0.8530914 , 0.
85228495,
        0.85067204, 0.85120968, 0.85067204, 0.85107527])}
```

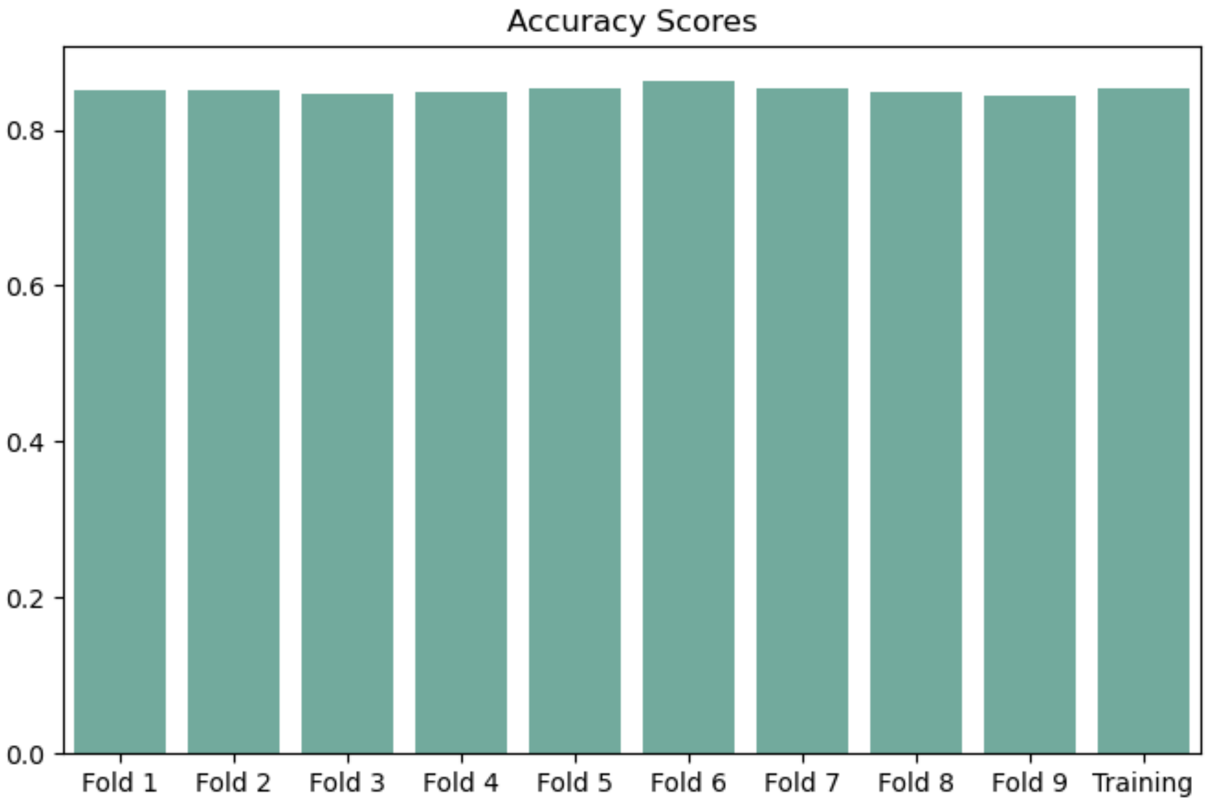
In [31]:
```
# using cross_val_score
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model_log, x_ex1_train, y_ex1_train, cv=9)

# Display
scores
```

Out[31]:
```
array([0.8516129 , 0.85053763, 0.84623656, 0.84731183, 0.85376344,
       0.86344086, 0.85376344, 0.84946237, 0.84408602])
```

In [32]:
```
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores.mean(
```

```
0.8511 accuracy with a standard deviation of 0.0053
```

In [33]:
```
# Add training score to cross validation array
scores = np.append(scores, training_score)

# Set x of plot to equal the range of the array (between 0 and 1)
x = np.arange(len(scores))

# Set the figure size
plt.figure(figsize=(8, 5))

# set title
plt.title('Accuracy Scores')

# plot a bar chart
sns.barplot(x, scores, color='#69b3a2').set(xticklabels=['Fold 1', 'Fold
plt.show()
```



From the visualised results, we can observe that the model prediction performance is very good. The accuracy on validation dataset is only slightly lower than the accuracy on the training dataset. This result is convincing since we have conducted 9 fold cross validation. Our cross validation indicates that we have obtained an accurate model.

Note that the test dataset is not used for evaluation. Since there is no hyperparameter in the Logistic Regression model, the cross-validation has reflected the performance of the model on unknown datasets.