Lending Club

July 8, 2024

[1]:
```
#Packages for data reading and manipulation
import pandas as pd
import numpy as np
#for data visualization
import plotly.express as px
import plotly.subplots as sp
import plotly.graph_objs as go
#for encoding
from sklearn.preprocessing import OneHotEncoder
#for data scaling
from sklearn.preprocessing import RobustScaler
#for data splitting
from sklearn.model_selection import train_test_split
#for modelling
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
#for evaluation
from sklearn.metrics import accuracy_score

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

```
2024-06-03 09:02:36.888380: I tensorflow/core/util/port.cc:110] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-06-03 09:02:36.926597: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

```
VOC-NOTICE: GPU memory for this assignment is capped at 1024MiB
```

```
2024-06-03 09:02:38.712422: E
tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:268] failed call to
cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
```

[2]:
```
#import dataset
df=pd.read_csv("loan_data.csv")
df.head()
```

[2]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

[3]:
```
#data summary statistics
df.describe()
```

[3]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.799236	1.577469	0.163708	0.062122	0.160054
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.014417	2.200245	0.546215	0.262126	0.366676
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.600000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.300000	1.000000	0.000000	0.000000	0.000000
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.900000	2.000000	0.000000	0.000000	0.000000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000	33.000000	13.000000	5.000000	1.000000

```
       inq.last.6mths  delinq.2yrs      pub.rec  not.fully.paid
count     9578.000000  9578.000000  9578.000000     9578.000000
mean         1.577469     0.163708     0.062122        0.160054
std          2.200245     0.546215     0.262126        0.366676
min          0.000000     0.000000     0.000000        0.000000
25%          0.000000     0.000000     0.000000        0.000000
50%          1.000000     0.000000     0.000000        0.000000
75%          2.000000     0.000000     0.000000        0.000000
max         33.000000    13.000000     5.000000        1.000000
```

[4]:
```
#null values
df.isnull().sum()
```

[4]:
```
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       0
dtype: int64
```

[5]:
```
#data dimensions
df.shape
```

[5]:
```
(9578, 14)
```

[6]:
```
df.columns
```

[6]:
```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

[7]:
```
#unique values
df[['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid']].nunique()
```

[7]:
```
credit.policy          2
purpose                7
int.rate             249
installment         4788
log.annual.inc      1987
dti                 2529
fico                  44
days.with.cr.line   2687
revol.bal           7869
revol.util          1035
inq.last.6mths        28
delinq.2yrs           11
pub.rec                6
not.fully.paid         2
dtype: int64
```

[8]:
```
# variables data types
df.dtypes
```

[8]:
```
credit.policy          int64
purpose               object
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                   int64
days.with.cr.line    float64
revol.bal              int64
revol.util           float64
inq.last.6mths         int64
delinq.2yrs            int64
pub.rec                int64
not.fully.paid         int64
dtype: object
```

[9]:
```
#EXPLOTATORY ANALYSIS
```

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

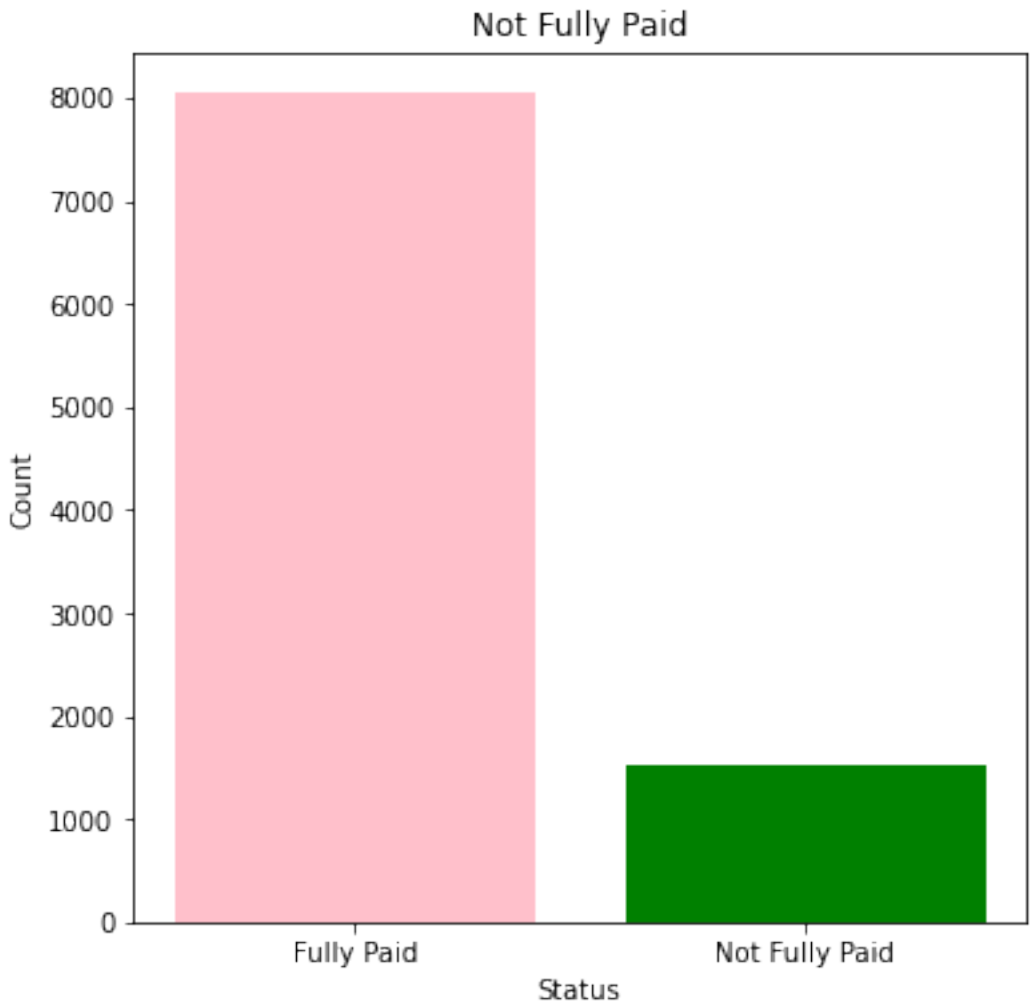
[10]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	\
0	1	debt_consolidation	0.1189	829.10	11.350407	
1	1	credit_card	0.1071	228.22	11.082143	
2	1	debt_consolidation	0.1357	366.86	10.373491	
3	1	debt_consolidation	0.1008	162.34	11.350407	
4	1	credit_card	0.1426	102.92	11.299732	

	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	\

```
0  19.48   737        5639.958333      28854        52.1                0
1  14.29   707        2760.000000      33623        76.7                0
2  11.63   682        4710.000000       3511        25.6                1
3   8.10   712        2699.958333      33667        73.2                1
4  14.97   667        4066.000000       4740        39.5                0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

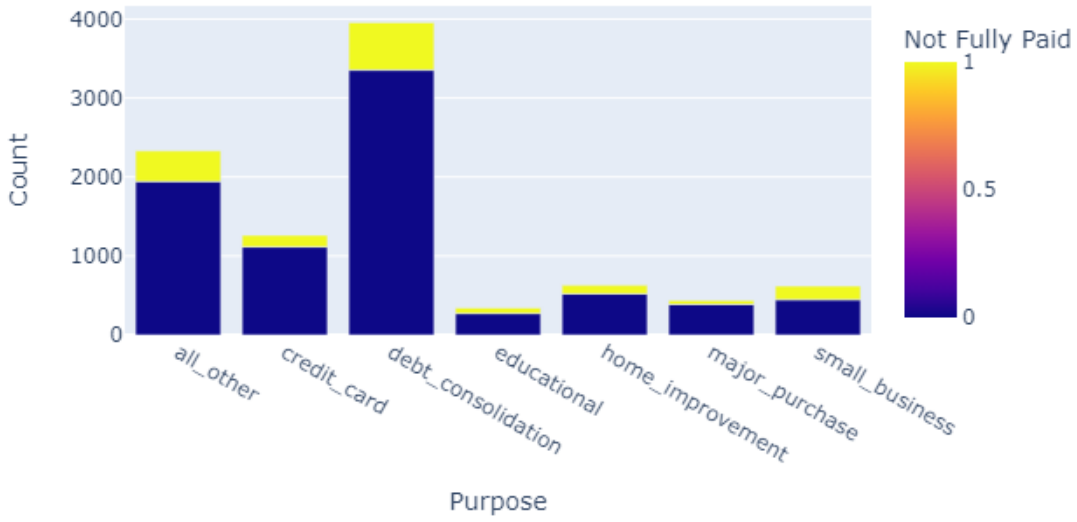
[11]:
```
# target value counts
import matplotlib.pyplot as plt
target_value_counts = df['not.fully.paid'].value_counts()
plt.figure(figsize=(6, 6))
plt.bar(target_value_counts.index, target_value_counts.values, color=['pink',␣
 ↪'green'])
plt.title('Not Fully Paid')
plt.xlabel('Status')
plt.ylabel('Count')
plt.xticks(target_value_counts.index, ['Fully Paid', 'Not Fully Paid'])
plt.show()
```



[12]:
```
#purpose vs not fully paid
grouped = df.groupby(['purpose', 'not.fully.paid']).size().
 ↪reset_index(name='count')
print(grouped)
fig = px.bar(grouped, x='purpose', y='count', color='not.fully.paid',
             barmode='group', labels={'count': 'Count', 'purpose': 'Purpose',␣
 ↪'not.fully.paid': 'Not Fully Paid'})
fig.show()
```

```
              purpose  not.fully.paid  count
0           all_other               0   1944
1           all_other               1    387
2         credit_card               0   1116
3         credit_card               1    146
4   debt_consolidation               0   3354
5   debt_consolidation               1    603
```

```
6          educational                0    274
7          educational                1     69
8     home_improvement                0    522
9     home_improvement                1    107
10      major_purchase                0    388
11      major_purchase                1     49
12      small_business                0    447
13      small_business                1    172
```

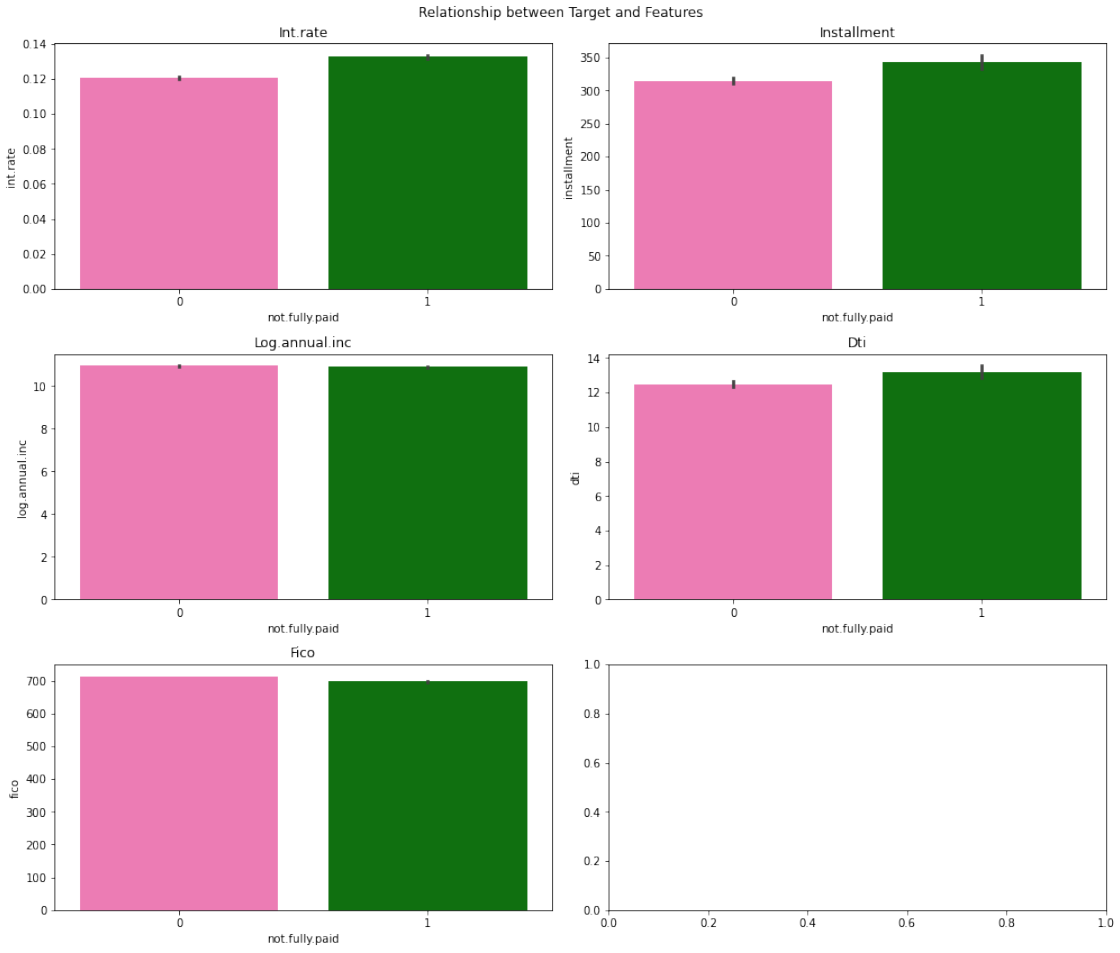


[13]:
```
df.columns
```

[13]:
```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

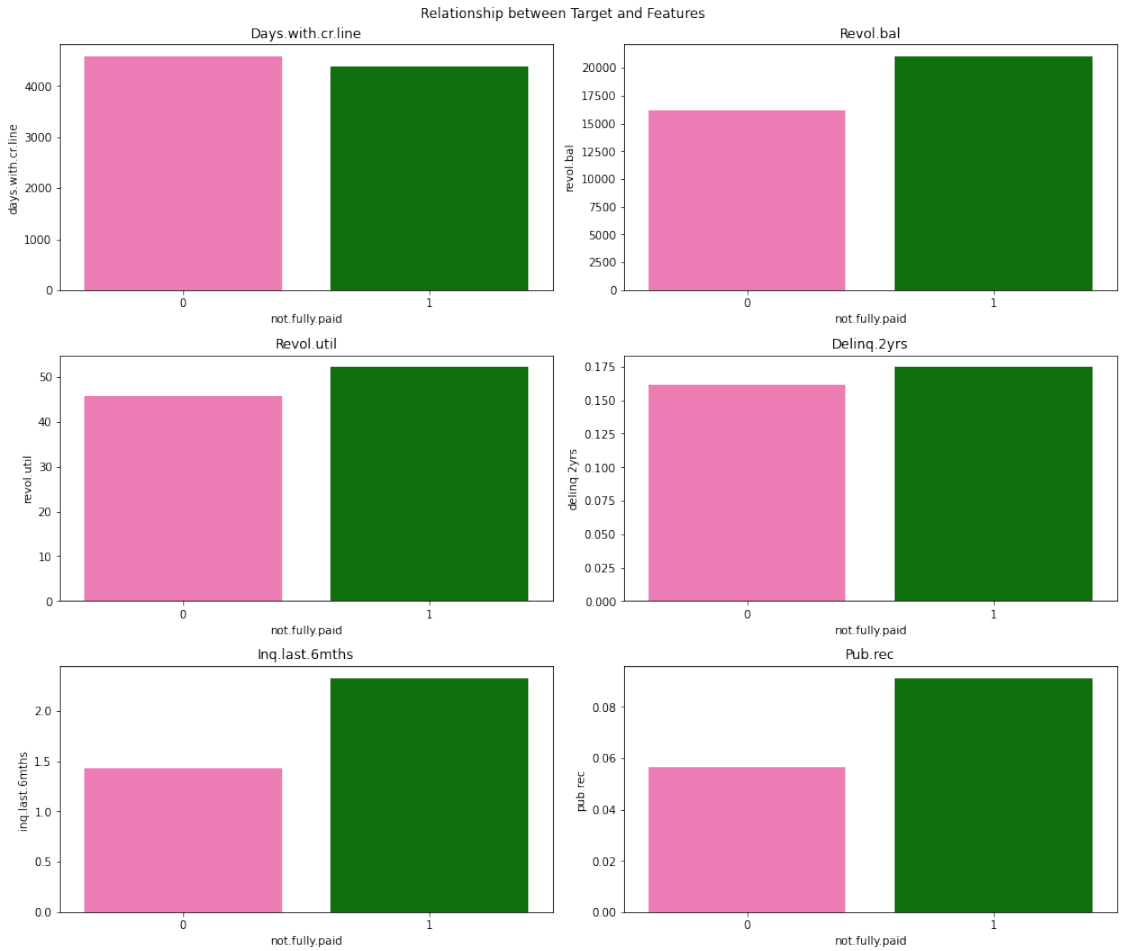
[14]:
```
#Relationship between target and features
import seaborn as sns
import matplotlib.pyplot as plt

# List of features
features = ['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico']
# Setting up subplots
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14, 12))
fig.suptitle('Relationship between Target and Features')

# Define custom colors
```

```
colors = ['#FF69B4', '#008000']

# Plotting each feature
for i, feature in enumerate(features):
    row = i // 2
    col = i % 2
    sns.barplot(x='not.fully.paid', y=feature, data=df, ax=axes[row, col],␣
 ↪palette=colors)
    axes[row, col].set_title(feature.capitalize())

# Adjust layout
plt.tight_layout()
plt.show()
```



[15]:
```
# Define features and pivot tables
features = ['days.with.cr.line', 'revol.bal', 'revol.util', 'delinq.2yrs', 'inq.
 ↪last.6mths', 'pub.rec']
```

```
grouped = [df.pivot_table(values=feat, index='not.fully.paid', aggfunc='mean')␣
 ↪for feat in features]

# Setting up subplots
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14, 12))
fig.suptitle('Relationship between Target and Features')

# Define custom colors
colors = ['#FF69B4', '#008000']

# Plotting each feature
for i, (feat, group) in enumerate(zip(features, grouped)):
    row = i // 2
    col = i % 2
    sns.barplot(x=group.index, y=feat, data=group.reset_index(), ax=axes[row,␣
 ↪col], palette=colors)
    axes[row, col].set_title(feat.capitalize())

# Adjust layout
plt.tight_layout()
plt.show()
```



[16]:
```
df.columns
```

[16]:
```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

[17]:
```
#DATA ENCODING
```

[18]:
```
# Perform one-hot encoding using pandas get_dummies
cat_encoded_df = pd.get_dummies(df['purpose'], drop_first=True)

cat_encoded_df.head()  # Displaying the encoded DataFrame
```

[18]:

	credit_card	debt_consolidation	educational	home_improvement	\
0	0	1	0	0	
1	1	0	0	0	

```
2                0                    1              0                    0
3                0                    1              0                    0
4                1                    0              0                    0

   major_purchase  small_business
0               0               0
1               0               0
2               0               0
3               0               0
4               0               0
```

[31]:
```
# Add a suffix to the columns of the encoded DataFrame
cat_encoded_df = cat_encoded_df.add_suffix('_encoded')

# Merge the encoded DataFrame with the original DataFrame
df = pd.concat([df, cat_encoded_df], axis=1)

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

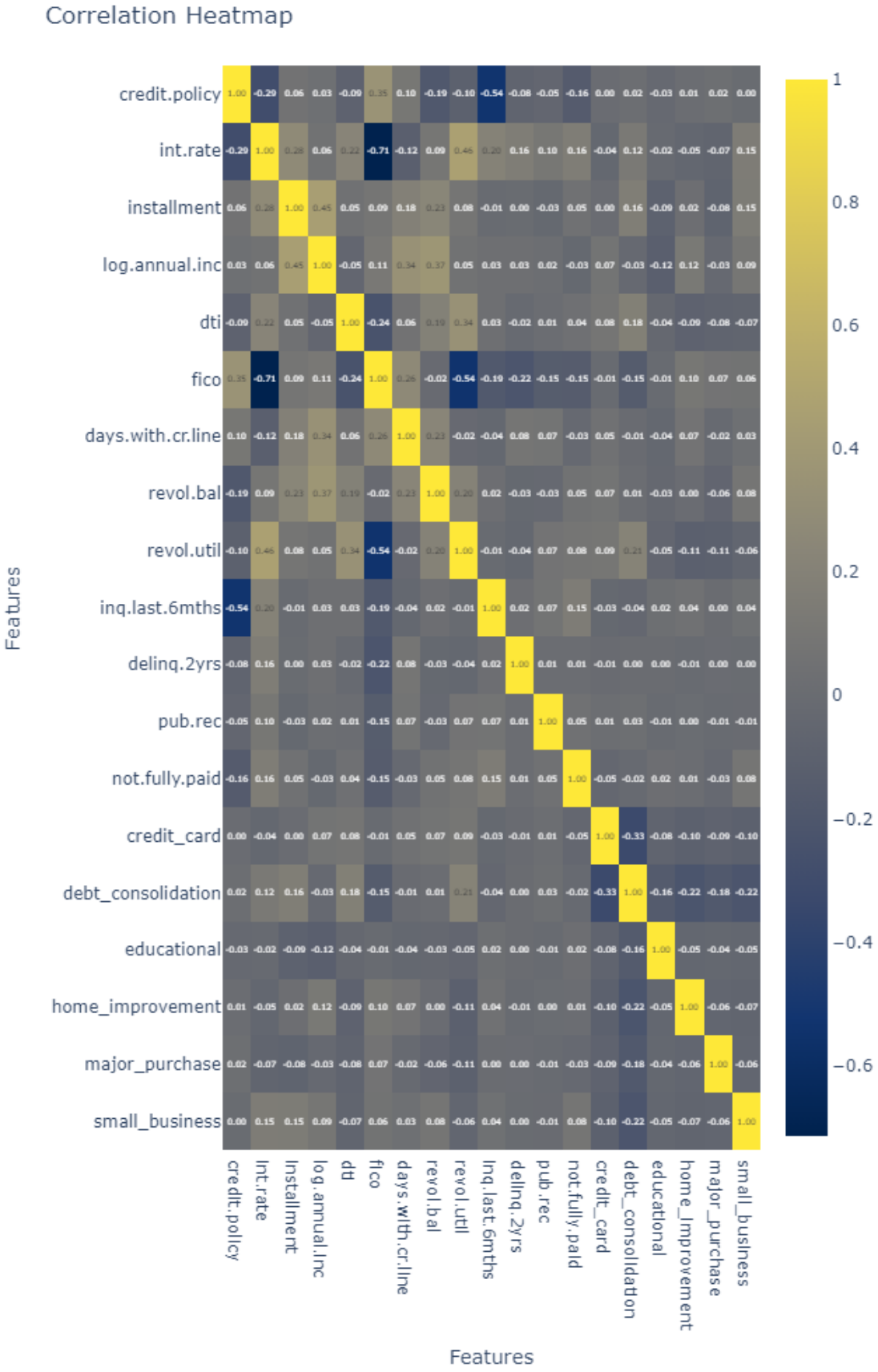
```
   credit.policy  int.rate  installment  log.annual.inc    dti  fico  \
0              1    0.1189       829.10       11.350407  19.48   737
1              1    0.1071       228.22       11.082143  14.29   707
2              1    0.1357       366.86       10.373491  11.63   682
3              1 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

   days.with.cr.line  revol.bal  revol.util  inq.last.6mths  …  educational  \
0        5639.958333      28854        52.1               0  …            0
1        2760.000000      33623        76.7               0  …            0
2        4710.000000       3511        25.6               1  …            0
3        2699.958333      33667        73.2               1  …            0
4        4066.000000       4740        39.5               0  …            0

   home_improvement  major_purchase  small_business  credit_card_encoded  \
0                 0               0               0                    0
1                 0               0               0                    1
2                 0               0               0                    0
3                 0               0               0                    0
4                 0               0               0                    1

   debt_consolidation_encoded  educational_encoded  home_improvement_encoded  \
0                           1                    0                         0
1                           0                    0                         0
2                           1                    0                         0
3                           1                    0                         0
4                           0                    0                         0
```

```
   major_purchase_encoded  small_business_encoded
0                       0                       0
1                       0                       0
2                       0                       0
3                       0                       0
4                       0                       0

[5 rows x 43 columns]
```

[32]:
```
df.columns
```

[32]:
```
Index(['credit.policy', 'int.rate', 'installment', 'log.annual.inc', 'dti',
       'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid',
       'credit_card', 'debt_consolidation', 'educational', 'home_improvement',
       'major_purchase', 'small_business', 'credit_card', 'debt_consolidation',
       'educational', 'home_improvement', 'major_purchase', 'small_business',
       'credit_card', 'debt_consolidation', 'educational', 'home_improvement',
       'major_purchase', 'small_business', 'credit_card', 'debt_consolidation',
       'educational', 'home_improvement', 'major_purchase', 'small_business',
       'credit_card_encoded', 'debt_consolidation_encoded',
       'educational_encoded', 'home_improvement_encoded',
       'major_purchase_encoded', 'small_business_encoded'],
      dtype='object')
```

[37]:
```
#Correlation Heatmap.
import plotly.express as px

corr = df[['credit.policy', 'int.rate', 'installment', 'log.annual.inc',
           'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
           'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid',
           'credit_card', 'debt_consolidation',
           'educational', 'home_improvement',
           'major_purchase', 'small_business']].corr()

fig = px.imshow(corr, text_auto=".2f", color_continuous_scale='Cividis',␣
 ↪aspect="auto")

fig.update_layout(title='Correlation Heatmap',
                  xaxis_title='Features',
                  yaxis_title='Features')
fig.update_layout(height=1000)

fig.show()
```


Correlation Heatmap
credit.policy
int.rate
installment
log.annual.inc
dti
fico
days.with.cr.line
revol.bal
revol.util
inq.last.6mths
delinq.2yrs
pub.rec
not.fully.paid
credit_card
debt_consolidation
educational
home_improvement
major_purchase
small_business
Features
Features
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6

[ ]:
```
#DATA SPLITTING
```

[38]:
```
df.columns
```

[38]:
```
Index(['credit.policy', 'int.rate', 'installment', 'log.annual.inc', 'dti',
       'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid',
       'credit_card', 'debt_consolidation', 'educational', 'home_improvement',
       'major_purchase', 'small_business', 'credit_card', 'debt_consolidation',
       'educational', 'home_improvement', 'major_purchase', 'small_business',
       'credit_card', 'debt_consolidation', 'educational', 'home_improvement',
       'major_purchase', 'small_business', 'credit_card', 'debt_consolidation',
       'educational', 'home_improvement', 'major_purchase', 'small_business',
       'credit_card_encoded', 'debt_consolidation_encoded',
       'educational_encoded', 'home_improvement_encoded',
       'major_purchase_encoded', 'small_business_encoded'],
      dtype='object')
```

[42]:
```
 #defining features and target variable
x = df[['credit.policy', 'int.rate', 'installment', 'log.annual.inc', 'dti',
        'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec',
       'credit_card', 'debt_consolidation',
       'educational', 'home_improvement',
       'major_purchase', 'small_business']]
y = df['not.fully.paid']
print(len(x))
print(len(y))
```

```
9578
9578
```

[43]:
```
#splitting the data into training and testing.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25)
print(x_train.shape)
print(x_test.shape)
```

```
(7183, 36)
(2395, 36)
```

[45]:
```
#scaling the data sets
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

[ ]:
```
#CUSTOM NEURAL NETWORK
```

[46]:
```
#custom model with hidden layers
import tensorflow as tf
class CustomNetwork(tf.keras.Model):
    def __init__(self):
        super(CustomNetwork, self).__init__()
        self.dense1 = tf.keras.layers.Dense(100, activation='sigmoid')
        self.dropout1 = tf.keras.layers.Dropout(0.2)
        self.dense2 = tf.keras.layers.Dense(50, activation='sigmoid')
        self.dropout2 = tf.keras.layers.Dropout(0.2)
        self.dense3 = tf.keras.layers.Dense(20, activation='sigmoid')
        self.dropout3 = tf.keras.layers.Dropout(0.2)
        self.dense4 = tf.keras.layers.Dense(1, activation='sigmoid')

    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dropout1(x)
        x = self.dense2(x)
        x = self.dropout2(x)
        x = self.dense3(x)
        x = self.dropout3(x)
        x = self.dense4(x)
        return x


model = CustomNetwork()
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
              loss=tf.keras.losses.BinaryCrossentropy(), #binary cross entropy␣
 ↪because the classification is binary 1,0
              metrics=['accuracy'])

model.fit(x_train_scaled, y_train, epochs=10)
```

```
Epoch 1/10
225/225 [==============================] - 1s 1ms/step - loss: 0.4478 -
accuracy: 0.8338
Epoch 2/10
225/225 [==============================] - 0s 1ms/step - loss: 0.4302 -
accuracy: 0.8359
Epoch 3/10
225/225 [==============================] - 0s 1ms/step - loss: 0.4263 -
accuracy: 0.8359
Epoch 4/10
225/225 [==============================] - 0s 1ms/step - loss: 0.4251 -
accuracy: 0.8359
Epoch 5/10
```

```
225/225 [==============================] - 0s 988us/step - loss: 0.4205 -
accuracy: 0.8359
Epoch 6/10
225/225 [==============================] - 0s 975us/step - loss: 0.4214 -
accuracy: 0.8356
Epoch 7/10
225/225 [==============================] - 0s 973us/step - loss: 0.4184 -
accuracy: 0.8359
Epoch 8/10
225/225 [==============================] - 0s 982us/step - loss: 0.4184 -
accuracy: 0.8354
Epoch 9/10
225/225 [==============================] - 0s 984us/step - loss: 0.4175 -
accuracy: 0.8346
Epoch 10/10
225/225 [==============================] - 0s 980us/step - loss: 0.4170 -
accuracy: 0.8368
```

[46]:
```
<keras.src.callbacks.History at 0x7f2fcdeaca90>
```

[47]:
```
#model predictions
y_pred = model.predict(x_test_scaled)
y_pred
```

```
75/75 [==============================] - 0s 624us/step
```

[47]:
```
array([[0.4469795 ],
       [0.2629432 ],
       [0.13791387],
       …,
       [0.25922832],
       [0.1283481 ],
       [0.13250448]], dtype=float32)
```

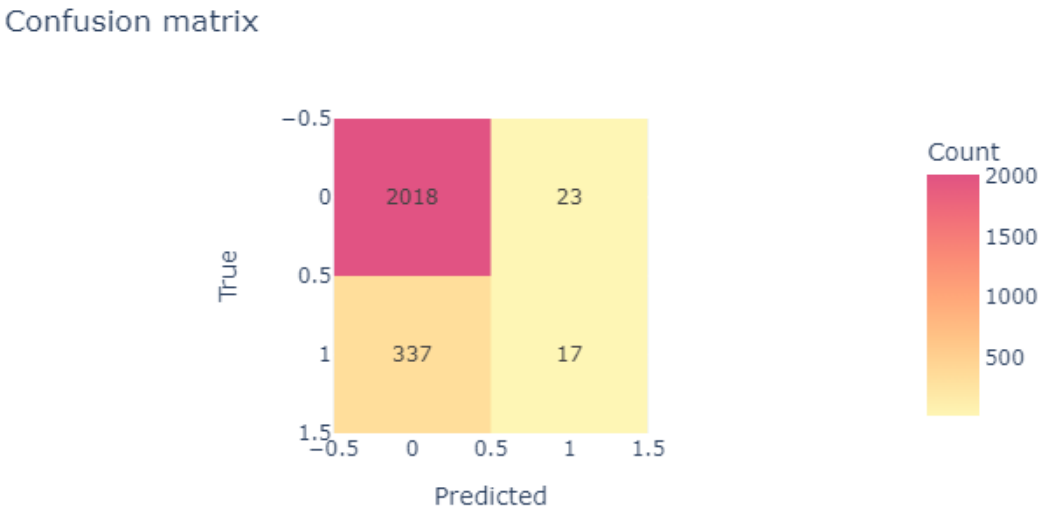
[49]:
```
df = pd.DataFrame({'Prediction': y_pred.flatten(), 'Label': y_test})
print(df.head(5))
```

```
      Prediction  Label
8465    0.446979      1
3270    0.262943      0
7356    0.137914      0
1380    0.227290      0
2607    0.130376      0
```

[50]:
```
#model evaluation
accuracy = accuracy_score(y_test, y_pred.round())
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8496868475991649
```

[52]:
```python
#Confusion Matrix.
import plotly.express as px
# Assuming y_test and y_pred are defined
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred.round())

fig = px.imshow(cm, text_auto=True, color_continuous_scale='Pinkyl',
                labels=dict(x='Predicted', y='True', color='Count'),
                title='Confusion matrix')

fig.show()
```



[53]:
```python
model2 = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation='sigmoid'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(50, activation='sigmoid'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='sigmoid'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
               loss=tf.keras.losses.BinaryCrossentropy(),
               metrics=['accuracy'])
```

```
model2.fit(x_train_scaled, y_train, epochs=10)
```

```
Epoch 1/10
225/225 [==============================] - 1s 1ms/step - loss: 0.4447 -
accuracy: 0.8332
Epoch 2/10
225/225 [==============================] - 0s 996us/step - loss: 0.4318 -
accuracy: 0.8359
Epoch 3/10
225/225 [==============================] - 0s 995us/step - loss: 0.4275 -
accuracy: 0.8359
Epoch 4/10
225/225 [==============================] - 0s 971us/step - loss: 0.4238 -
accuracy: 0.8359
Epoch 5/10
225/225 [==============================] - 0s 984us/step - loss: 0.4193 -
accuracy: 0.8353
Epoch 6/10
225/225 [==============================] - 0s 981us/step - loss: 0.4214 -
accuracy: 0.8359
Epoch 7/10
225/225 [==============================] - 0s 988us/step - loss: 0.4195 -
accuracy: 0.8360
Epoch 8/10
225/225 [==============================] - 0s 989us/step - loss: 0.4152 -
accuracy: 0.8347
Epoch 9/10
225/225 [==============================] - 0s 990us/step - loss: 0.4148 -
accuracy: 0.8345
Epoch 10/10
225/225 [==============================] - 0s 978us/step - loss: 0.4166 -
accuracy: 0.8352
```

[53]:
```
<keras.src.callbacks.History at 0x7f2f2c1eb100>
```

[54]:
```
y_pred2 = model2.predict(x_test_scaled)
y_pred2
```

```
75/75 [==============================] - 0s 640us/step
```

[54]:
```
array([[0.40044606],
       [0.26314956],
       [0.10542049],
       …,
       [0.25139442],
       [0.13323942],
```

```
       [0.1449118 ]], dtype=float32)
```

[55]:
```
#Printing accuracy.
accuracy = accuracy_score(y_test, y_pred2.round())
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8521920668058455
```

[ ]:
```

```