In [1]:
```
#Import required libraries
import torch
import numpy as np
import torch.nn as nn
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Create dataframe to store our dataset
loan_df = pd.read_csv("Dropbox/Coding/AI and ML Bootcamp/Course 4 Deep Learning with K
```

In [3]:
```
#Feature Transformation
loan_df = pd.get_dummies(loan_df, columns=['purpose'], drop_first=True)
print(loan_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  delinq.2yrs  \
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            1

   pub.rec  not.fully.paid  purpose_credit_card  purpose_debt_consolidation  \
0        0               0                    0                           1
1        0               0                    1                           0
2        0               0                    0                           1
3        0               0                    0                           1
4        0               0                    1                           0

   purpose_educational  purpose_home_improvement  purpose_major_purchase  \
0                    0                         0                       0
1                    0                         0                       0
2                    0                         0                       0
3                    0                         0                       0
4                    0                         0                       0

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

In [4]:
```
#Exploratory data analysis
#Look at descriptions of variables
print(loan_df.describe())
```

	credit.policy	int.rate	installment	log.annual.inc	dti
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679
std	0.396245	0.026847	207.071301	0.614813	6.883970
min	0.000000	0.060000	15.670000	7.547502	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.212500
50%	1.000000	0.122100	268.950000	10.928884	12.665000
75%	1.000000	0.140700	432.762500	11.291293	17.950000
max	1.000000	0.216400	940.140000	14.528354	29.960000

	fico	days.with.cr.line	revol.bal	revol.util
count	9578.000000	9578.000000	9.578000e+03	9578.000000
mean	710.846314	4560.767197	1.691396e+04	46.799236
std	37.970537	2496.930377	3.375619e+04	29.014417
min	612.000000	178.958333	0.000000e+00	0.000000
25%	682.000000	2820.000000	3.187000e+03	22.600000
50%	707.000000	4139.958333	8.596000e+03	46.300000
75%	737.000000	5730.000000	1.824950e+04	70.900000
max	827.000000	17639.958330	1.207359e+06	119.000000

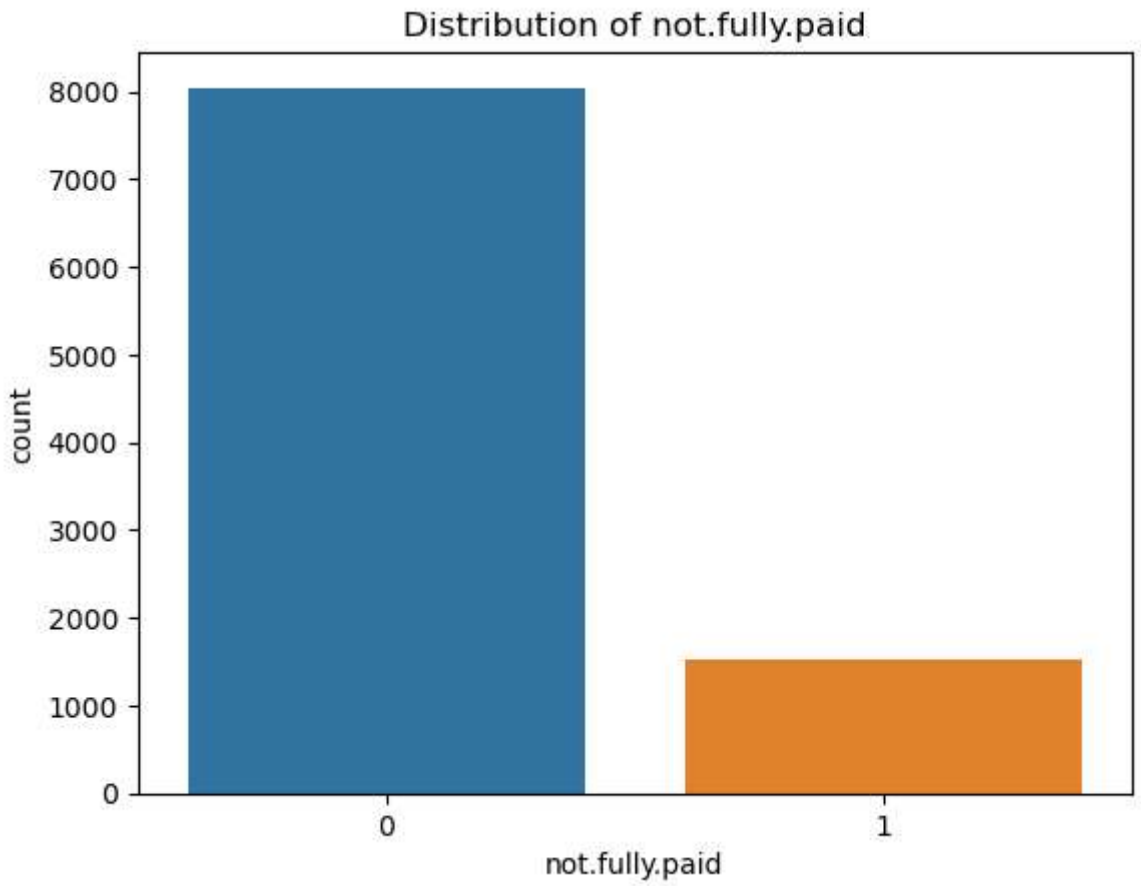
	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
count	9578.000000	9578.000000	9578.000000	9578.000000
mean	1.577469	0.163708	0.062122	0.160054
std	2.200245	0.546215	0.262126	0.366676
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000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

	purpose_credit_card	purpose_debt_consolidation	purpose_educational
count	9578.000000	9578.000000	9578.000000
mean	0.131760	0.413134	0.035811
std	0.338248	0.492422	0.185829
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000

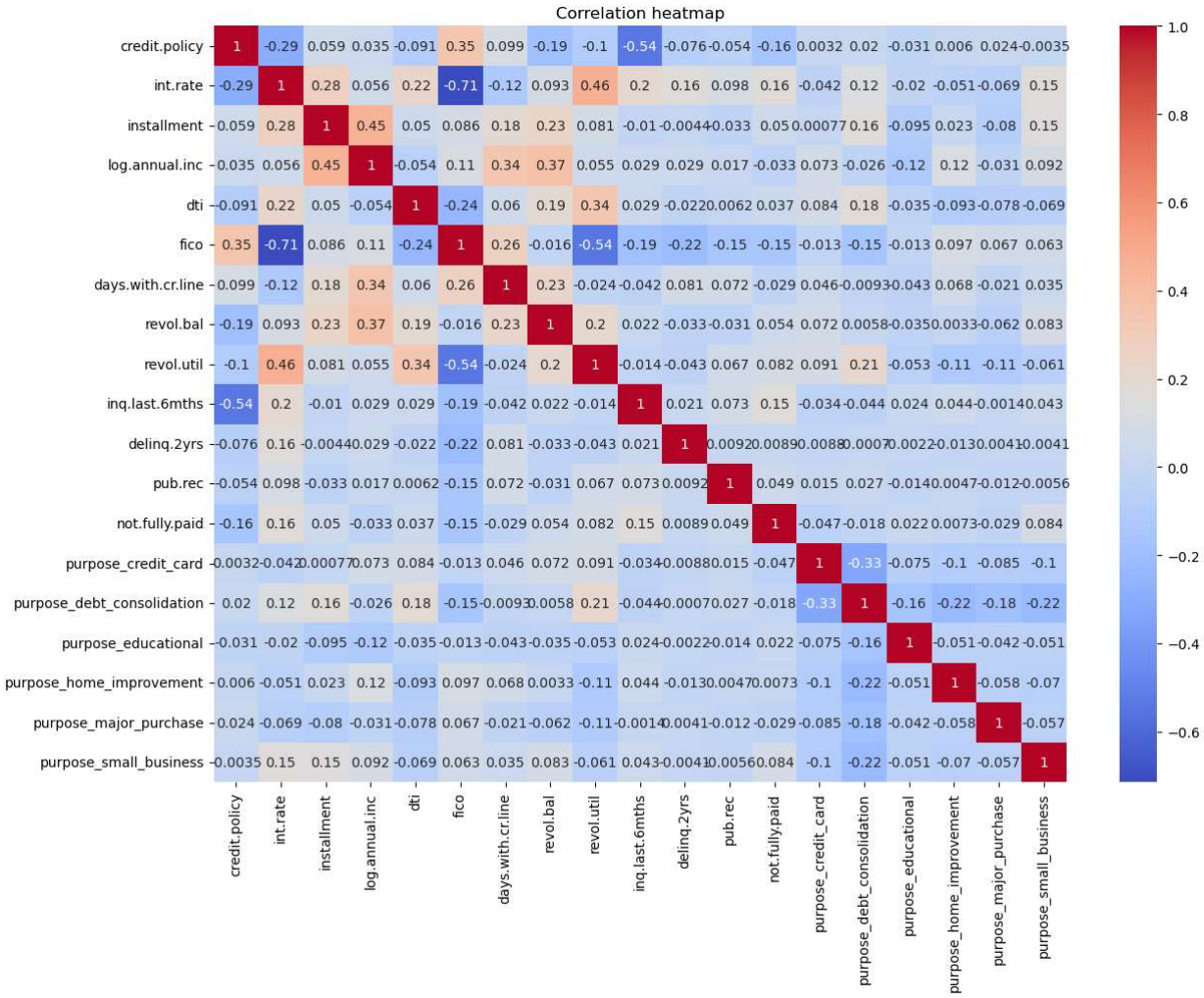
	purpose_home_improvement	purpose_major_purchase
count	9578.000000	9578.000000
mean	0.065671	0.045625
std	0.247720	0.208682
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	1.000000	1.000000

	purpose_small_business
count	9578.000000
mean	0.064627
std	0.245880
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

In [5]:
```
#Check distribution of target variable
sns.countplot(x='not.fully.paid', data=loan_df)
plt.title('Distribution of not.fully.paid')
plt.show()
```

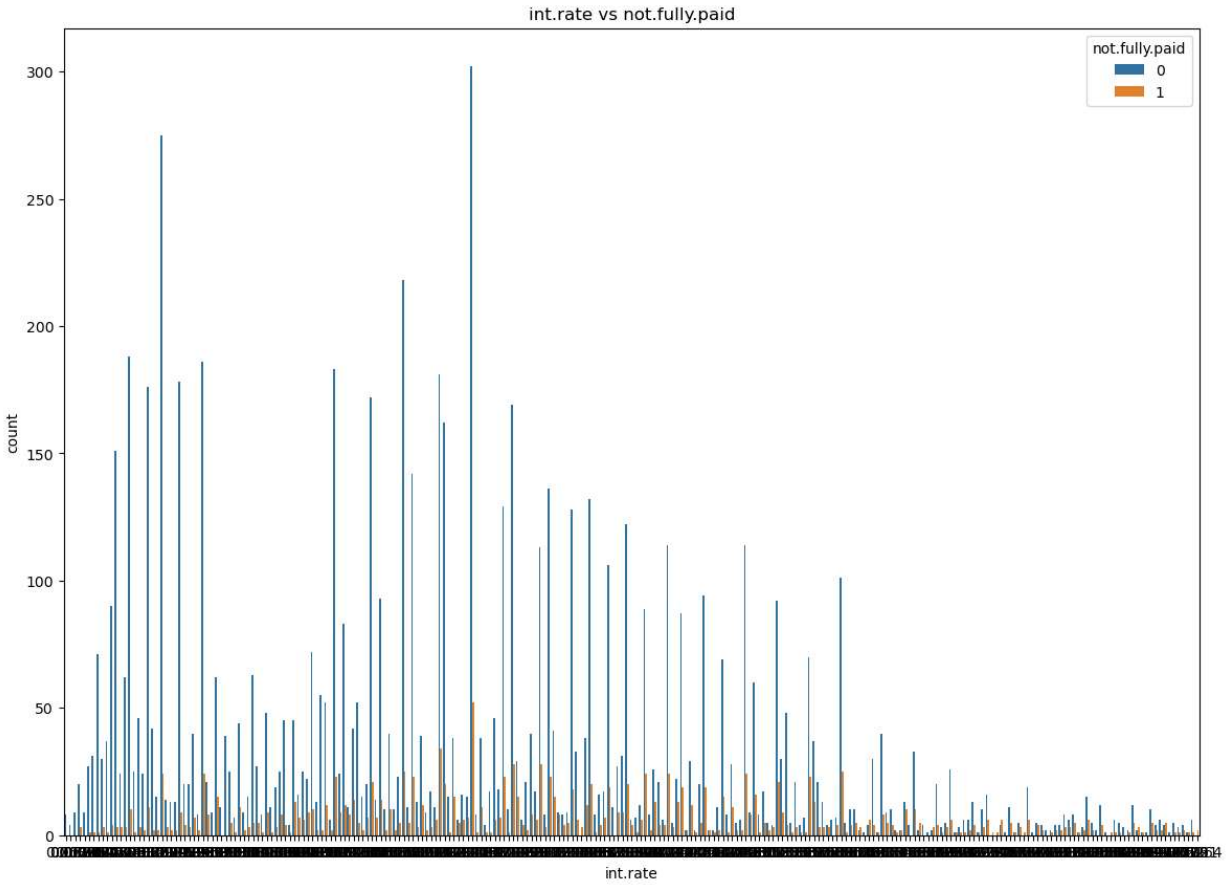


In [6]:
```
#Create correlation matrix to check variable correlations
correlation_matrix = loan_df.corr()
plt.figure(figsize=(14,10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation heatmap')
plt.show()
```

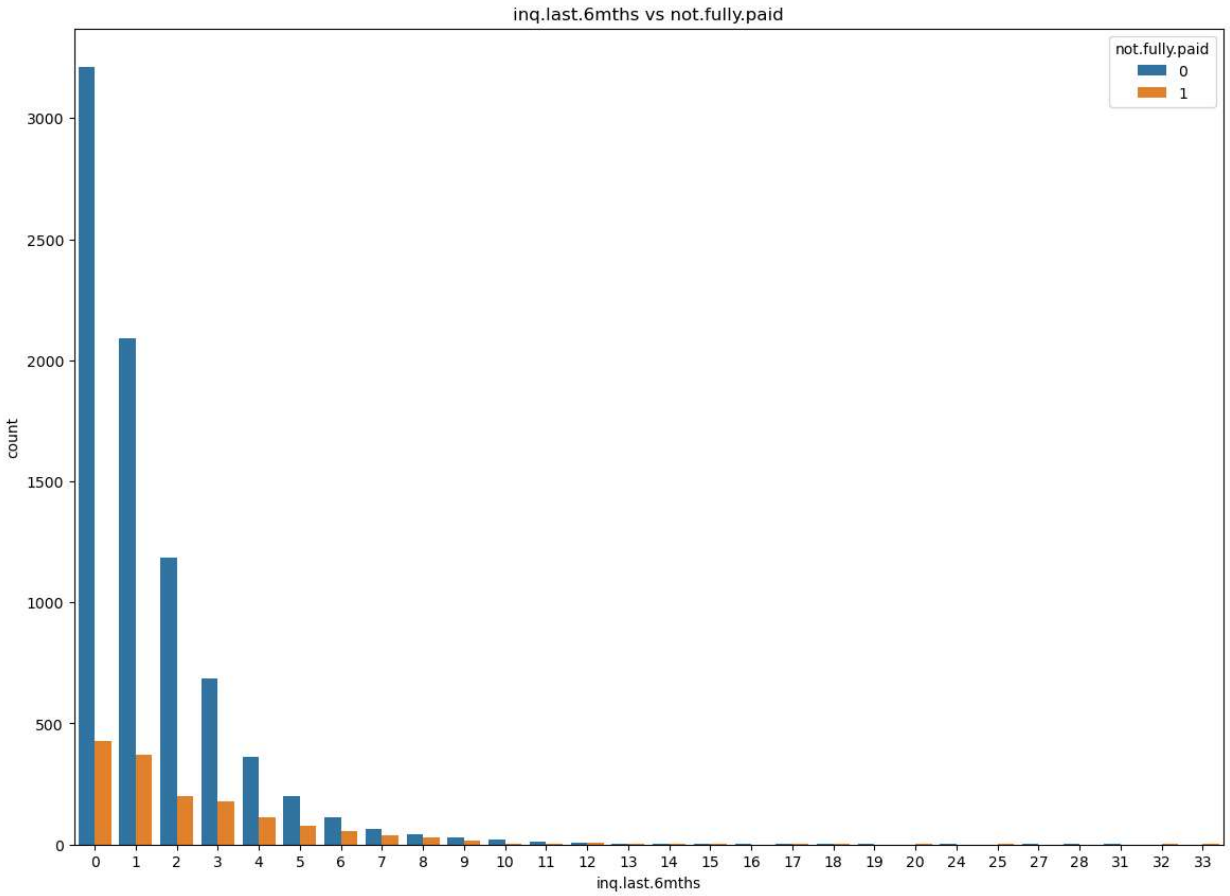


In [7]:
```
#int.rate and inq.last.6mths are most closely correlated to not.fully.paid
#Features with highest correlation are int.rate and revol.util, installment and log.ar
```

In [8]:
```
#'int.rate' variable correlation with not.fully.paid
plt.figure(figsize=(14,10))
sns.countplot(x='int.rate', hue='not.fully.paid', data=loan_df)
plt.title('int.rate vs not.fully.paid')
plt.show()
```



In [9]:
```
#'inq.last.6mths' variable correlation with not.fully.paid
plt.figure(figsize=(14,10))
sns.countplot(x='inq.last.6mths', hue='not.fully.paid', data=loan_df)
plt.title('inq.last.6mths vs not.fully.paid')
plt.show()
```



In [10]:
```
#Additional feature engineering
#Drop features with low correlation to target variable. The least correlated features
loan_df_mod = loan_df.drop('credit.policy', axis=1)
loan_df_mod = loan_df_mod.drop('fico', axis=1)
```

In [11]:
```
#Drop 'log.annual.inc' due to low correlation with target and high correlation with 'i
loan_df_mod = loan_df_mod.drop('log.annual.inc', axis=1)
```

In [12]:
```
#Prepare data for input into model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X = loan_df_mod.drop('not.fully.paid', axis=1)
y = loan_df_mod['not.fully.paid']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [72]:
```
#Build the model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
model = Sequential()
#Input layer
model.add(Dense(128, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.3))
#Hidden layer 1 (with added weight regularization to help fix overfitting)
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.3))
#Hidden layer 2
```

```
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
#Output layer
model.add(Dense(1, activation='sigmoid'))
#Add learning rate to optimizer to reduce overfitting
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
#Compile model
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

In [73]:
```
#View model summary
model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_57 (Dense)            (None, 128)               2048

 dropout_41 (Dropout)        (None, 128)               0

 dense_58 (Dense)            (None, 64)                8256

 dropout_42 (Dropout)        (None, 64)                0

 dense_59 (Dense)            (None, 64)                4160

 dropout_43 (Dropout)        (None, 64)                0

 dense_60 (Dense)            (None, 1)                 65

=================================================================
Total params: 14529 (56.75 KB)
Trainable params: 14529 (56.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [74]:
```
#Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=400, ba
```

```
Epoch 1/400
240/240 [==============================] - 1s 1ms/step - loss: 0.4865 - accuracy: 0.8
370 - val_loss: 0.4297 - val_accuracy: 0.8408
Epoch 2/400
240/240 [==============================] - 0s 933us/step - loss: 0.4401 - accuracy:
0.8395 - val_loss: 0.4188 - val_accuracy: 0.8408
Epoch 3/400
240/240 [==============================] - 0s 937us/step - loss: 0.4353 - accuracy:
0.8393 - val_loss: 0.4271 - val_accuracy: 0.8408
Epoch 4/400
240/240 [==============================] - 0s 921us/step - loss: 0.4337 - accuracy:
0.8395 - val_loss: 0.4258 - val_accuracy: 0.8408
Epoch 5/400
240/240 [==============================] - 0s 929us/step - loss: 0.4342 - accuracy:
0.8393 - val_loss: 0.4219 - val_accuracy: 0.8408
Epoch 6/400
240/240 [==============================] - 0s 920us/step - loss: 0.4339 - accuracy:
0.8389 - val_loss: 0.4190 - val_accuracy: 0.8408
Epoch 7/400
240/240 [==============================] - 0s 916us/step - loss: 0.4302 - accuracy:
0.8393 - val_loss: 0.4179 - val_accuracy: 0.8408
Epoch 8/400
240/240 [==============================] - 0s 912us/step - loss: 0.4326 - accuracy:
0.8396 - val_loss: 0.4190 - val_accuracy: 0.8408
Epoch 9/400
240/240 [==============================] - 0s 912us/step - loss: 0.4285 - accuracy:
0.8391 - val_loss: 0.4176 - val_accuracy: 0.8408
Epoch 10/400
240/240 [==============================] - 0s 912us/step - loss: 0.4323 - accuracy:
0.8399 - val_loss: 0.4255 - val_accuracy: 0.8408
Epoch 11/400
240/240 [==============================] - 0s 912us/step - loss: 0.4310 - accuracy:
0.8397 - val_loss: 0.4216 - val_accuracy: 0.8408
Epoch 12/400
240/240 [==============================] - 0s 922us/step - loss: 0.4334 - accuracy:
0.8397 - val_loss: 0.4229 - val_accuracy: 0.8408
Epoch 13/400
240/240 [==============================] - 0s 921us/step - loss: 0.4310 - accuracy:
0.8396 - val_loss: 0.4228 - val_accuracy: 0.8408
Epoch 14/400
240/240 [==============================] - 0s 933us/step - loss: 0.4286 - accuracy:
0.8392 - val_loss: 0.4193 - val_accuracy: 0.8408
Epoch 15/400
240/240 [==============================] - 0s 941us/step - loss: 0.4306 - accuracy:
0.8396 - val_loss: 0.4203 - val_accuracy: 0.8408
Epoch 16/400
240/240 [==============================] - 0s 954us/step - loss: 0.4293 - accuracy:
0.8396 - val_loss: 0.4212 - val_accuracy: 0.8408
Epoch 17/400
240/240 [==============================] - 0s 929us/step - loss: 0.4303 - accuracy:
0.8395 - val_loss: 0.4363 - val_accuracy: 0.8408
Epoch 18/400
240/240 [==============================] - 0s 921us/step - loss: 0.4341 - accuracy:
0.8397 - val_loss: 0.4267 - val_accuracy: 0.8408
Epoch 19/400
240/240 [==============================] - 0s 921us/step - loss: 0.4296 - accuracy:
0.8397 - val_loss: 0.4186 - val_accuracy: 0.8408
Epoch 20/400
240/240 [==============================] - 0s 929us/step - loss: 0.4317 - accuracy:
0.8397 - val_loss: 0.4173 - val_accuracy: 0.8408
```

```
Epoch 361/400
240/240 [==============================] - 0s 916us/step - loss: 0.4336 - accuracy:
0.8397 - val_loss: 0.4278 - val_accuracy: 0.8408
Epoch 362/400
240/240 [==============================] - 0s 921us/step - loss: 0.4333 - accuracy:
0.8397 - val_loss: 0.4253 - val_accuracy: 0.8408
Epoch 363/400
240/240 [==============================] - 0s 920us/step - loss: 0.4372 - accuracy:
0.8397 - val_loss: 0.4313 - val_accuracy: 0.8408
Epoch 364/400
240/240 [==============================] - 0s 916us/step - loss: 0.4385 - accuracy:
0.8397 - val_loss: 0.4350 - val_accuracy: 0.8408
Epoch 365/400
240/240 [==============================] - 0s 967us/step - loss: 0.4378 - accuracy:
0.8397 - val_loss: 0.4274 - val_accuracy: 0.8408
Epoch 366/400
240/240 [==============================] - 0s 971us/step - loss: 0.4401 - accuracy:
0.8397 - val_loss: 0.4299 - val_accuracy: 0.8408
Epoch 367/400
240/240 [==============================] - 0s 983us/step - loss: 0.4343 - accuracy:
0.8397 - val_loss: 0.4293 - val_accuracy: 0.8408
Epoch 368/400
240/240 [==============================] - 0s 992us/step - loss: 0.4350 - accuracy:
0.8397 - val_loss: 0.4266 - val_accuracy: 0.8408
Epoch 369/400
240/240 [==============================] - 0s 975us/step - loss: 0.4347 - accuracy:
0.8397 - val_loss: 0.4231 - val_accuracy: 0.8408
Epoch 370/400
240/240 [==============================] - 0s 975us/step - loss: 0.4326 - accuracy:
0.8397 - val_loss: 0.4291 - val_accuracy: 0.8408
Epoch 371/400
240/240 [==============================] - 0s 933us/step - loss: 0.4368 - accuracy:
0.8397 - val_loss: 0.4240 - val_accuracy: 0.8408
Epoch 372/400
240/240 [==============================] - 0s 923us/step - loss: 0.4347 - accuracy:
0.8397 - val_loss: 0.4270 - val_accuracy: 0.8408
Epoch 373/400
240/240 [==============================] - 0s 962us/step - loss: 0.4311 - accuracy:
0.8397 - val_loss: 0.4302 - val_accuracy: 0.8408
Epoch 374/400
240/240 [==============================] - 0s 962us/step - loss: 0.4324 - accuracy:
0.8397 - val_loss: 0.4302 - val_accuracy: 0.8408
Epoch 375/400
240/240 [==============================] - 0s 967us/step - loss: 0.4325 - accuracy:
0.8397 - val_loss: 0.4297 - val_accuracy: 0.8408
Epoch 376/400
240/240 [==============================] - 0s 925us/step - loss: 0.4328 - accuracy:
0.8397 - val_loss: 0.4272 - val_accuracy: 0.8408
Epoch 377/400
240/240 [==============================] - 0s 1ms/step - loss: 0.4324 - accuracy: 0.8
397 - val_loss: 0.4360 - val_accuracy: 0.8408
Epoch 378/400
240/240 [==============================] - 0s 971us/step - loss: 0.4360 - accuracy:
0.8397 - val_loss: 0.4282 - val_accuracy: 0.8408
Epoch 379/400
240/240 [==============================] - 0s 933us/step - loss: 0.4364 - accuracy:
0.8397 - val_loss: 0.4267 - val_accuracy: 0.8408
Epoch 380/400
240/240 [==============================] - 0s 946us/step - loss: 0.4306 - accuracy:
0.8397 - val_loss: 0.4235 - val_accuracy: 0.8408
```

```
Epoch 381/400
240/240 [==============================] - 0s 912us/step - loss: 0.4350 - accuracy:
0.8397 - val_loss: 0.4333 - val_accuracy: 0.8408
Epoch 382/400
240/240 [==============================] - 0s 916us/step - loss: 0.4371 - accuracy:
0.8397 - val_loss: 0.4339 - val_accuracy: 0.8408
Epoch 383/400
240/240 [==============================] - 0s 971us/step - loss: 0.4377 - accuracy:
0.8397 - val_loss: 0.4366 - val_accuracy: 0.8408
Epoch 384/400
240/240 [==============================] - 0s 920us/step - loss: 0.4367 - accuracy:
0.8397 - val_loss: 0.4246 - val_accuracy: 0.8408
Epoch 385/400
240/240 [==============================] - 0s 921us/step - loss: 0.4378 - accuracy:
0.8397 - val_loss: 0.4271 - val_accuracy: 0.8408
Epoch 386/400
240/240 [==============================] - 0s 962us/step - loss: 0.4340 - accuracy:
0.8397 - val_loss: 0.4254 - val_accuracy: 0.8408
Epoch 387/400
240/240 [==============================] - 0s 925us/step - loss: 0.4349 - accuracy:
0.8397 - val_loss: 0.4305 - val_accuracy: 0.8408
Epoch 388/400
240/240 [==============================] - 0s 916us/step - loss: 0.4382 - accuracy:
0.8397 - val_loss: 0.4272 - val_accuracy: 0.8408
Epoch 389/400
240/240 [==============================] - 0s 926us/step - loss: 0.4337 - accuracy:
0.8397 - val_loss: 0.4295 - val_accuracy: 0.8408
Epoch 390/400
240/240 [==============================] - 0s 912us/step - loss: 0.4340 - accuracy:
0.8397 - val_loss: 0.4277 - val_accuracy: 0.8408
Epoch 391/400
240/240 [==============================] - 0s 916us/step - loss: 0.4347 - accuracy:
0.8397 - val_loss: 0.4294 - val_accuracy: 0.8408
Epoch 392/400
240/240 [==============================] - 0s 912us/step - loss: 0.4325 - accuracy:
0.8397 - val_loss: 0.4329 - val_accuracy: 0.8408
Epoch 393/400
240/240 [==============================] - 0s 917us/step - loss: 0.4399 - accuracy:
0.8397 - val_loss: 0.4333 - val_accuracy: 0.8408
Epoch 394/400
240/240 [==============================] - 0s 908us/step - loss: 0.4373 - accuracy:
0.8397 - val_loss: 0.4342 - val_accuracy: 0.8408
Epoch 395/400
240/240 [==============================] - 0s 975us/step - loss: 0.4348 - accuracy:
0.8397 - val_loss: 0.4304 - val_accuracy: 0.8408
Epoch 396/400
240/240 [==============================] - 0s 912us/step - loss: 0.4383 - accuracy:
0.8397 - val_loss: 0.4320 - val_accuracy: 0.8408
Epoch 397/400
240/240 [==============================] - 0s 921us/step - loss: 0.4391 - accuracy:
0.8397 - val_loss: 0.4372 - val_accuracy: 0.8408
Epoch 398/400
240/240 [==============================] - 0s 916us/step - loss: 0.4348 - accuracy:
0.8397 - val_loss: 0.4312 - val_accuracy: 0.8408
Epoch 399/400
240/240 [==============================] - 0s 921us/step - loss: 0.4343 - accuracy:
0.8397 - val_loss: 0.4281 - val_accuracy: 0.8408
Epoch 400/400
240/240 [==============================] - 0s 962us/step - loss: 0.4341 - accuracy:
0.8397 - val_loss: 0.4290 - val_accuracy: 0.8408
```

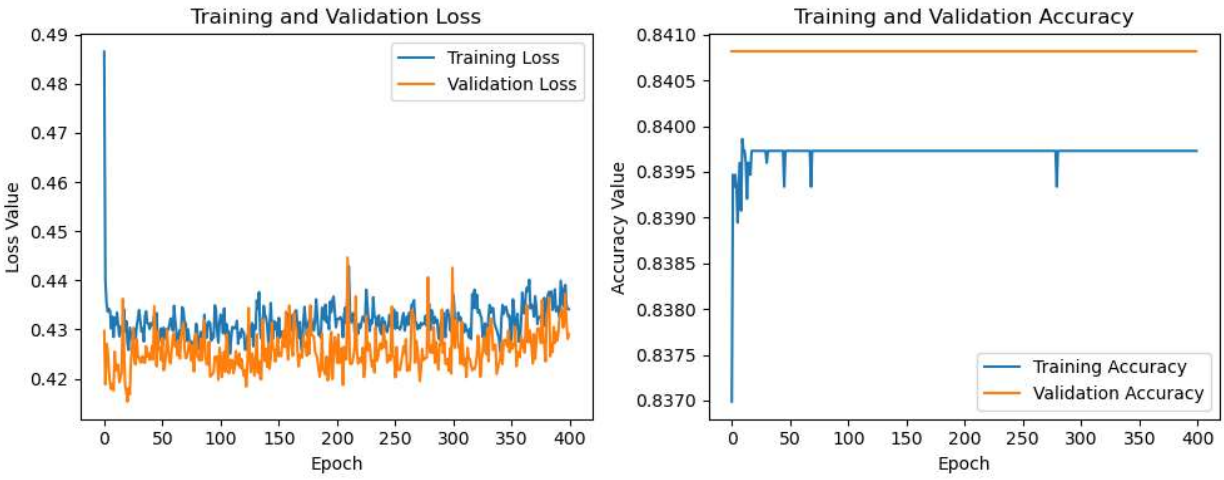
In [75]:
```
#Evaluate the matrix
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
60/60 [==============================] - 0s 475us/step - loss: 0.4290 - accuracy: 0.8
408
Test Loss: 0.4290
Test Accuracy: 0.8408
```

In [78]:
```
#Plot training and validation loss
plt.figure(figsize=(10,4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.legend()

#Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy Value')
plt.legend()

plt.tight_layout()
plt.show()
```



During exploratory data analysis and feature engineering, the variables 'credit.policy' and 'fico' were dropped because they had the lowest correlation with the target variable not.fully.paid. The variable 'log.annual.inc' was dropped because it had a high correlation with 'installment' and a relatively low correlation with the target. The model was initially built with one hidden layer, an unspecified learning rate and no regularizer attached. The results heavily suggested overfitting, and the model had a low initial validation accuracy and high validation loss. A regularizer was added to the hidden layer to reduce overfitting and multiple values were tested to find the best result. The optimizer was then given a learning rate, and 0.1 was found to be

the best value. After the validation results were better but still somewhat unsatisfying, another two hidden layers were added, but only two hidden layers were kept because having three yielded unfavorable results.

In [ ]: