Lending Club Loan Project

Danny Mathieson - March 2022

Downloading and Displaying the Dataset

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
df = pd.read_csv('datasets/loan_data.csv')
df.head()
```

Out[3]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	re
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	

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

Out[4]:
```
(9578, 14)
```

In [5]:
```
# check for nulls
df.isna().sum()
```

Out[5]:
```
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
```

1. Feature Transformation - Transform Categorical Values into Numerical Values

In [6]:
```
df.dtypes
```

Out[6]:
```
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
```

In [7]:
```
# Only purpose needs to be changed to numerical values - get dummies
df_dummy = pd.get_dummies(df, drop_first=True)
df_dummy.head()
```

Out[7]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5

In [8]:
```
df_dummy.columns
```

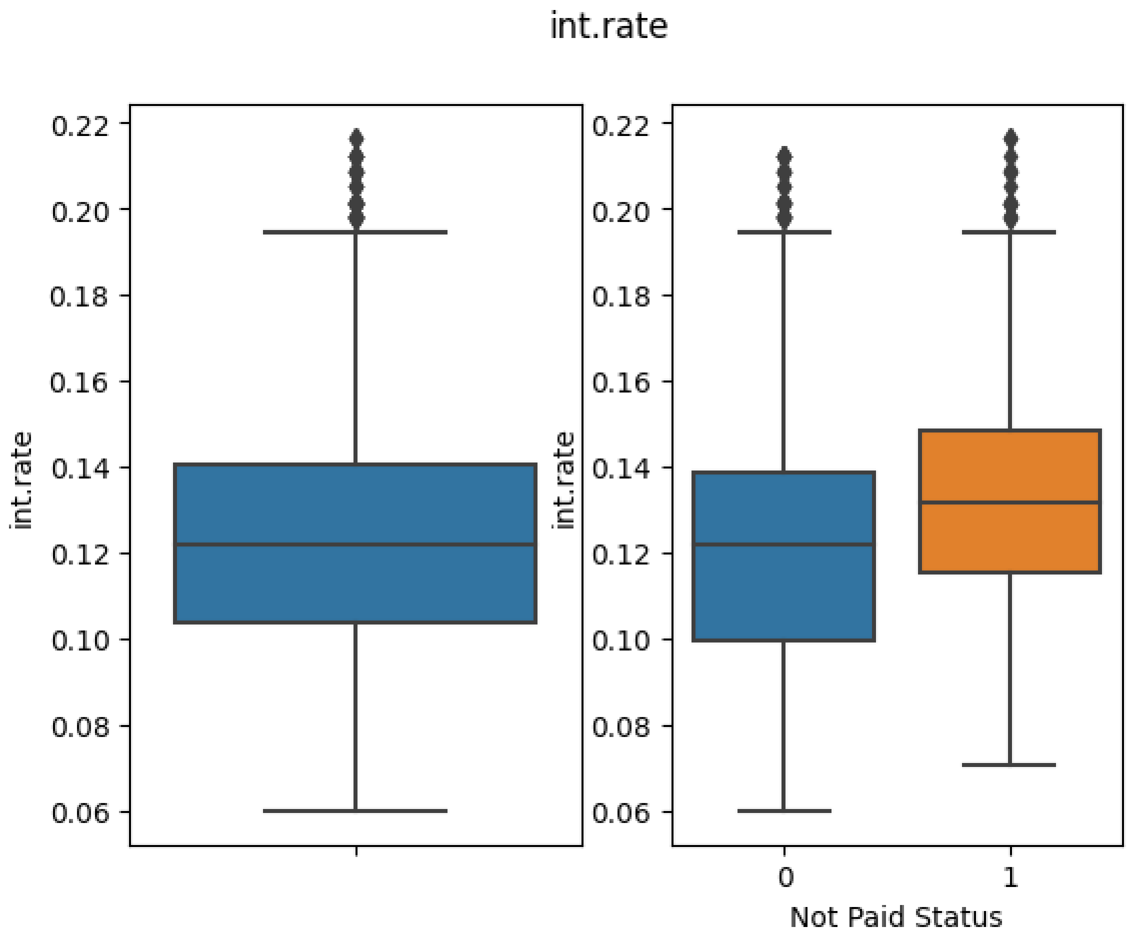
Out[8]:
```
Index(['credit.policy', 'int.rate', 'installment', 'log.annual.inc', 'dti',
       'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement',
       'purpose_major_purchase', 'purpose_small_business'],
      dtype='object')
```

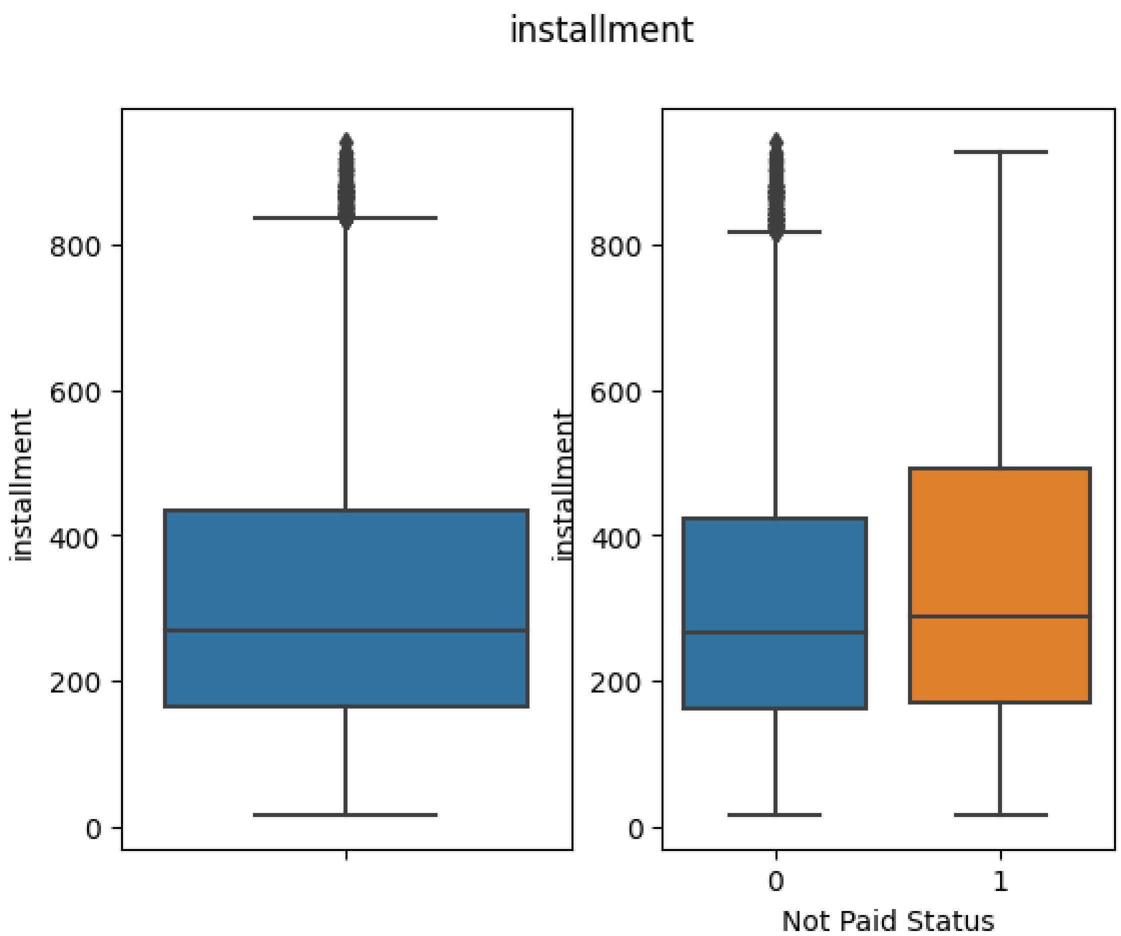
2. EDA on Different Factors of the Dataset

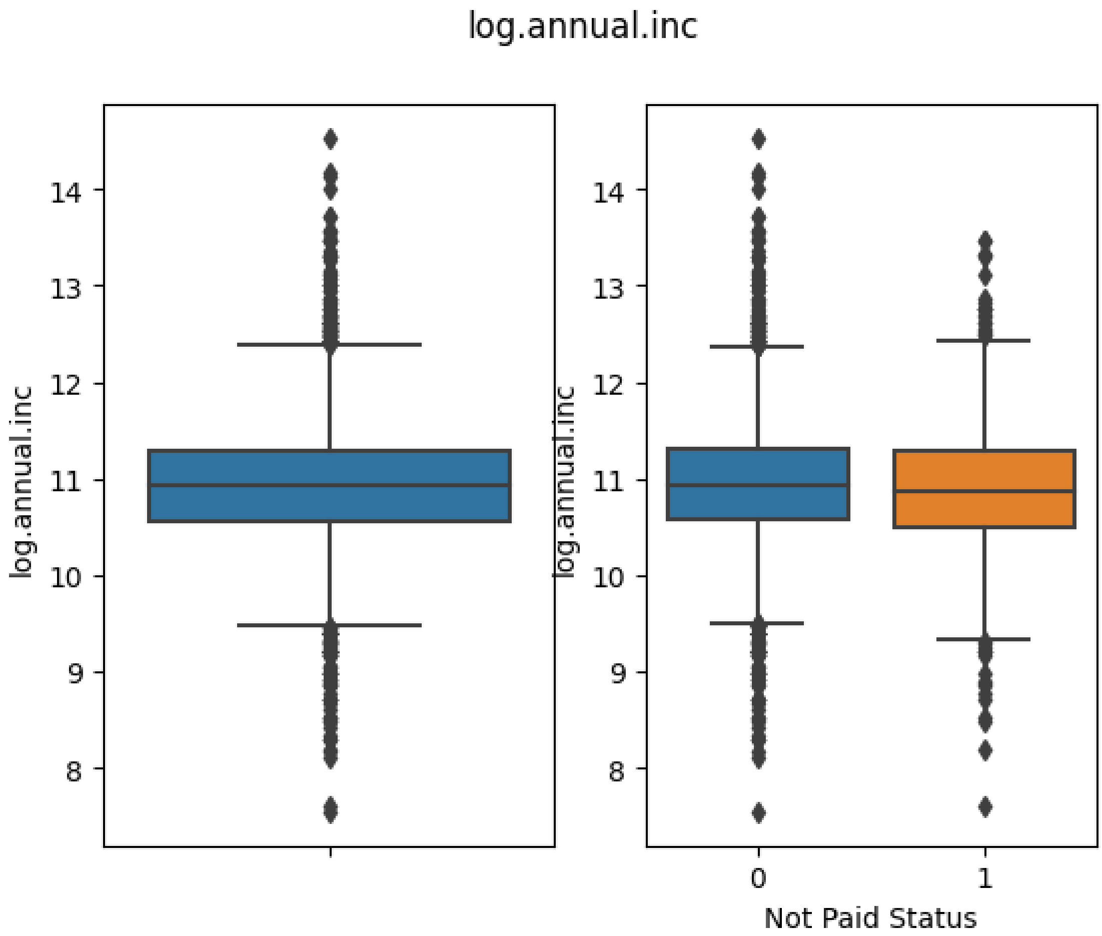
In [9]:
```
# Describe the data, split columns into either binary or numerical sub-types
df_dummy.describe()
numerical_cols = ['int.rate','installment','log.annual.inc','dti','fico','days.with.c
binary_cols = ['credit.policy','purpose_credit_card', 'purpose_debt_consolidation','p
```

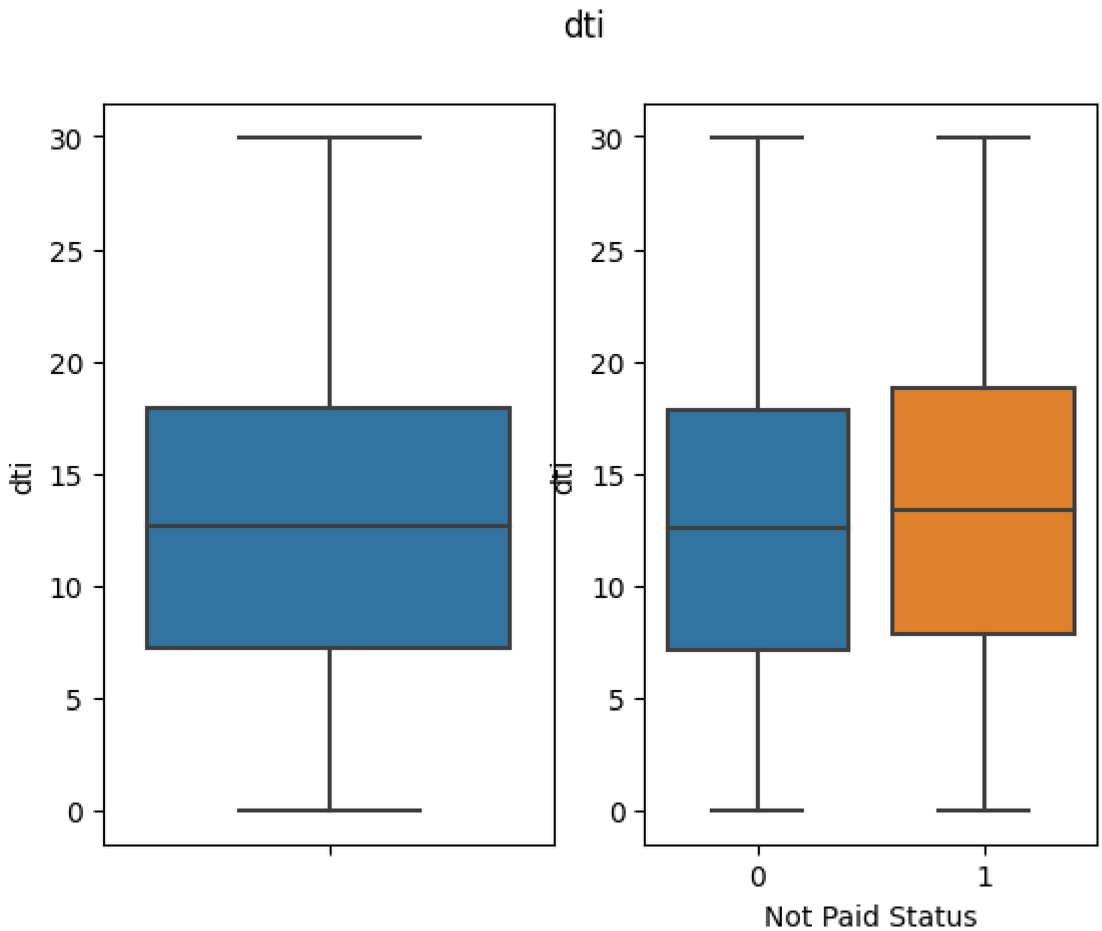
In [10]:
```
from matplotlib import pyplot as plt
import seaborn as sns
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']
```

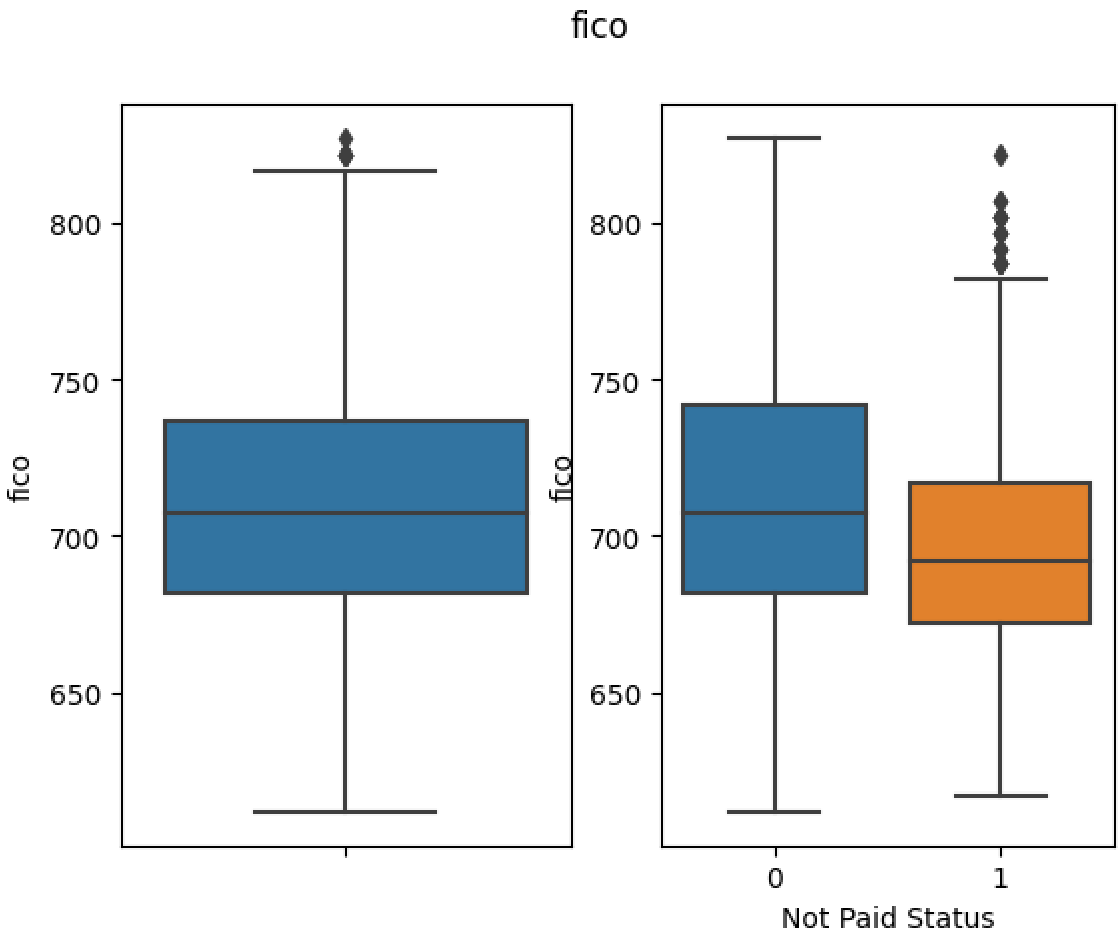
In [11]:
```
# Loop over numerical columns - Box Plot overall and by fully paid status
for c in numerical_cols:
    # get datasets by category
    total_data = df_dummy[[c]]
    cat_data = df_dummy[[c, 'not.fully.paid']]
    # create boxplots
    sns.boxplot(x=None, y=c, data=total_data, ax=plt.subplot(1,2,1))
    sns.boxplot(x='not.fully.paid', y=c, data=cat_data, ax=plt.subplot(1,2,2))
    # format chart and show
    plt.suptitle(c)
    plt.xlabel('Not Paid Status')
    plt.show()
```

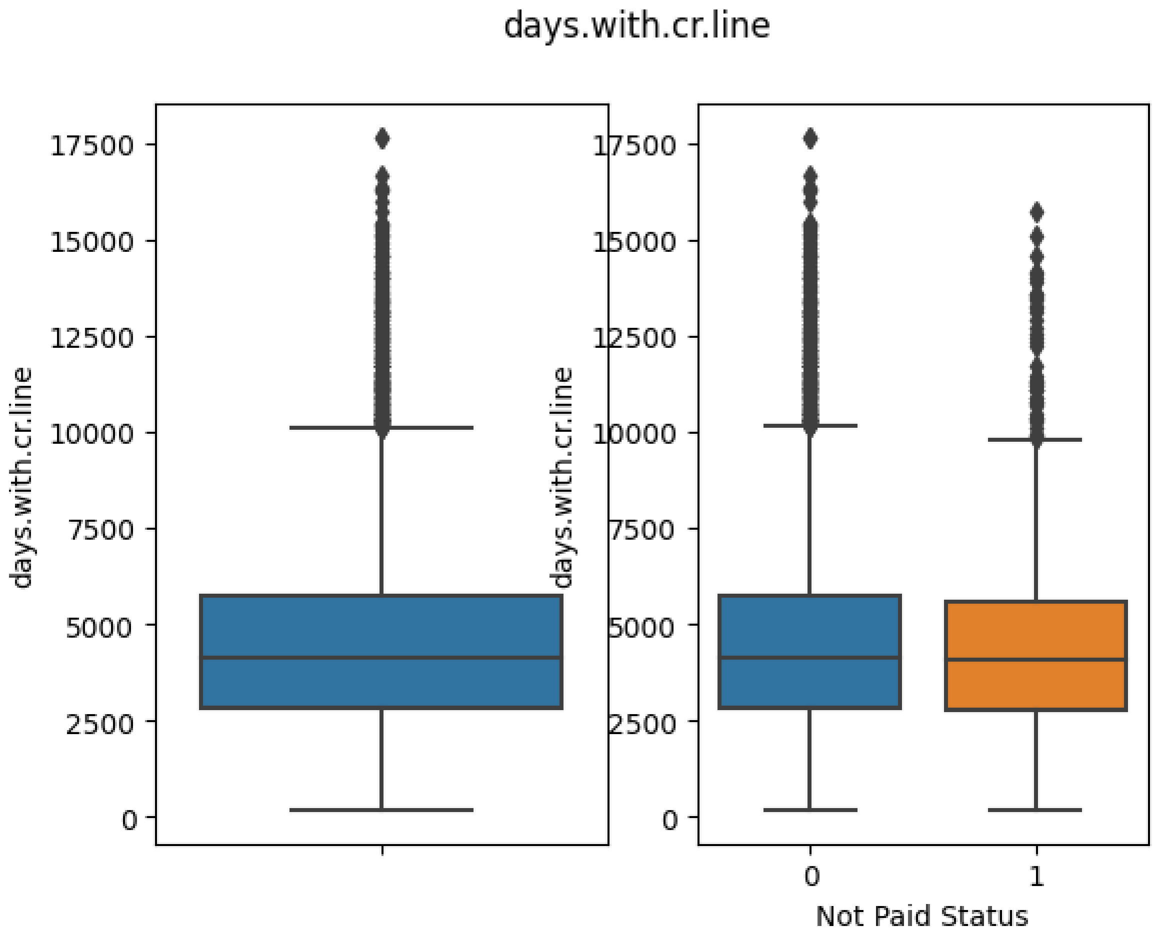

int.rate
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
int.rate
0
1
Not Paid Status

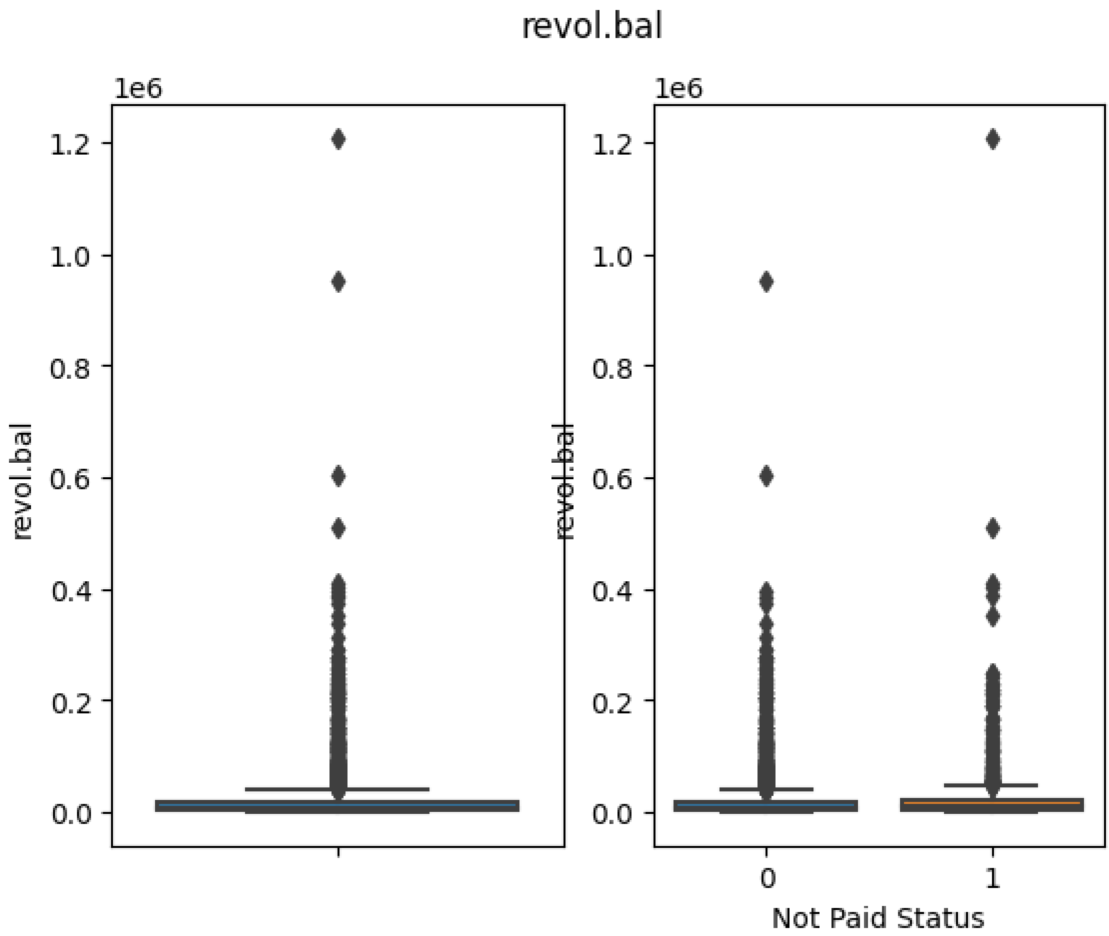

installment
800
600
400
200
0
installment
0
1
Not Paid Status

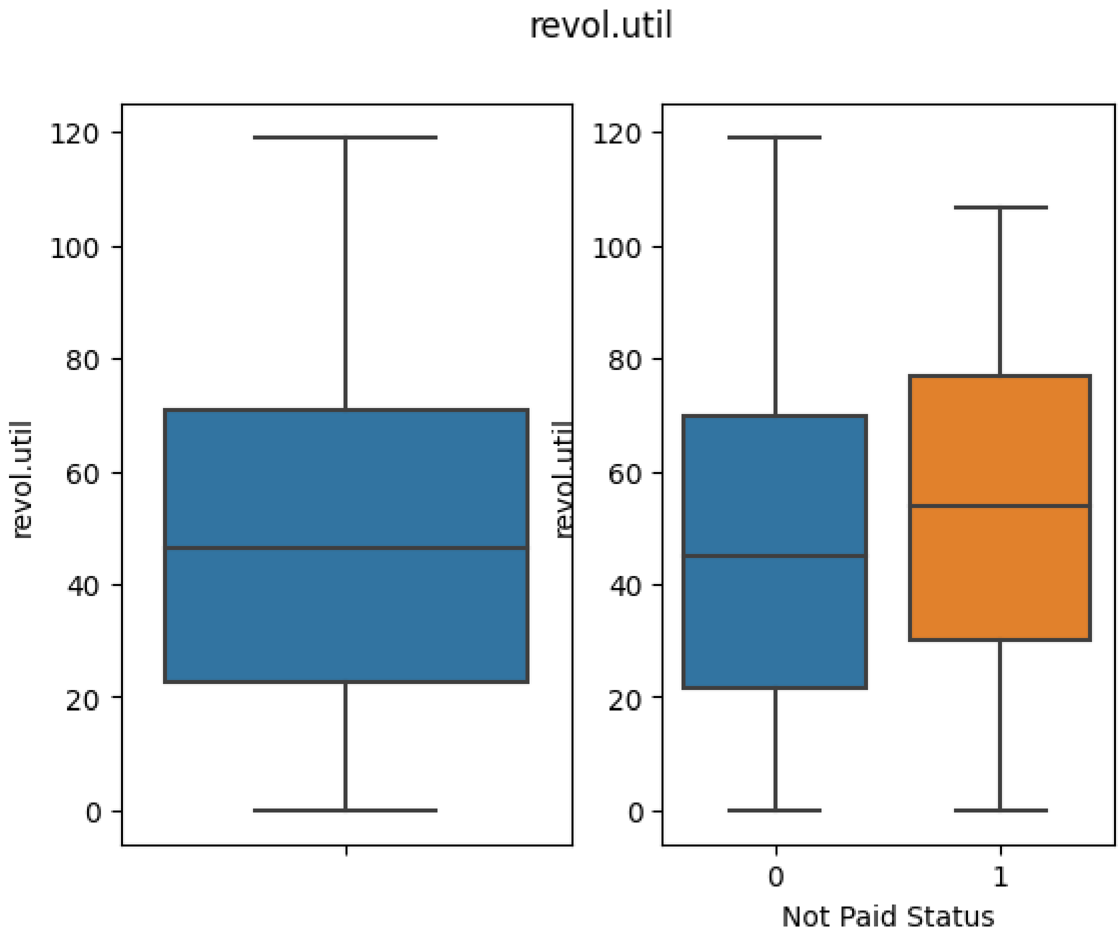

log.annual.inc
log.annual.inc
Not Paid Status
0
1

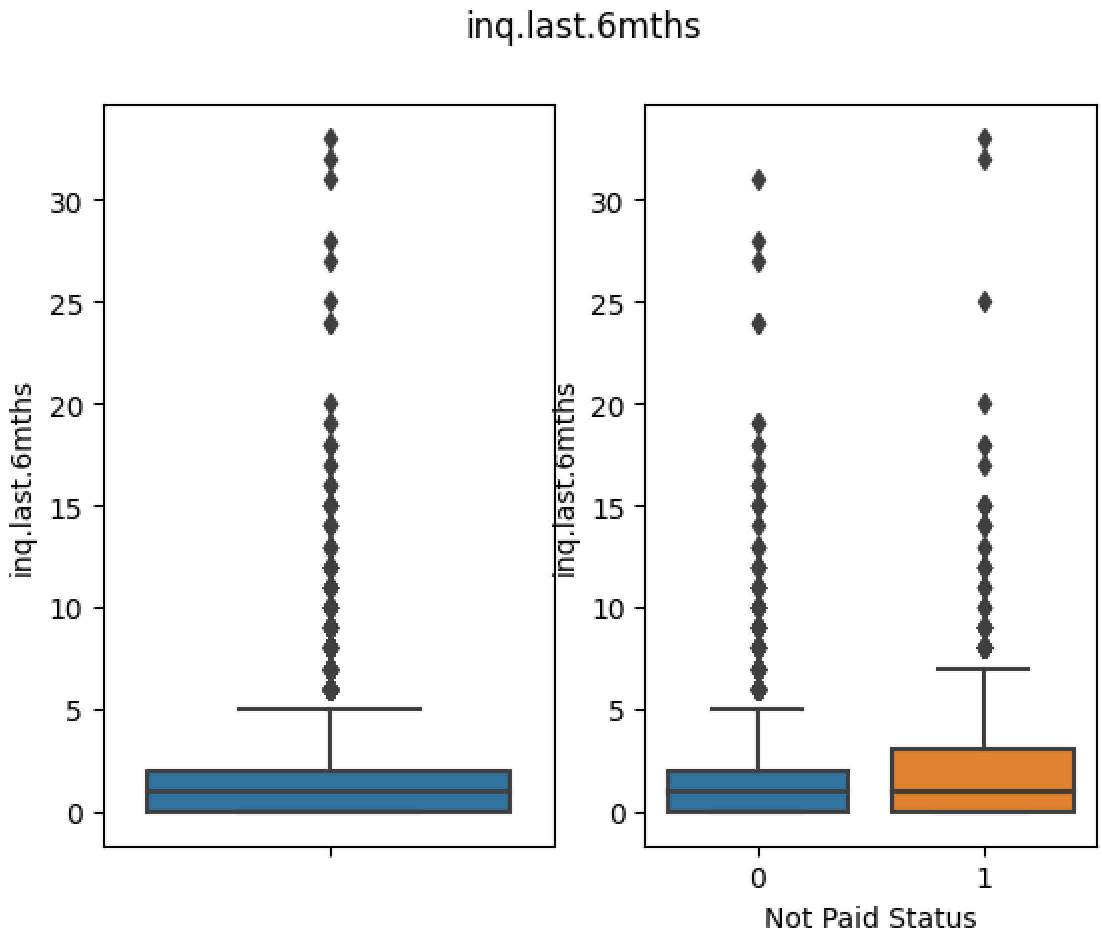

dti
dti
Not Paid Status
0
1

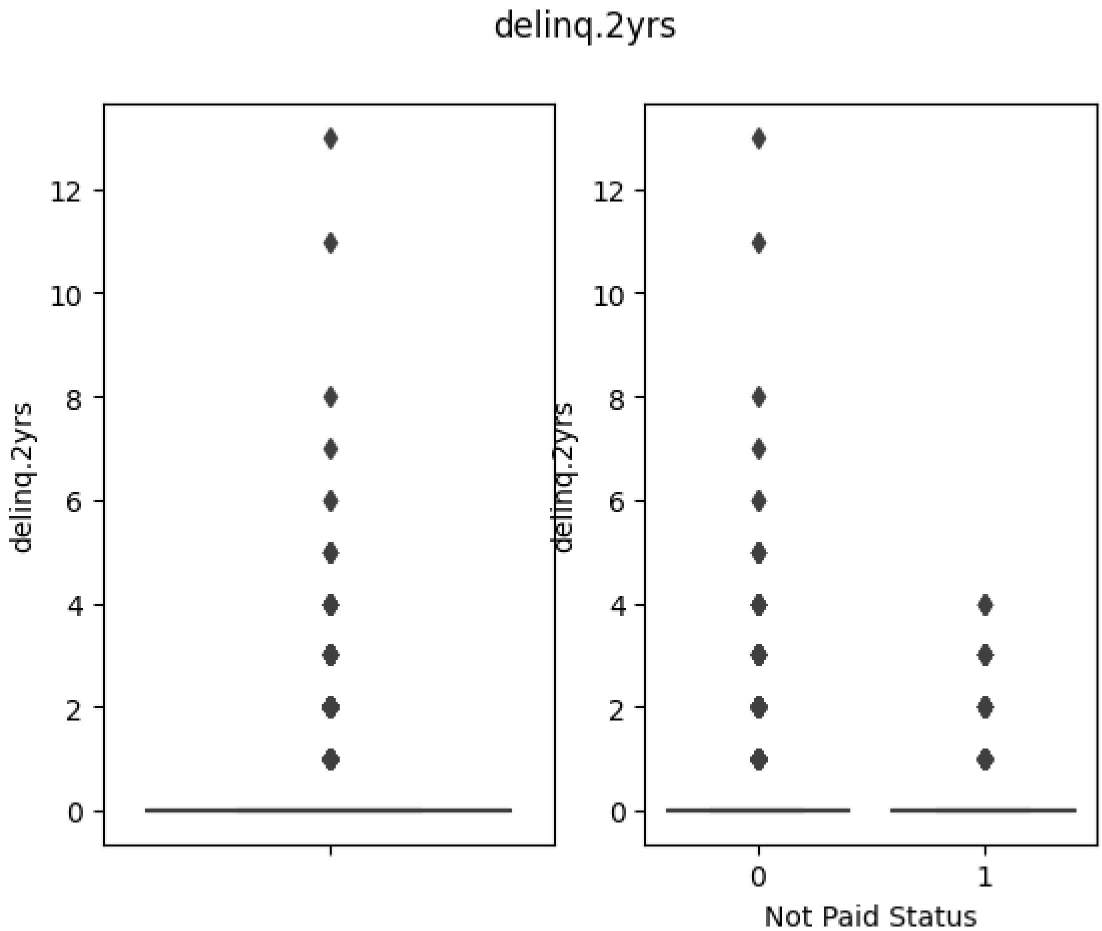

fico
fico
800
750
700
650
fico
800
750
700
650
0
1
Not Paid Status

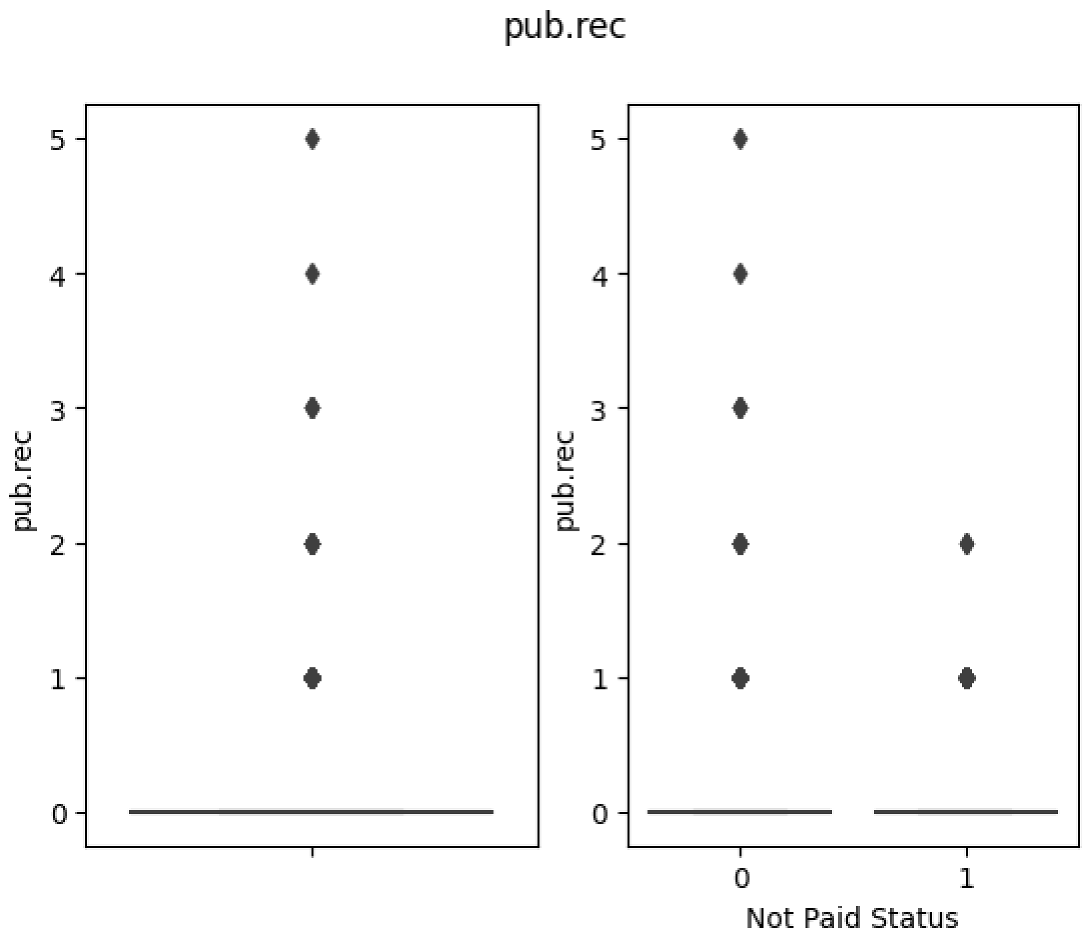

days.with.cr.line
days.with.cr.line
17500
15000
12500
10000
7500
5000
2500
0
days.with.cr.line
17500
15000
12500
10000
7500
5000
2500
0
0
1
Not Paid Status


revol.bal
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
revol.bal
0
1
Not Paid Status


revol.util
120
100
80
60
40
20
0
revol.util
0
1
Not Paid Status


inq.last.6mths
inq.last.6mths
0
5
10
15
20
25
30
inq.last.6mths
0
1
Not Paid Status


delinq.2yrs
delinq.2yrs
0
2
4
6
8
10
12
delinq.2yrs
0
1
Not Paid Status



- int.rate

    - Generally int.rate seems to correspond with a higher liklihood of not paying as the 25%, Median, and 75% are all higher than the paid class by a full percent or two
    - The upper extreme doesn't seem to impact the outcome much as the max percent excluding outliers is the same and they seem to have even numbers of outliers on the high end
    - interesting how no one with a sub 7% rate defaulted

- installment

    - Once we start to get above 3.5 years the liklihood of default seems to go up, especially above 5 years

- log.annual.inc - nothing

- dti - nothing

- fico

    - On first glance, lower fico scores definitely seem to have an impact on payback liklihood, but the entire range of paid back loans' fico scores contains the range of not paid back loans, including outliers.

- days.with.cr.line - nothing

- revol.bal - scale is too messed up to see much

- revol.util

    - higher utilization rates has a slight impact on not paying back fully

- inq.last.6mths

```
- slightly impactful if above 2
```

- delinq.2yrs

```
- not enough non-zero data
- more outliers that have paid pack than haven't
```

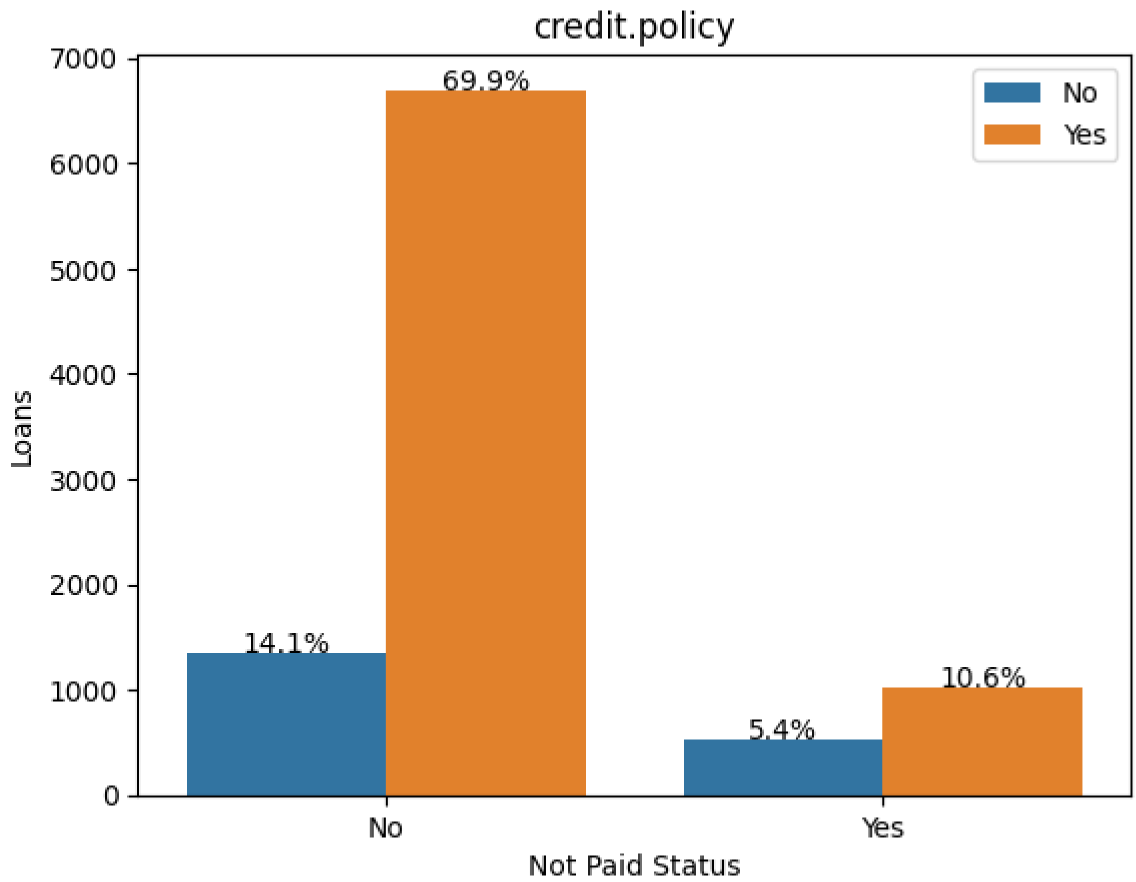
pub.rec

```
- not enough non-zero data
- more outliers that have paid pack than haven't
```

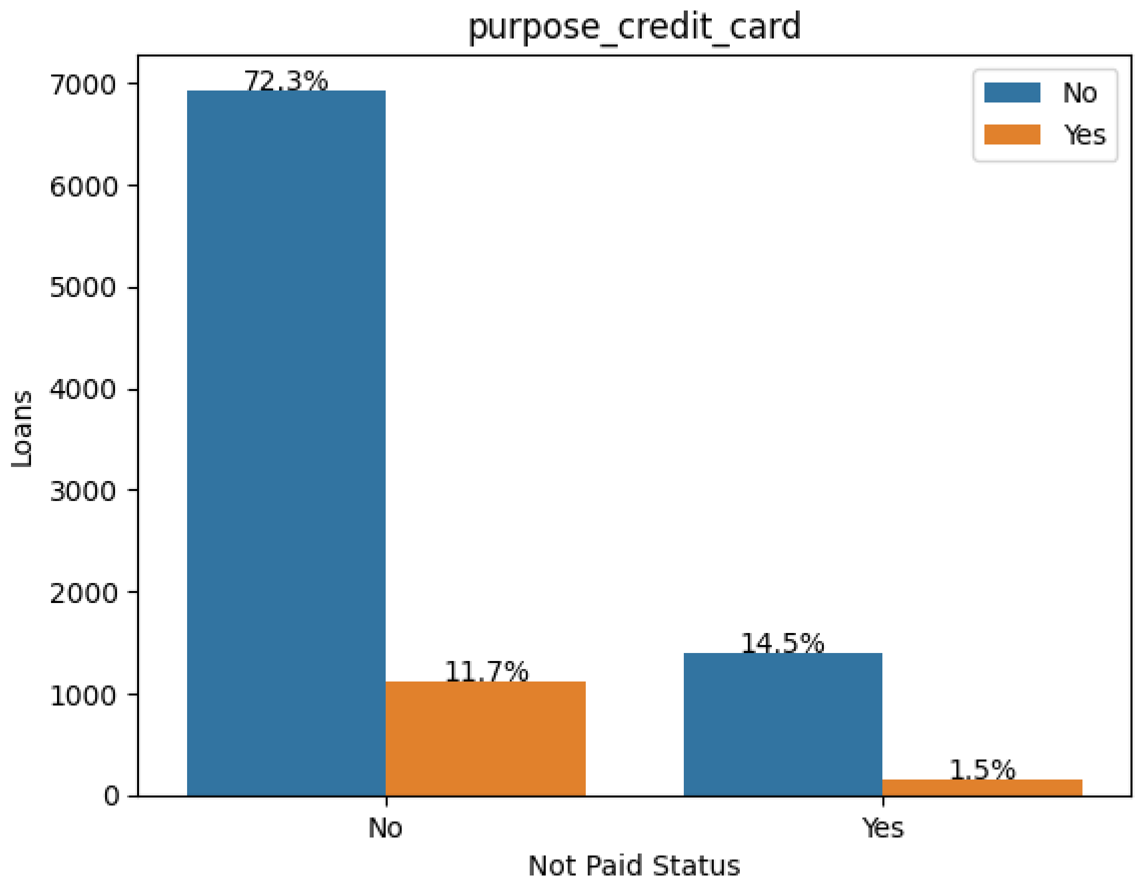
In [12]:

```python
# Loop over Binary Columns & create a bar plot & stacked bar plot
for c in binary_cols:
    bin_data = df_dummy[[c, 'not.fully.paid']]
    bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
    bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
    # calculate percentages
    percent = bin_data.groupby(['Condition', 'Paid Status']).size().reset_index(name=
    percent['pct'] = percent['count'] / percent['count'].sum() * 100
    # create the plot
    order = {
        'Paid Status': ['No','Yes'],
        'Condition': ['No','Yes']
    }
    axis = sns.countplot(x='Paid Status', hue='Condition', data=bin_data, order=order
    # add percentages for tooltips
    counter=0
    for p in axis.patches:
        h = p.get_height()
        pct = f"{round(percent['pct'][counter],1)}%"
        x_ax_pos = p.get_x() + p.get_width() / 2.0
        counter += 1
        axis.text(
            x_ax_pos,
            h + 3,
            pct,
            ha='center'
        )
    plt.title(c)
    plt.xlabel('Not Paid Status')
    plt.ylabel('Loans')
    plt.legend()
    plt.show()
```

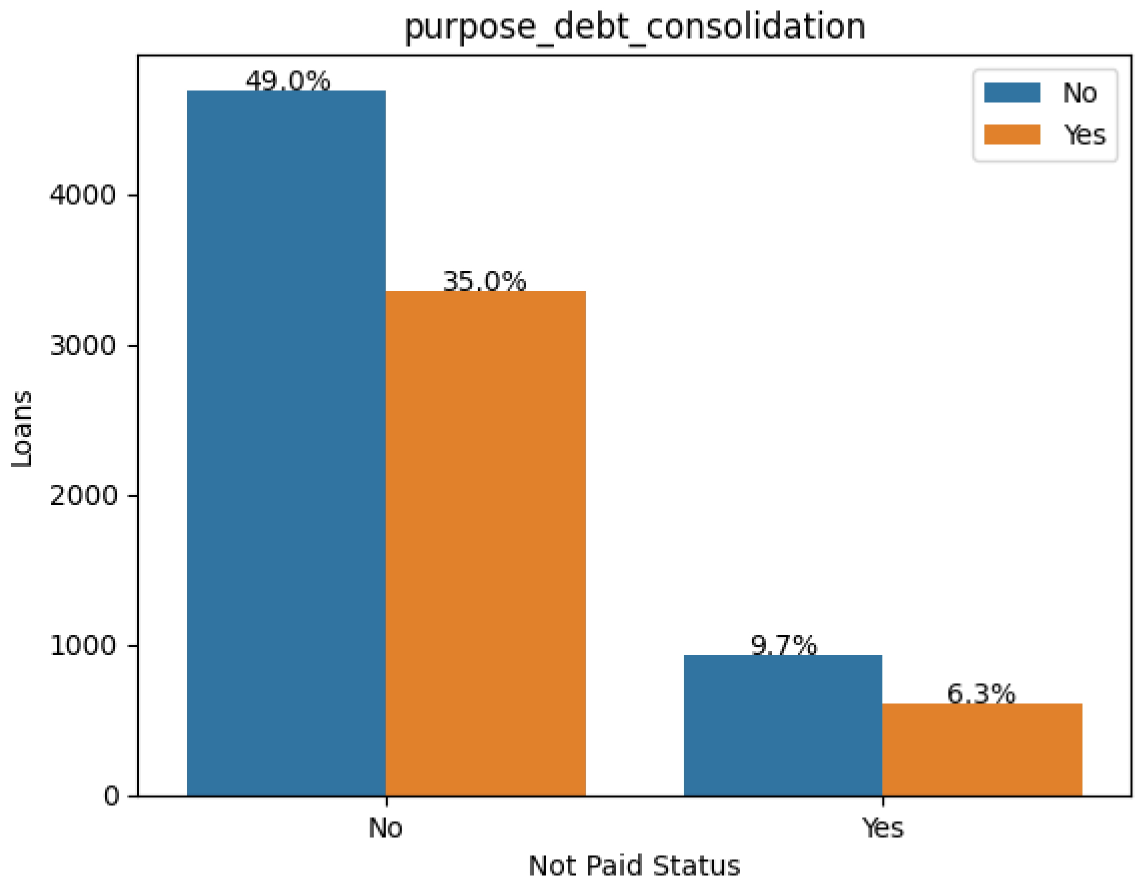
```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```



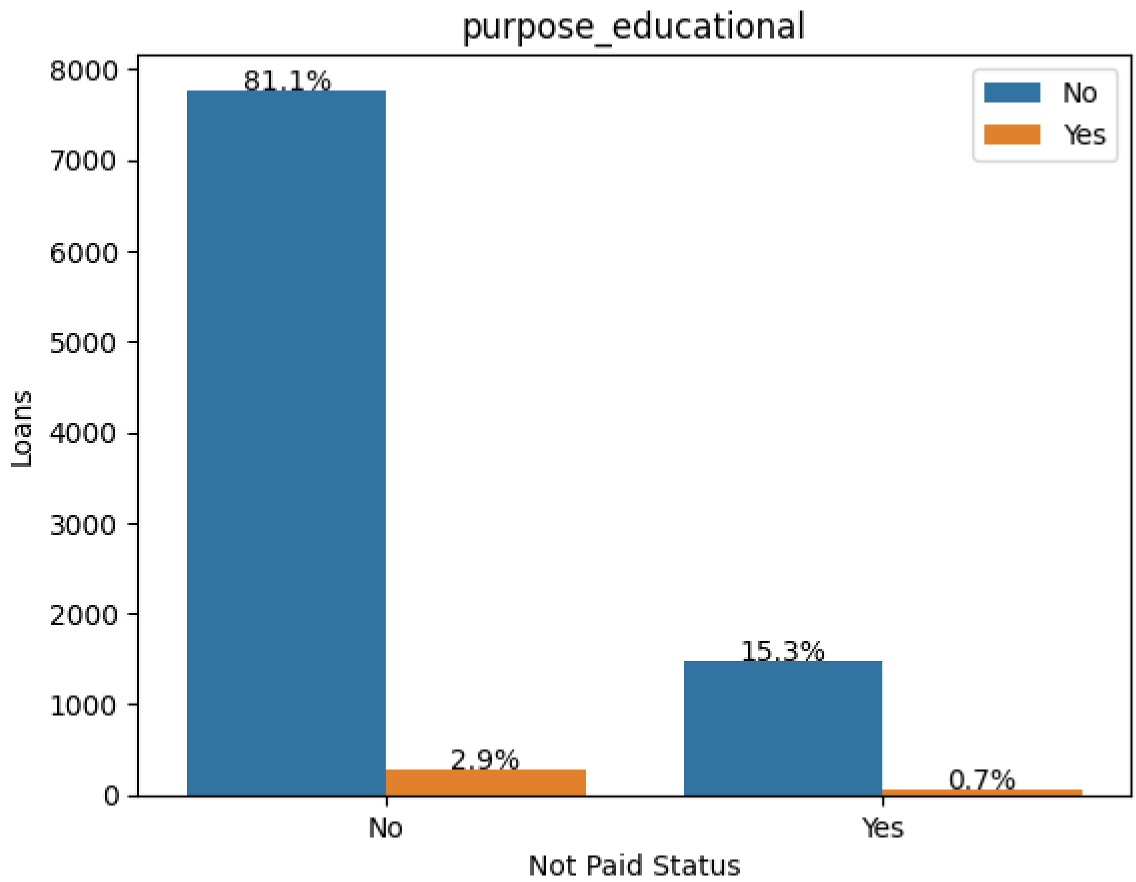

```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```



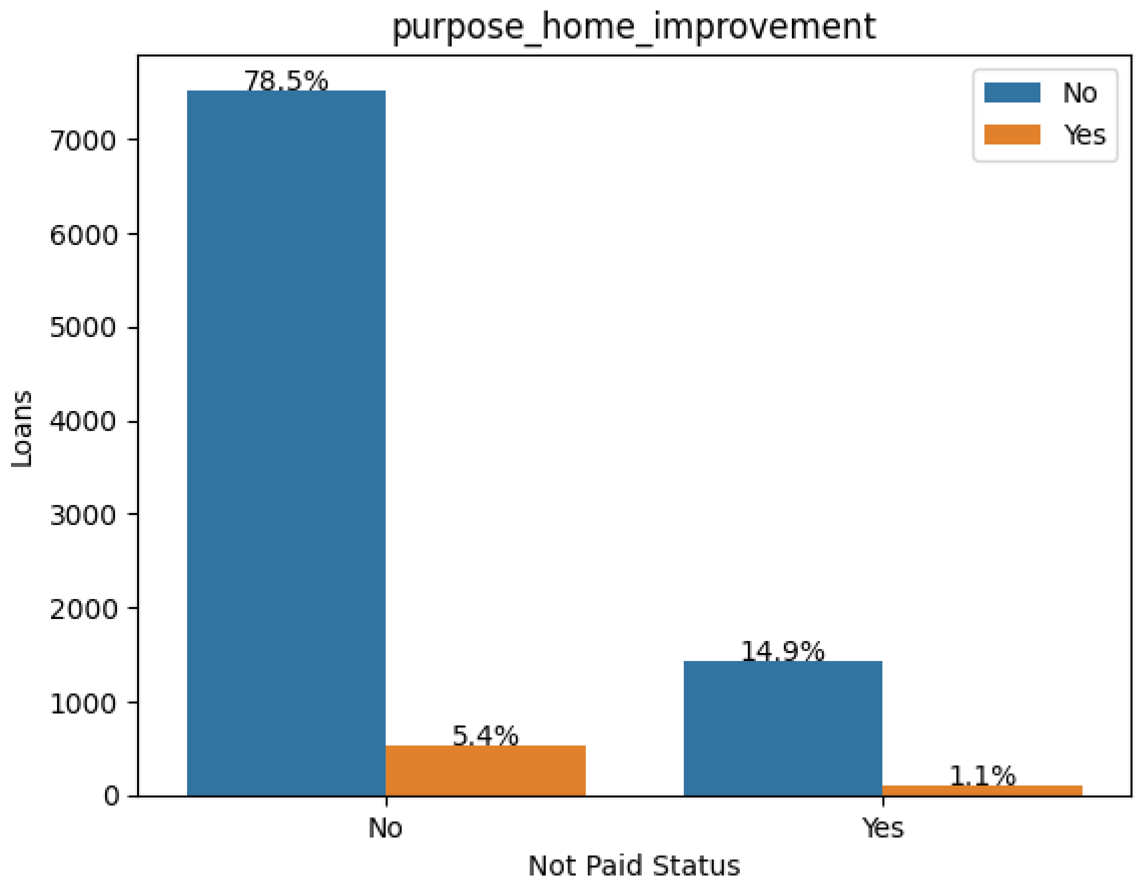

```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```



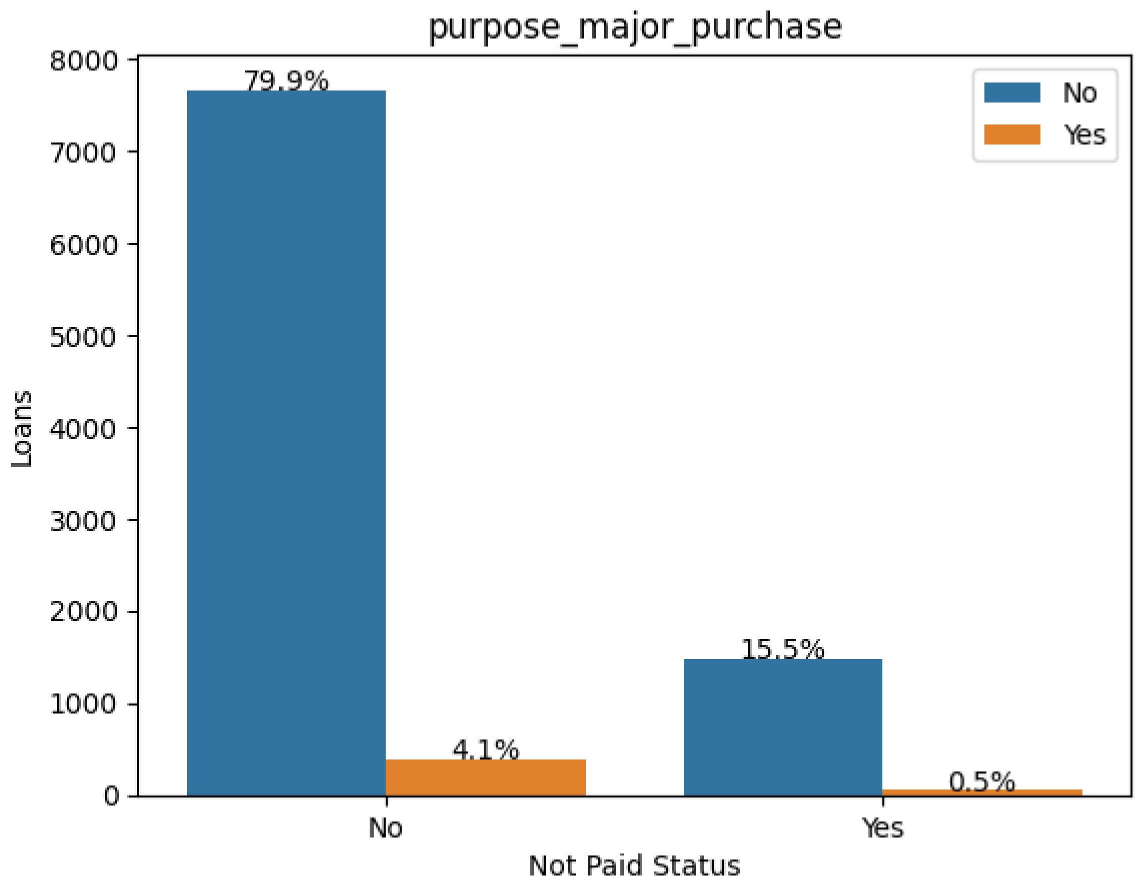

```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```



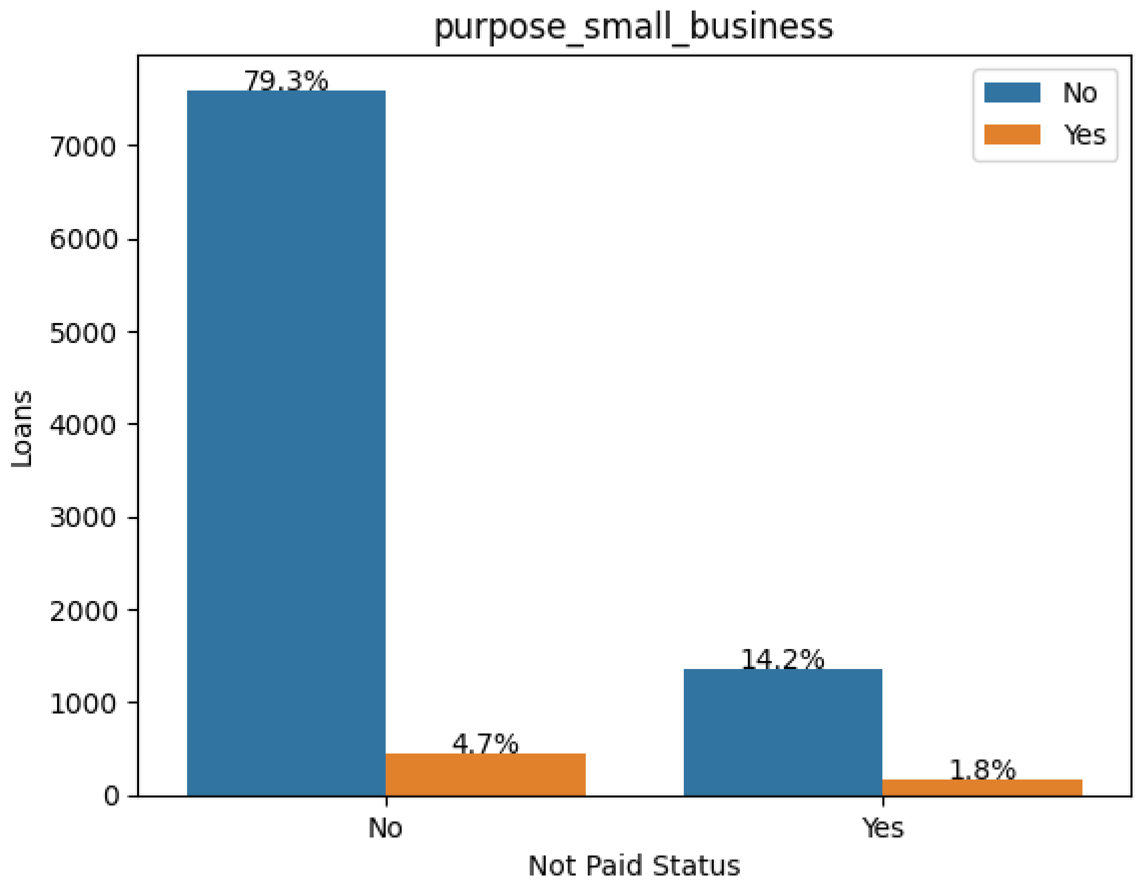

```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```




```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```




```
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:4: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Paid Status'] = np.where(df_dummy['not.fully.paid'] == 0, 'No', 'Yes')
/var/folders/r7/dtjny68152z02rjhb55rxv300000gn/T/ipykernel_94535/3147033967.py:5: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  bin_data['Condition'] = np.where(df_dummy[c] == 0, 'No', 'Yes')
```



- credit.policy

    ```
    - About 4/5 of all peope are approved for credit
    - About 2/3 of the unpaid loans are from people that fit the credit
    policy criteria
    - In general unapproved people are more likely to default. 25% of
    unapproved loans defaulted compared to just 13% of approved loans
    ```

- purpose_credit_card

    ```
    - 13.2% of loans are credit card loans
    - 11.4% of credit card loans default
    ```

- purpose_debt_consolidation

    ```
    - 41.3% of loans are debt consolidation
    - 15.2% of debt consolidation loans default
    ```

- purpose_educational

    ```
    - Only 3.6% of loans are educational
    - 19.4% of educational loans default
    ```

- purpose_home_improvement

    ```
    - 6.5% of loans are home improvement
    - 16.9% of home improvement loans default
    ```

- purpose_major_purchase

    ```
    - 4.6% of loans are for a major purchase
    - 10.9% of major purchase loans default
    ```

- purpose_small_business

```
- 6.5% of loans are for a small business
- 27.7% of small business loans default
```

- Key Takeaways

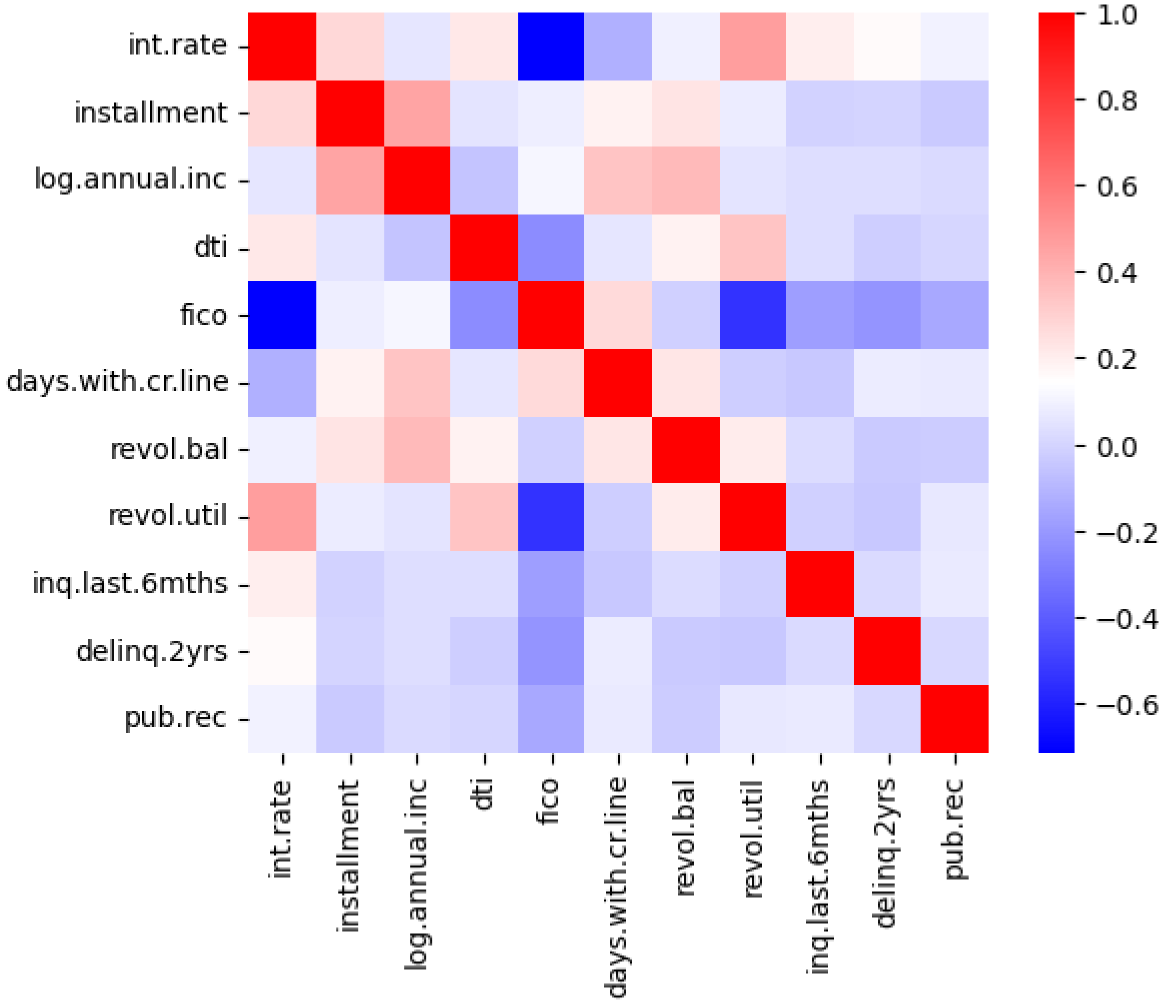
```
- Small Business loans are by far the riskiest, followed by
educational
- Debt Consolidation is 41.3% of all loans, and they default at a
slightly lower rate than average (16% default)
- Educational makes up such a small percentage, that small business is
likely not super impactful
- The only other loan type that is above average is Home Improvement
```

3. Additional Feature Engineering

In [13]:
```
# Find correlations between numerical features
features = df_dummy.drop(columns=['not.fully.paid'], axis=1)
num_features = features[numerical_cols]
labels = df_dummy[['not.fully.paid']]
```

In [14]:
```
num_corr = num_features.corr()
sns.heatmap(data=num_corr, square=True, cmap='bwr')
```

Out[14]: <AxesSubplot: >



In [15]:
```
corr_arr = num_corr.unstack()
corr_arr = corr_arr[corr_arr != 1]
corr_arr = corr_arr.drop_duplicates()
sorted_corr = corr_arr.sort_values(ascending=False)
opp_sorted_corr = corr_arr.sort_values(ascending=True)
```

```
print(f'Top Positive Correlations:\n\n{sorted_corr.head(10)}')
print(f'\n\nTop Negative Correlations:\n\n{opp_sorted_corr.head(10)}')
```

```
Top Positive Correlations:

int.rate           revol.util         0.464837
installment        log.annual.inc     0.448102
log.annual.inc     revol.bal          0.372140
dti                revol.util         0.337109
log.annual.inc     days.with.cr.line  0.336896
int.rate           installment        0.276140
fico               days.with.cr.line  0.263880
installment        revol.bal          0.233625
days.with.cr.line  revol.bal          0.229344
int.rate           dti                0.220006
dtype: float64


Top Negative Correlations:

int.rate           fico              -0.714821
fico               revol.util        -0.541289
dti                fico              -0.241191
fico               delinq.2yrs       -0.216340
                   inq.last.6mths    -0.185293
                   pub.rec           -0.147592
int.rate           days.with.cr.line -0.124022
log.annual.inc     dti               -0.054065
revol.util         delinq.2yrs       -0.042740
days.with.cr.line  inq.last.6mths    -0.041736
dtype: float64
```

Correlation takeaways:

- The positive correlations are all below 0.5. I think we shouldn't remove any features due to those correlations
- FICO score is negatively correlated with quite a few features and should be removed because of it
- I may come back to check on int.rate & revol.util after building the model

In [16]:
```
features = features.drop(columns=['fico'], axis=1)
```

4. Modeling

In [17]:
```
# Data Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, precision_recall_curve,confusion_matrix
import joblib

def data_split_standardise(x,y=None):
    if y is None:
        st=StandardScaler()
        st.fit(x)
        x_std=st.transform(x)
        joblib.dump(st,"model_objects/StandardScalar_trained.h5")
        return(x_std)
    else:
        x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0)
        st=StandardScaler()
        st.fit(x_train)
        x_train_std=st.transform(x_train)
        x_test_std=st.transform(x_test)
```

```
        joblib.dump(st,"model_objects/StandardScalar_trained.h5")
        return(x_train_std,x_test_std,y_train,y_test)
```

In [18]:
```
x_train, x_test, y_train, y_test = data_split_standardise(features,labels)
```

In [19]:
```
# Build Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.initializers import Constant
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy, TruePositives
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
from livelossplot import PlotLossesKerasTF
```

```
2023-03-12 10:43:27.803024: I tensorflow/core/platform/cpu_feature_guard.cc:193] This
TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to us
e the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler
flags.
```

In [20]:
```
# Modeling Constants
METRICS = [
    BinaryAccuracy(name='Binary_Accuracy'),
    Precision(name='Precision'),
    Recall(name='Recall'),
    TruePositives(name='True_Positives'),
    TrueNegatives(name='True_Negatives'),
    FalsePositives(name='False_Positives'),
    FalseNegatives(name='False_Negatives'),
    AUC(name='AUC'),
    AUC(name='Precision-Recall', curve='PR')
]
EPOCHS = 100
BATCH_SIZE = 512
```

```
2023-03-12 10:43:34.330481: I tensorflow/core/platform/cpu_feature_guard.cc:193] This
TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to us
e the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler
flags.
```

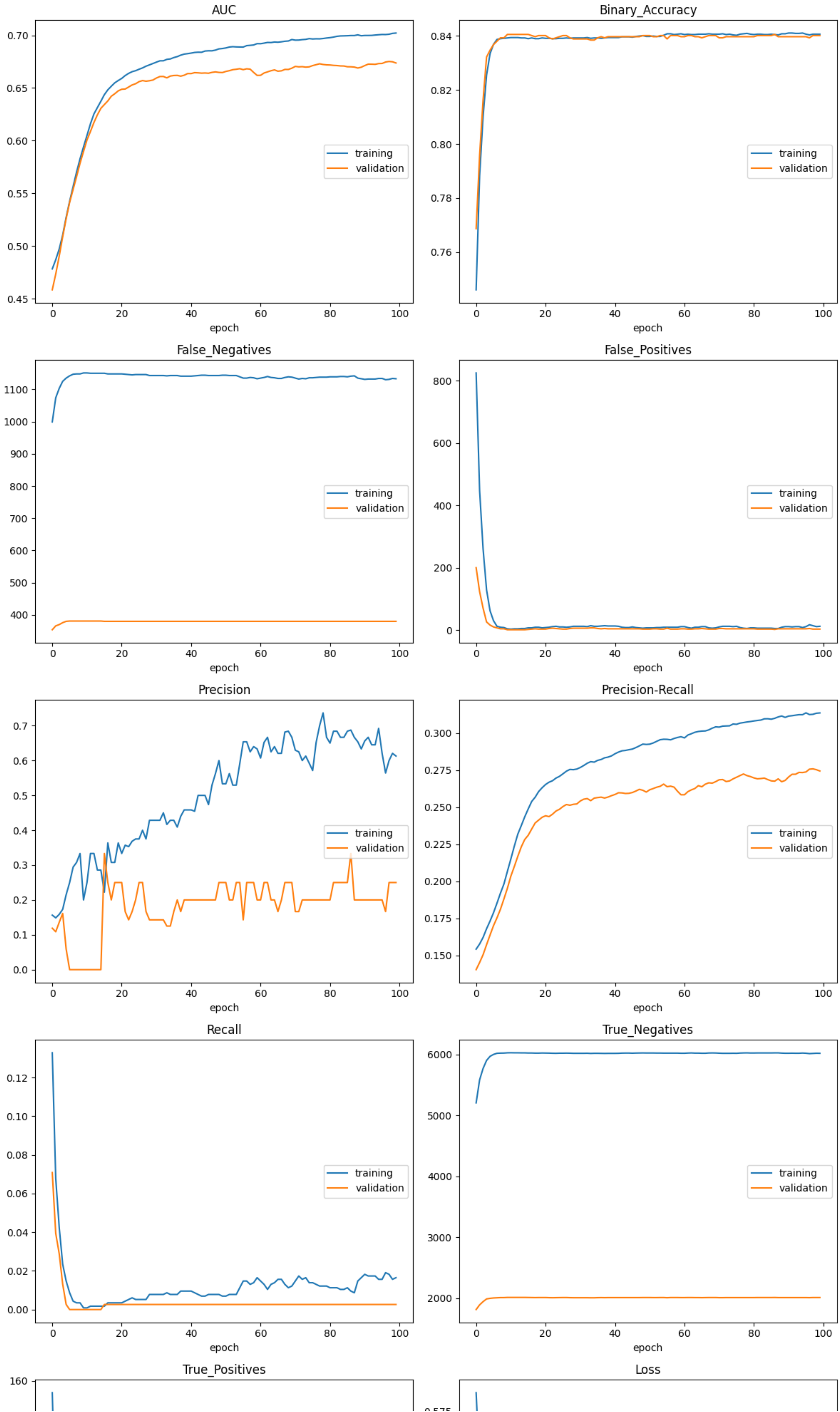
In [21]:
```
# Helpful plotting functions

# Confusion Matrix
def plot_cm(y_act, y_pred, p=0.5):
    cm = confusion_matrix(y_act, y_pred > p)
    plt.figure(figsize=(5,5))
    sns.heatmap(cm, annot=True)
    plt.title(f'Confusion Matrix P={p}')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')

# ROC Curve
def plot_roc(y_act, y_pred, name='ROC', **kwargs):
    false_positive, true_positive, na = roc_curve(y_act, y_pred)

    plt.plot(false_positive, true_positive, label=name, **kwargs)
    plt.xlabel('False Positives')
    plt.ylabel('True Positives')
    plt.grid(True)
    ax = plt.gca()
    ax.set_aspect('equal')

# Precision-Recall Curve
def plot_prc(y_act, y_pred, name='ROC', **kwargs):
```

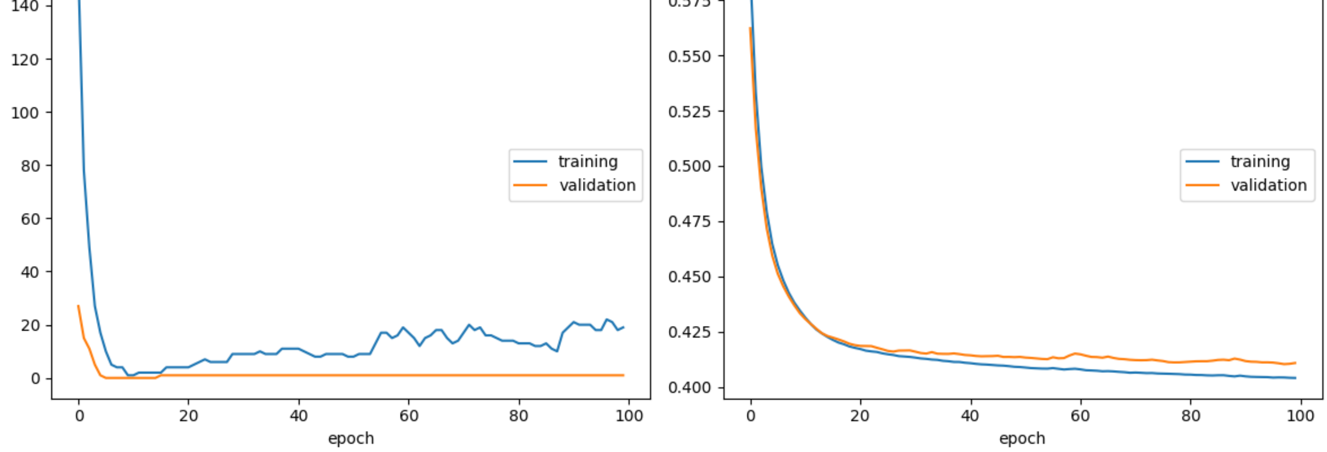
```python
    precision, recall, _ = precision_recall_curve(y_act, y_pred)

    plt.plot(precision, recall, label=name, **kwargs)
    plt.xlabel('Precision')
    plt.ylabel('Recall')
    plt.grid(True)
    ax = plt.gca()
    ax.set_aspect('equal')
```

In [22]:
```python
# First Attempt
def make_basic_model(metrics=METRICS, output_bias=None):
    if output_bias:
        output_bias = Constant(output_bias)

    model = Sequential()
    model.add(Input(shape=(None,x_train.shape[1]),name='Input_Layer'))
    model.add(Dense(12,activation='relu',name='Hidden_Layer_1'))
    model.add(Dense(8,activation='relu',name='Hidden_Layer_2'))
    model.add(Dense(1,activation='sigmoid',name='Output_Layer',bias_initializer=outpu

    model.compile(
        loss=BinaryCrossentropy(),
        optimizer=Adam(learning_rate=0.001),
        metrics=METRICS
    )

    return model
```

In [23]:
```python
model = make_basic_model(metrics=METRICS)
```

In [24]:
```python
model.fit(
    x_train,
    y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_data=(x_test,y_test),
    callbacks=[PlotLossesKerasTF()]
)
```


AUC
Binary_Accuracy
False_Negatives
False_Positives
Precision
Precision-Recall
Recall
True_Negatives
True_Positives
Loss
training
validation
epoch




```
AUC
        training                     (min:    0.478, max:    0.702, cur:    0.702)
        validation                   (min:    0.459, max:    0.675, cur:    0.674)
Binary_Accuracy
        training                     (min:    0.746, max:    0.841, cur:    0.841)
        validation                   (min:    0.769, max:    0.841, cur:    0.840)
False_Negatives
        training                     (min:  999.000, max: 1151.000, cur: 1133.000)
        validation                   (min:  354.000, max:  381.000, cur:  380.000)
False_Positives
        training                     (min:    3.000, max:  825.000, cur:   12.000)
        validation                   (min:    1.000, max:  200.000, cur:    3.000)
Precision
        training                     (min:    0.149, max:    0.737, cur:    0.613)
        validation                   (min:    0.000, max:    0.333, cur:    0.250)
Precision-Recall
        training                     (min:    0.154, max:    0.314, cur:    0.314)
        validation                   (min:    0.141, max:    0.276, cur:    0.274)
Recall
        training                     (min:    0.001, max:    0.133, cur:    0.016)
        validation                   (min:    0.000, max:    0.071, cur:    0.003)
True_Negatives
        training                     (min: 5206.000, max: 6028.000, cur: 6019.000)
        validation                   (min: 1814.000, max: 2013.000, cur: 2011.000)
True_Positives
        training                     (min:    1.000, max:  153.000, cur:   19.000)
        validation                   (min:    0.000, max:   27.000, cur:    1.000)
Loss
        training                     (min:    0.404, max:    0.588, cur:    0.404)
        validation                   (min:    0.410, max:    0.562, cur:    0.411)
15/15 [==============================] - 2s 126ms/step - loss: 0.4042 - Binary_Accura
cy: 0.8406 - Precision: 0.6129 - Recall: 0.0165 - True_Positives: 19.0000 - True_Nega
tives: 6019.0000 - False_Positives: 12.0000 - False_Negatives: 1133.0000 - AUC: 0.702
2 - Precision-Recall: 0.3136 - val_loss: 0.4110 - val_Binary_Accuracy: 0.8401 - val_P
recision: 0.2500 - val_Recall: 0.0026 - val_True_Positives: 1.0000 - val_True_Negativ
es: 2011.0000 - val_False_Positives: 3.0000 - val_False_Negatives: 380.0000 - val_AU
C: 0.6737 - val_Precision-Recall: 0.2744
```

Out[24]: <keras.callbacks.History at 0x13798f4f0>

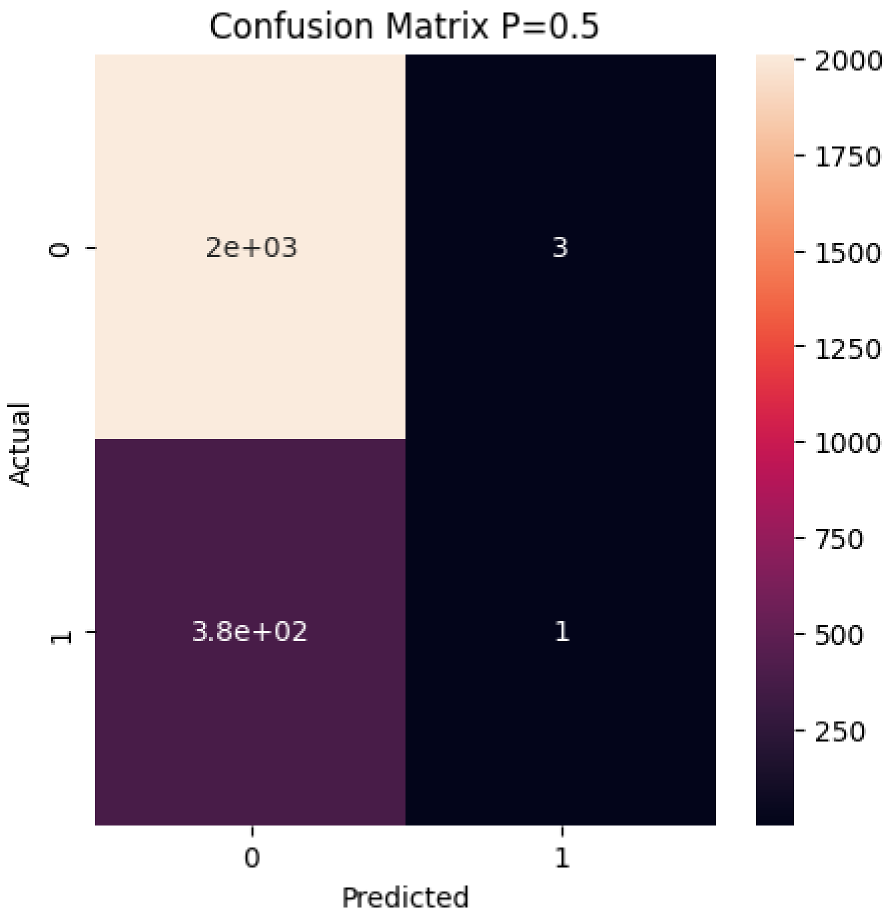
In [25]:

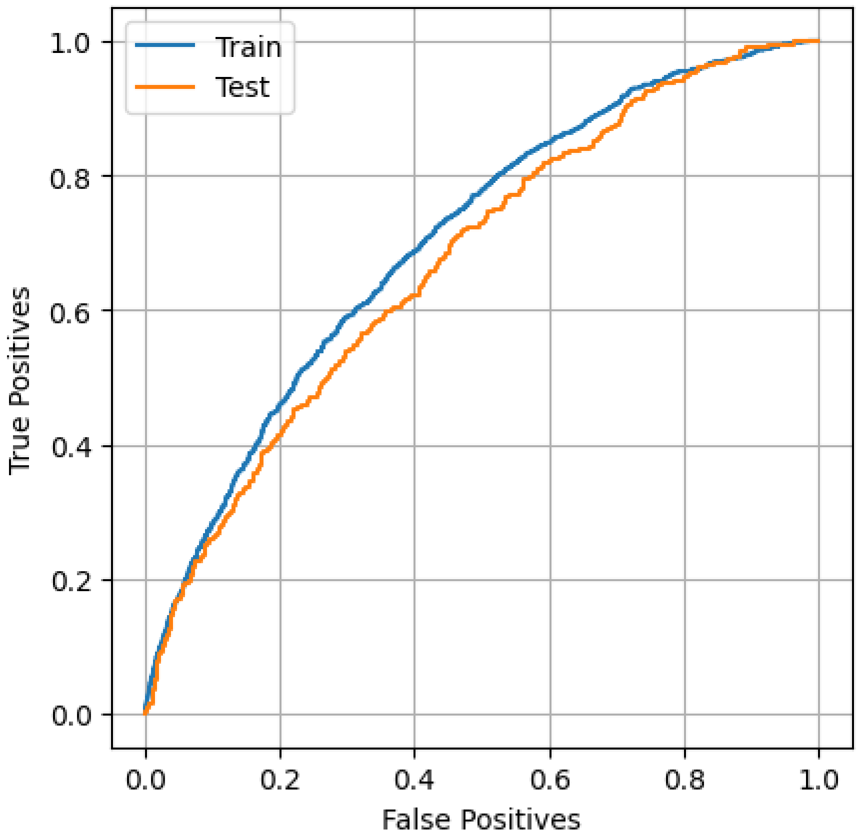
```python
# Helper function for fitting & evaluating models
def evaluate_and_plot(model, x_train, x_test, y_train, y_test, batch_size=BATCH_SIZE)
    # Get Predictions & Evaluate
    train_preds = model.predict(x_train, batch_size=batch_size)
    test_preds = model.predict(x_test, batch_size=batch_size)
    results = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=0)

    # Print Metric Scores
    print('\n\n')
    for metric, value in zip(model.metrics_names, results):
        print(f'{metric}:\t{value}')
```

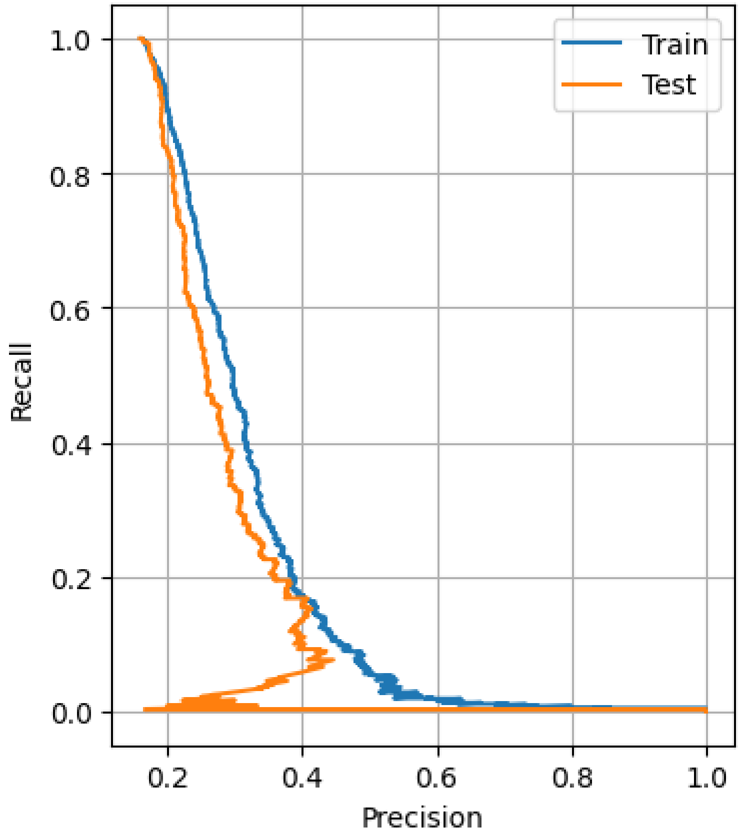
```python
    # Plot Confusion Matrix
    print('\n\n')
    plot_cm(y_test, test_preds)
    plt.show()

    # Plot ROC
    print('\n\n')
    plot_roc(y_train, train_preds, name='Train', color=colors[0])
    plot_roc(y_test, test_preds, name='Test', color=colors[1])
    plt.legend()
    plt.show()

    # Plot Precision-Recall
    print('\n\n')
    plot_prc(y_train, train_preds, name='Train', color=colors[0])
    plot_prc(y_test, test_preds, name='Test', color=colors[1])
    plt.legend()
    plt.show()
```

In [26]:
```python
evaluate_and_plot(
    model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

```
WARNING:tensorflow:Model was constructed with shape (None, None, 17) for input KerasT
ensor(type_spec=TensorSpec(shape=(None, None, 17), dtype=tf.float32, name='Input_Laye
r'), name='Input_Layer', description="created by layer 'Input_Layer'"), but it was ca
lled on an input with incompatible shape (None, 17).
15/15 [==============================] - 0s 885us/step
5/5 [==============================] - 0s 1ms/step



loss:	0.4109605848789215
Binary_Accuracy:	0.8400834798812866
Precision:	0.25
Recall: 0.002624671906232834
True_Positives: 1.0
True_Negatives: 2011.0
False_Positives:	3.0
False_Negatives:	380.0
AUC:	0.6737378835678101
Precision-Recall:	0.27441197633743286
```


Confusion Matrix P=0.5
2e+03
3
3.8e+02
1
Actual
Predicted
0
1
2000
1750
1500
1250
1000
750
500
250


Train
Test
True Positives
False Positives
0.0
0.2
0.4
0.6
0.8
1.0



Model Overfit

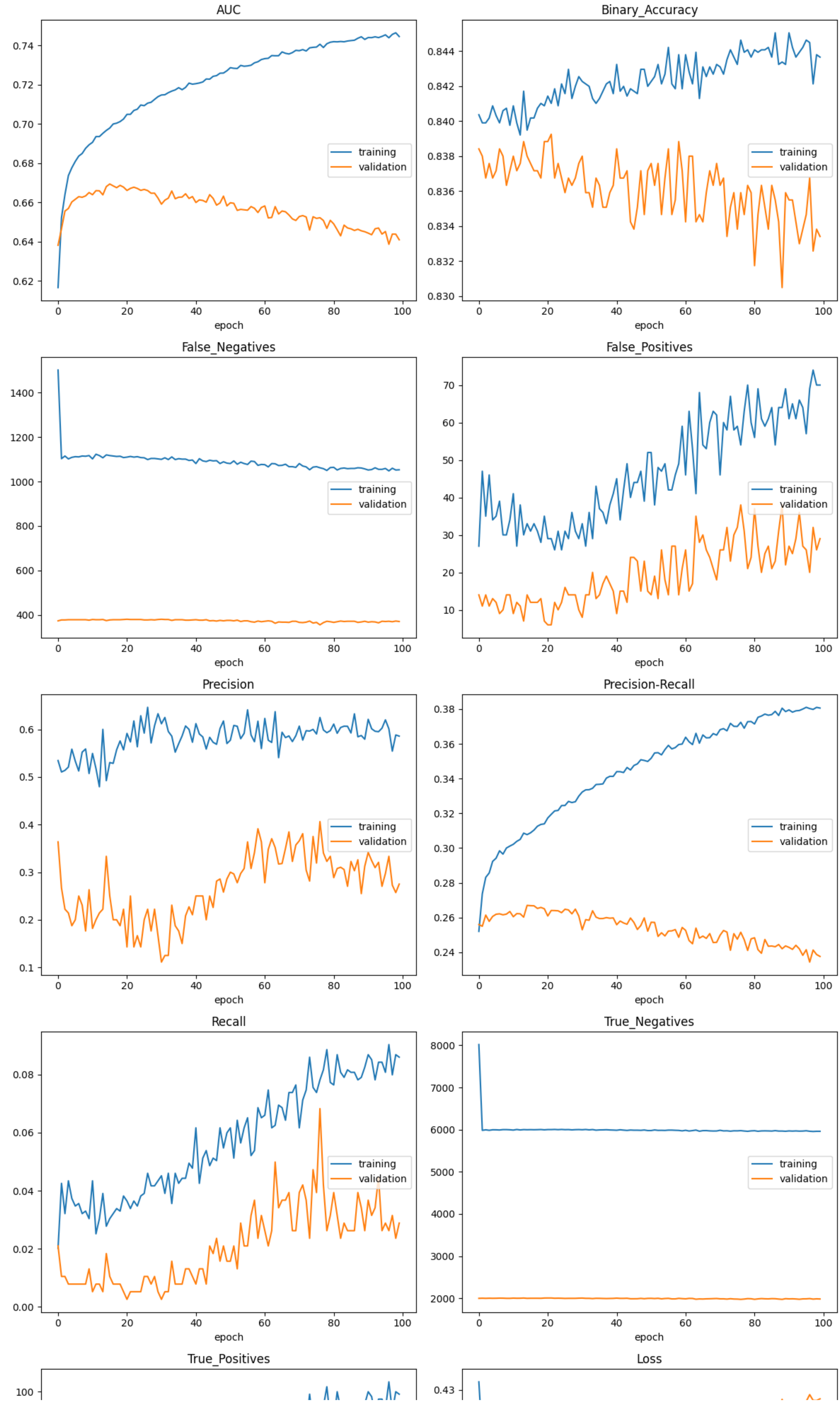
Accuracy was great because we didn't predict any defaults - Class Imbalance

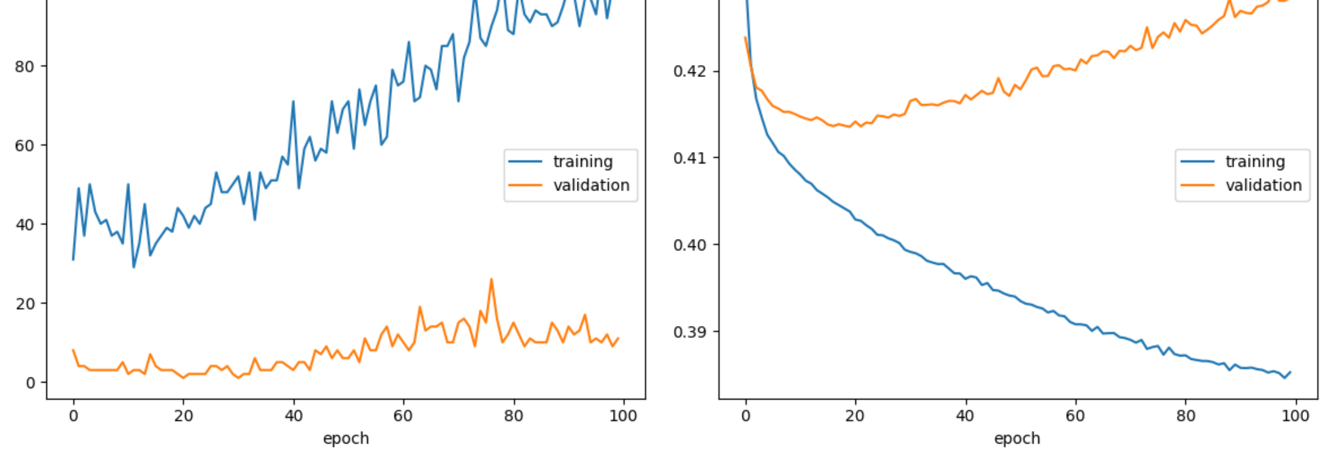
Let's try to add some initial bias

In [27]:
```
# set initial bias
neg, pos = np.bincount(labels['not.fully.paid'])
initial_bias = np.log([pos/neg])
initial_bias
```

Out[27]:
```
array([-1.65782418])
```

In [28]:
```
model = make_basic_model(metrics=METRICS, output_bias=initial_bias)

model.fit(
    x_train,
    y_train,
    epochs=100,
    validation_data=(x_test,y_test),
    callbacks=[PlotLossesKerasTF()]
)
```


AUC
training
validation
epoch
Binary_Accuracy
training
validation
epoch
False_Negatives
training
validation
epoch
False_Positives
training
validation
epoch
Precision
training
validation
epoch
Precision-Recall
training
validation
epoch
Recall
training
validation
epoch
True_Negatives
training
validation
epoch
True_Positives
Loss




```
AUC
	training         	 (min:    0.617, max:    0.746, cur:    0.745)
	validation       	 (min:    0.638, max:    0.669, cur:    0.641)
Binary_Accuracy
	training         	 (min:    0.839, max:    0.845, cur:    0.844)
	validation       	 (min:    0.830, max:    0.839, cur:    0.833)
False_Negatives
	training         	 (min: 1048.000, max: 1502.000, cur: 1053.000)
	validation       	 (min:  355.000, max:  380.000, cur:  370.000)
False_Positives
	training         	 (min:   26.000, max:   74.000, cur:   70.000)
	validation       	 (min:    6.000, max:   38.000, cur:   29.000)
Precision
	training         	 (min:    0.479, max:    0.646, cur:    0.586)
	validation       	 (min:    0.111, max:    0.406, cur:    0.275)
Precision-Recall
	training         	 (min:    0.252, max:    0.381, cur:    0.381)
	validation       	 (min:    0.234, max:    0.267, cur:    0.238)
Recall
	training         	 (min:    0.020, max:    0.090, cur:    0.086)
	validation       	 (min:    0.003, max:    0.068, cur:    0.029)
True_Negatives
	training         	 (min: 5957.000, max: 8018.000, cur: 5961.000)
	validation       	 (min: 1976.000, max: 2008.000, cur: 1985.000)
True_Positives
	training         	 (min:   29.000, max:  104.000, cur:   99.000)
	validation       	 (min:    1.000, max:   26.000, cur:   11.000)
Loss
	training         	 (min:    0.385, max:    0.431, cur:    0.385)
	validation       	 (min:    0.414, max:    0.429, cur:    0.428)
225/225 [==============================] - 2s 10ms/step - loss: 0.3853 - Binary_Accur
acy: 0.8437 - Precision: 0.5858 - Recall: 0.0859 - True_Positives: 99.0000 - True_Neg
atives: 5961.0000 - False_Positives: 70.0000 - False_Negatives: 1053.0000 - AUC: 0.74
45 - Precision-Recall: 0.3806 - val_loss: 0.4283 - val_Binary_Accuracy: 0.8334 - val_
Precision: 0.2750 - val_Recall: 0.0289 - val_True_Positives: 11.0000 - val_True_Negat
ives: 1985.0000 - val_False_Positives: 29.0000 - val_False_Negatives: 370.0000 - val_
AUC: 0.6409 - val_Precision-Recall: 0.2376
```

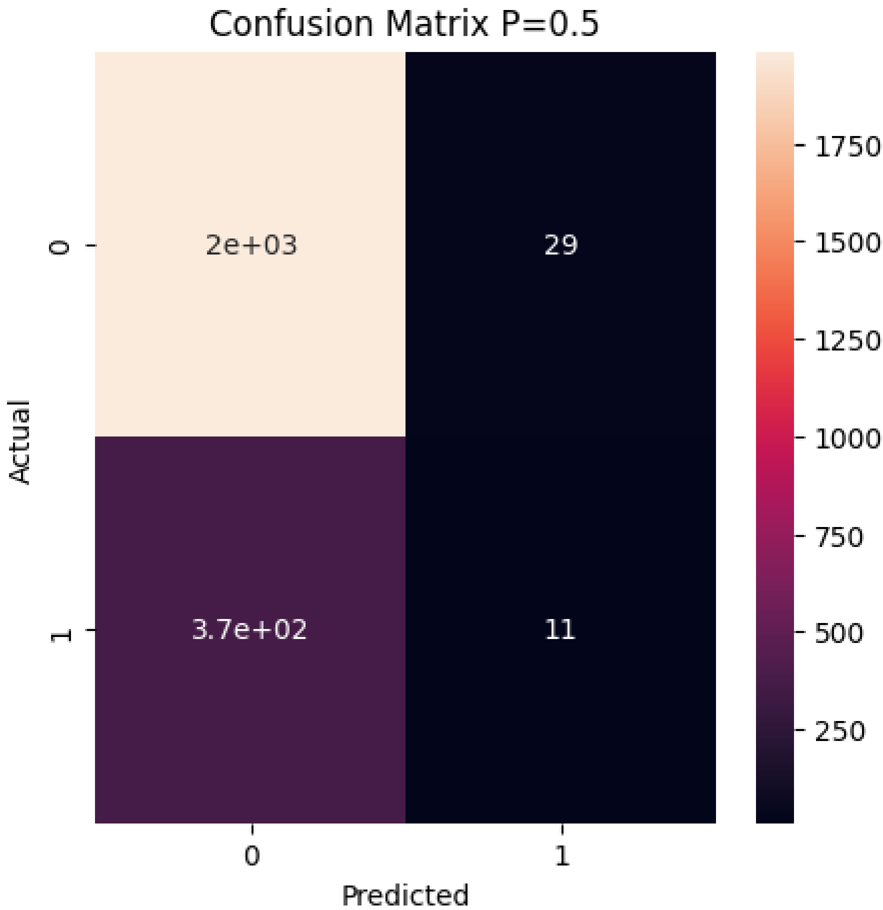
Out[28]: <keras.callbacks.History at 0x138593250>

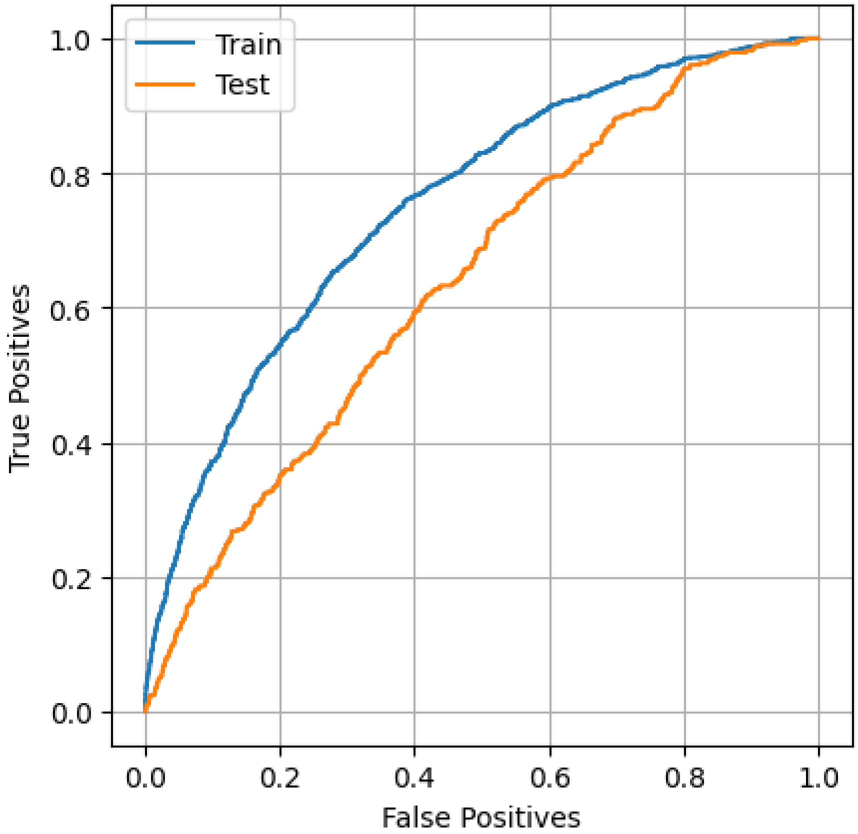
In [29]:

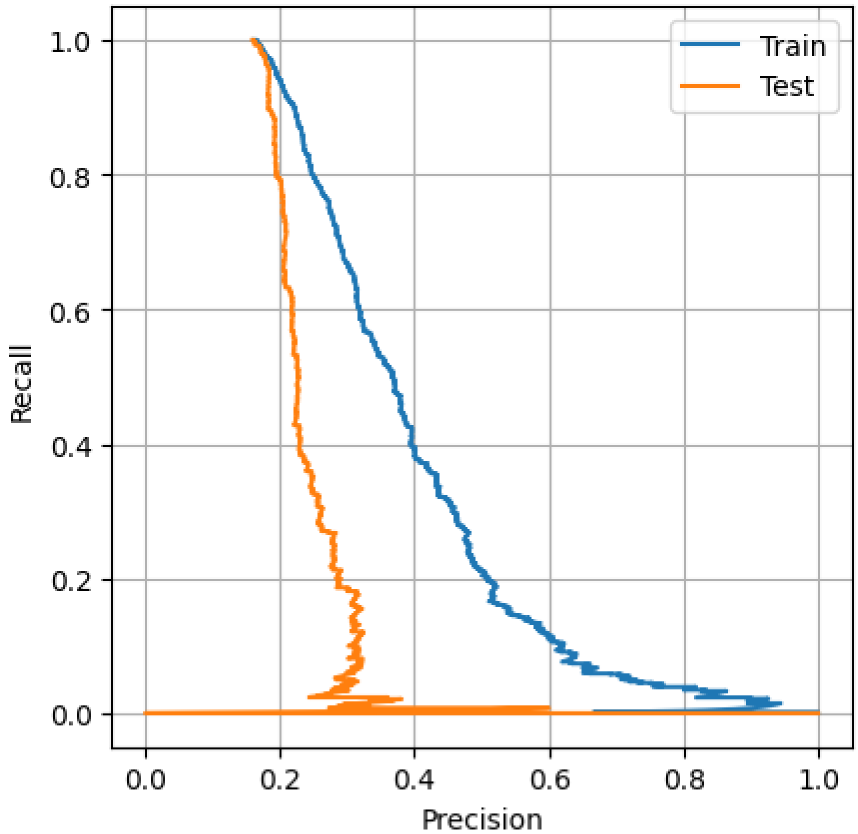
```
evaluate_and_plot(
    model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

```
WARNING:tensorflow:Model was constructed with shape (None, None, 17) for input KerasTensor(type_spec=TensorSpec(shape=(None, None, 17), dtype=tf.float32, name='Input_Layer'), name='Input_Layer', description="created by layer 'Input_Layer'"), but it was called on an input with incompatible shape (None, 17).
15/15 [==============================] - 0s 1ms/step
5/5 [==============================] - 0s 1ms/step
```

```
loss:   0.4283325970172882
Binary_Accuracy:        0.8334029316902161
Precision:      0.2750000059604645
Recall: 0.028871390968561172
True_Positives: 11.0
True_Negatives: 1985.0
False_Positives:        29.0
False_Negatives:        370.0
AUC:    0.6409314274787903
Precision-Recall:       0.2376335859298706
```







Model Still Overfit - We actually Got Some Non-Payers Though!

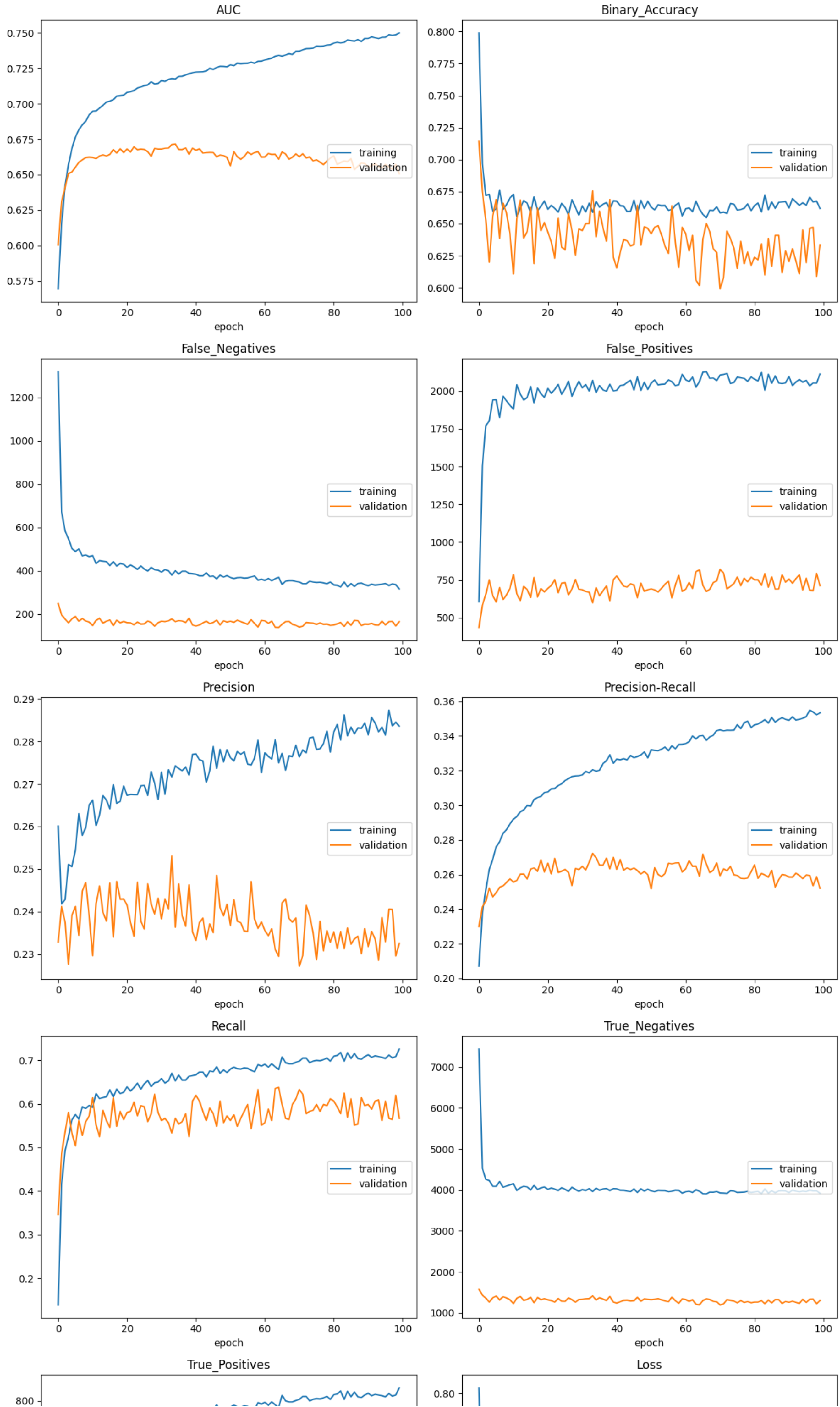
Let's add some class weights to try to make sure we catch more fraud

In [30]:
```
neg_weight = (1 / neg) * (labels.shape[0] / 2.0)
pos_weight = (1 / pos) * (labels.shape[0] / 2.0)
weights = {0: neg_weight, 1: pos_weight}
weights
```

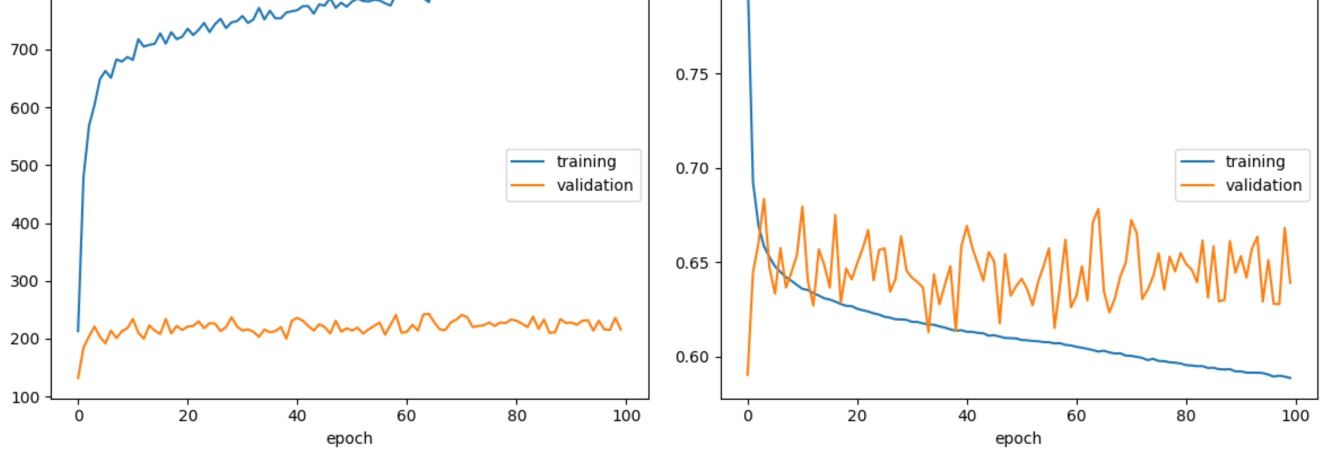
Out[30]:
```
{0: 0.5952765692977005, 1: 3.1239399869536855}
```

In [31]:

```
model = make_basic_model(metrics=METRICS, output_bias=initial_bias)

model.fit(
    x_train,
    y_train,
    epochs=100,
    validation_data=(x_test,y_test),
    callbacks=[PlotLossesKerasTF()],
    class_weight=weights
)
```



AUC
training
validation
epoch
Binary_Accuracy
training
validation
epoch
False_Negatives
training
validation
epoch
False_Positives
training
validation
epoch
Precision
training
validation
epoch
Precision-Recall
training
validation
epoch
Recall
training
validation
epoch
True_Negatives
training
validation
epoch
True_Positives
Loss




```
AUC
	training          	 (min:    0.569, max:    0.750, cur:    0.750)
	validation        	 (min:    0.600, max:    0.672, cur:    0.651)
Binary_Accuracy
	training          	 (min:    0.655, max:    0.799, cur:    0.662)
	validation        	 (min:    0.599, max:    0.714, cur:    0.633)
False_Negatives
	training          	 (min:  316.000, max: 1320.000, cur:  316.000)
	validation        	 (min:  138.000, max:  249.000, cur:  165.000)
False_Positives
	training          	 (min:  606.000, max: 2128.000, cur: 2112.000)
	validation        	 (min:  435.000, max:  820.000, cur:  713.000)
Precision
	training          	 (min:    0.242, max:    0.287, cur:    0.284)
	validation        	 (min:    0.227, max:    0.253, cur:    0.233)
Precision-Recall
	training          	 (min:    0.207, max:    0.355, cur:    0.353)
	validation        	 (min:    0.230, max:    0.272, cur:    0.252)
Recall
	training          	 (min:    0.139, max:    0.726, cur:    0.726)
	validation        	 (min:    0.346, max:    0.638, cur:    0.567)
True_Negatives
	training          	 (min: 3903.000, max: 7439.000, cur: 3919.000)
	validation        	 (min: 1194.000, max: 1579.000, cur: 1301.000)
True_Positives
	training          	 (min:  213.000, max:  836.000, cur:  836.000)
	validation        	 (min:  132.000, max:  243.000, cur:  216.000)
Loss
	training          	 (min:    0.589, max:    0.805, cur:    0.589)
	validation        	 (min:    0.590, max:    0.684, cur:    0.639)
225/225 [==============================] - 2s 10ms/step - loss: 0.5886 - Binary_Accur
acy: 0.6620 - Precision: 0.2836 - Recall: 0.7257 - True_Positives: 836.0000 - True_Ne
gatives: 3919.0000 - False_Positives: 2112.0000 - False_Negatives: 316.0000 - AUC: 0.
7500 - Precision-Recall: 0.3534 - val_loss: 0.6390 - val_Binary_Accuracy: 0.6334 - va
l_Precision: 0.2325 - val_Recall: 0.5669 - val_True_Positives: 216.0000 - val_True_Ne
gatives: 1301.0000 - val_False_Positives: 713.0000 - val_False_Negatives: 165.0000 -
val_AUC: 0.6507 - val_Precision-Recall: 0.2521
```

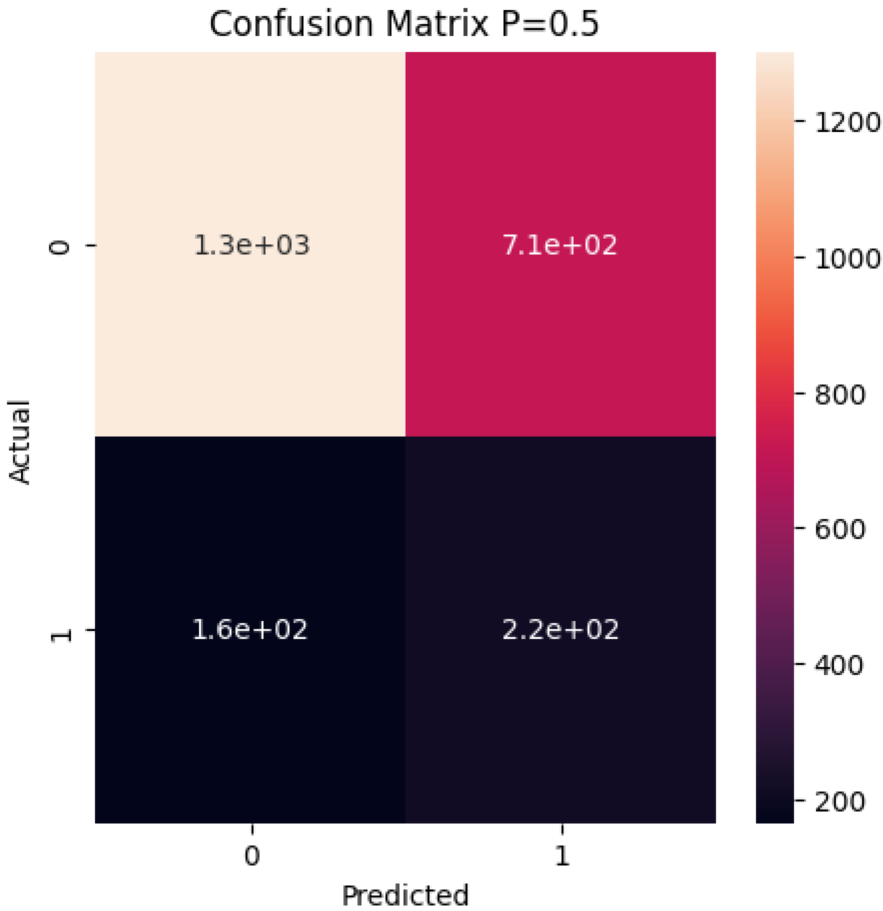
Out[31]: <keras.callbacks.History at 0x139277310>

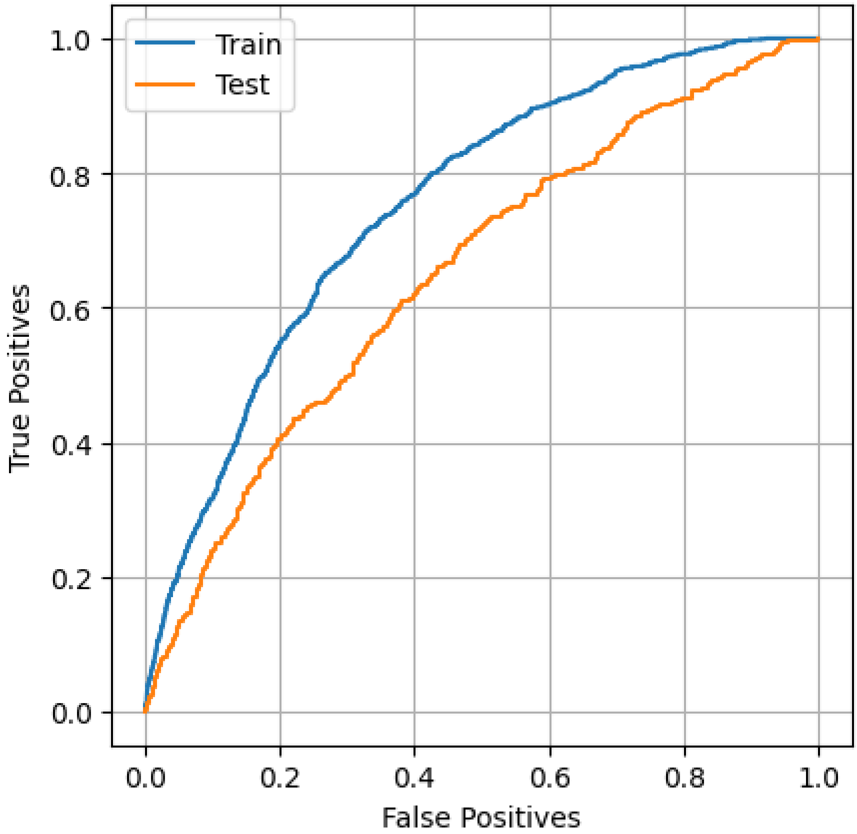
In [32]:

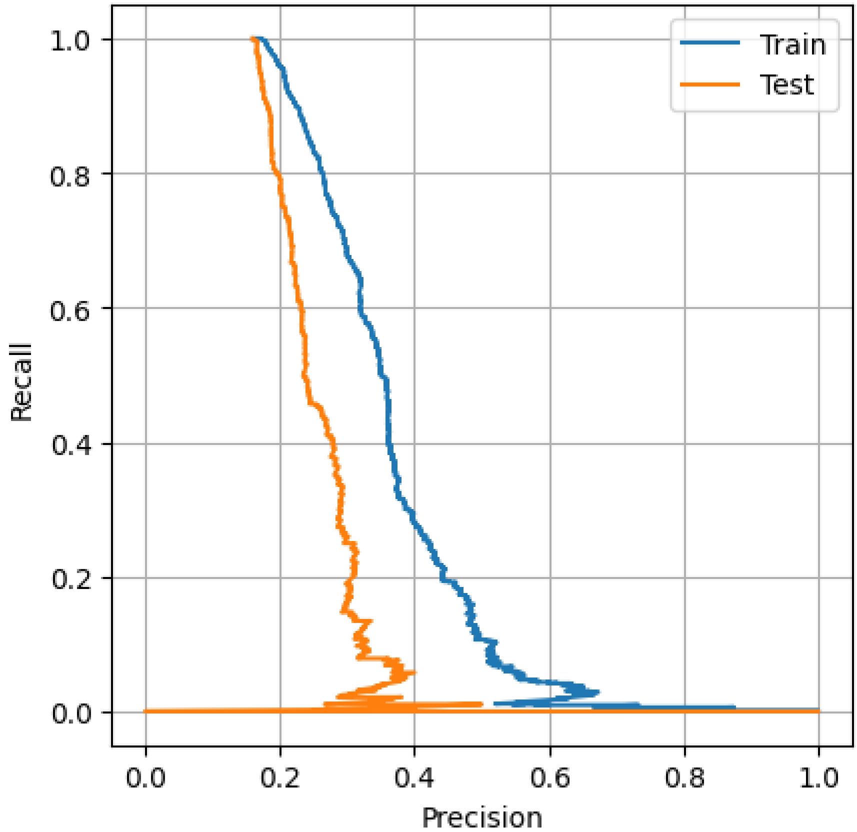
```
evaluate_and_plot(
    model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

```
WARNING:tensorflow:Model was constructed with shape (None, None, 17) for input KerasTensor(type_spec=TensorSpec(shape=(None, None, 17), dtype=tf.float32, name='Input_Layer'), name='Input_Layer', description="created by layer 'Input_Layer'"), but it was called on an input with incompatible shape (None, 17).
15/15 [==============================] - 0s 1ms/step
5/5 [==============================] - 0s 1ms/step
```

```
loss:	0.6390401124954224
Binary_Accuracy:	0.633402943611145
Precision:	0.2325080782175064
Recall: 0.5669291615486145
True_Positives: 216.0
True_Negatives: 1301.0
False_Positives:	713.0
False_Negatives:	165.0
AUC:	0.6507068872451782
Precision-Recall:	0.2520981729030609
```







Loss improved a bit, but still definitely overfit and in need of a bit more work

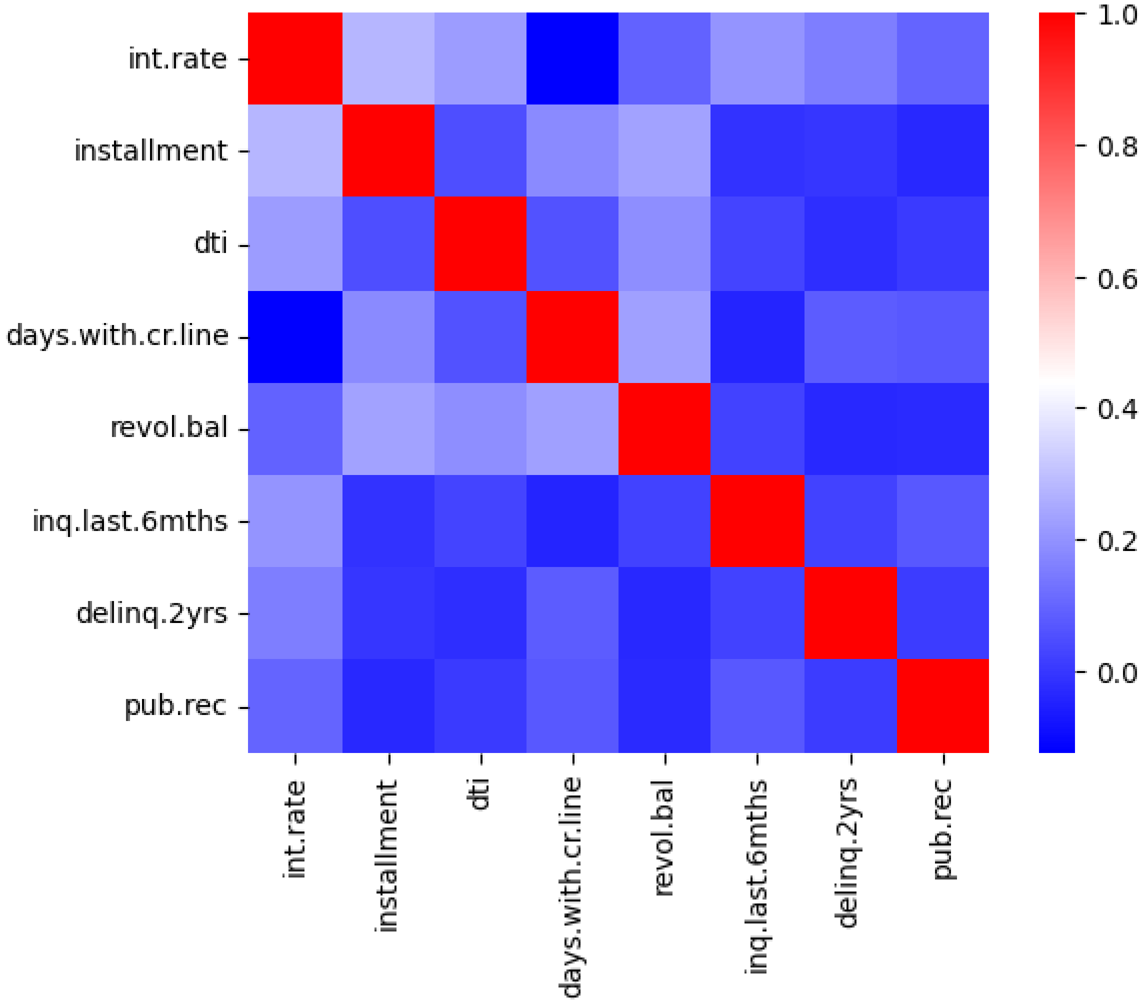
Additional Feature Engineering

Let's remove `log.annual.inc` and `revol.util` and look at correlations again

In [33]:
```
non_feature_cols = ['not.fully.paid','fico','log.annual.inc','revol.util']
features = df_dummy.drop(columns=non_feature_cols, axis=1)
num_features = features.drop(columns=binary_cols, axis=1)
```

In [34]:
```
num_corr = num_features.corr()
sns.heatmap(data=num_corr, square=True, cmap='bwr')
```

Out[34]: <AxesSubplot: >



In [35]:
```
corr_arr = num_corr.unstack()
corr_arr = corr_arr[corr_arr != 1]
corr_arr = corr_arr.drop_duplicates()
sorted_corr = corr_arr.sort_values(ascending=False)
opp_sorted_corr = corr_arr.sort_values(ascending=True)
print(f'Top Positive Correlations:\n\n{sorted_corr.head(10)}')
print(f'\n\nTop Negative Correlations:\n\n{opp_sorted_corr.head(10)}')
```

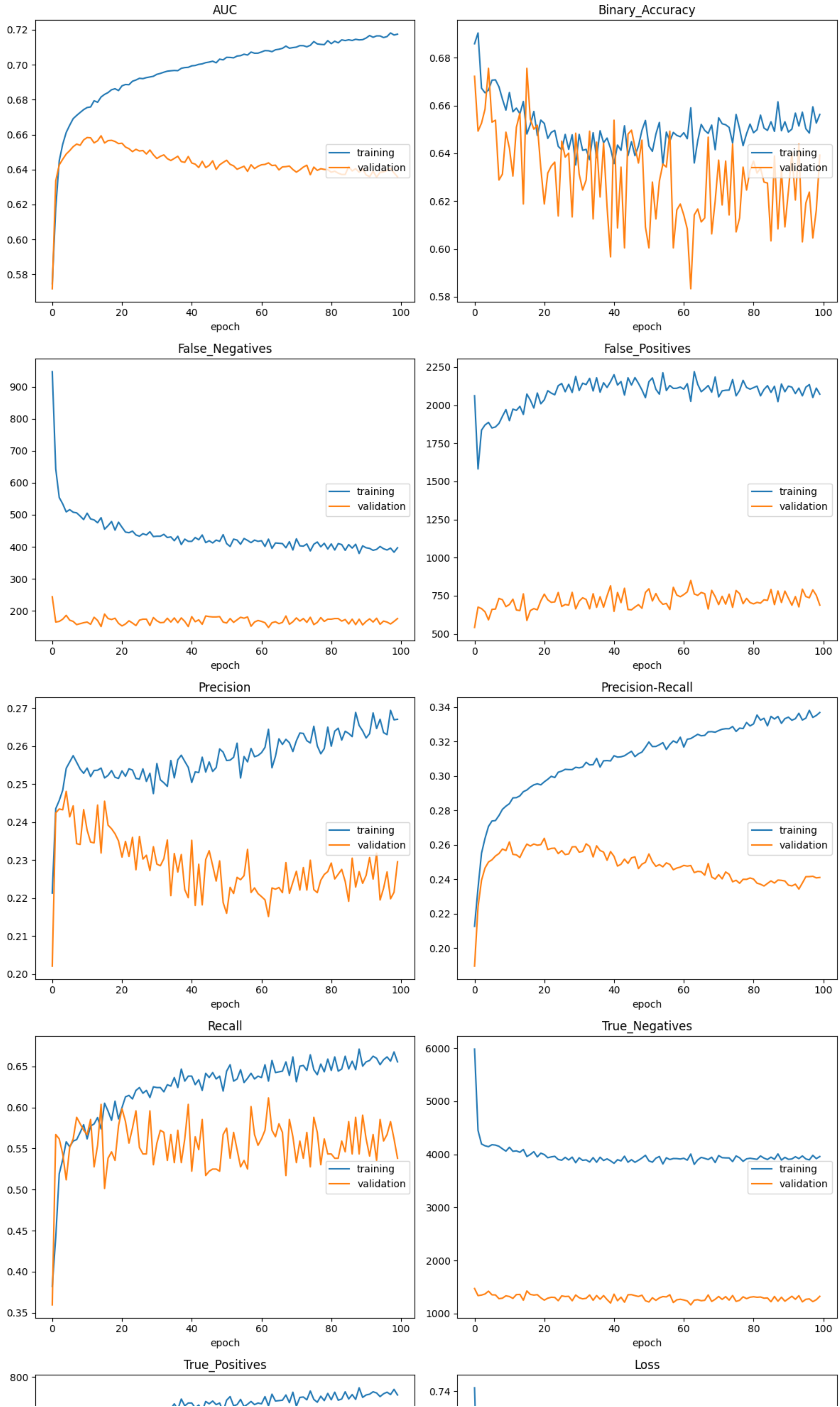
```
Top Positive Correlations:

int.rate           installment          0.276140
installment        revol.bal            0.233625
days.with.cr.line  revol.bal            0.229344
int.rate           dti                  0.220006
                   inq.last.6mths       0.202780
dti                revol.bal            0.188748
installment        days.with.cr.line    0.183297
int.rate           delinq.2yrs          0.156079
                   pub.rec              0.098162
                   revol.bal            0.092527
dtype: float64


Top Negative Correlations:

int.rate           days.with.cr.line   -0.124022
days.with.cr.line  inq.last.6mths      -0.041736
revol.bal          delinq.2yrs         -0.033243
installment        pub.rec             -0.032760
revol.bal          pub.rec             -0.031010
dti                delinq.2yrs         -0.021792
installment        inq.last.6mths      -0.010419
                   delinq.2yrs         -0.004368
dti                pub.rec              0.006209
delinq.2yrs        pub.rec              0.009184
dtype: float64
```

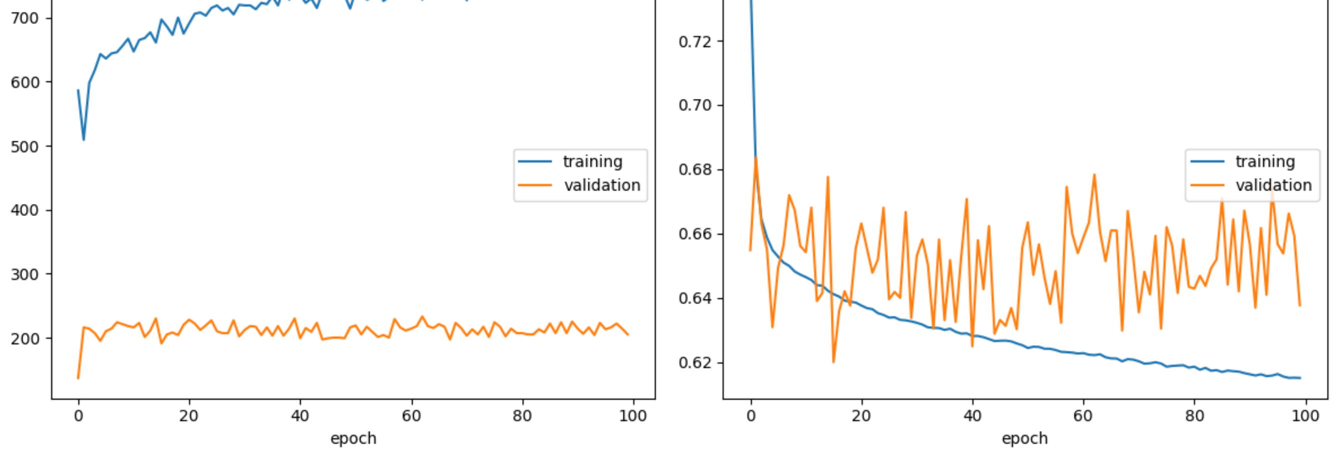
In [36]:

```
# Let's model with the same structure as the last run we did
x_train, x_test, y_train, y_test = data_split_standardise(features,labels)
```

In [37]:

```
model = make_basic_model(metrics=METRICS, output_bias=initial_bias)

model.fit(
    x_train,
    y_train,
    epochs=100,
    validation_data=(x_test,y_test),
    callbacks=[PlotLossesKerasTF()],
    class_weight=weights
)
```


AUC
Binary_Accuracy
False_Negatives
False_Positives
Precision
Precision-Recall
Recall
True_Negatives
True_Positives
Loss
epoch
training
validation



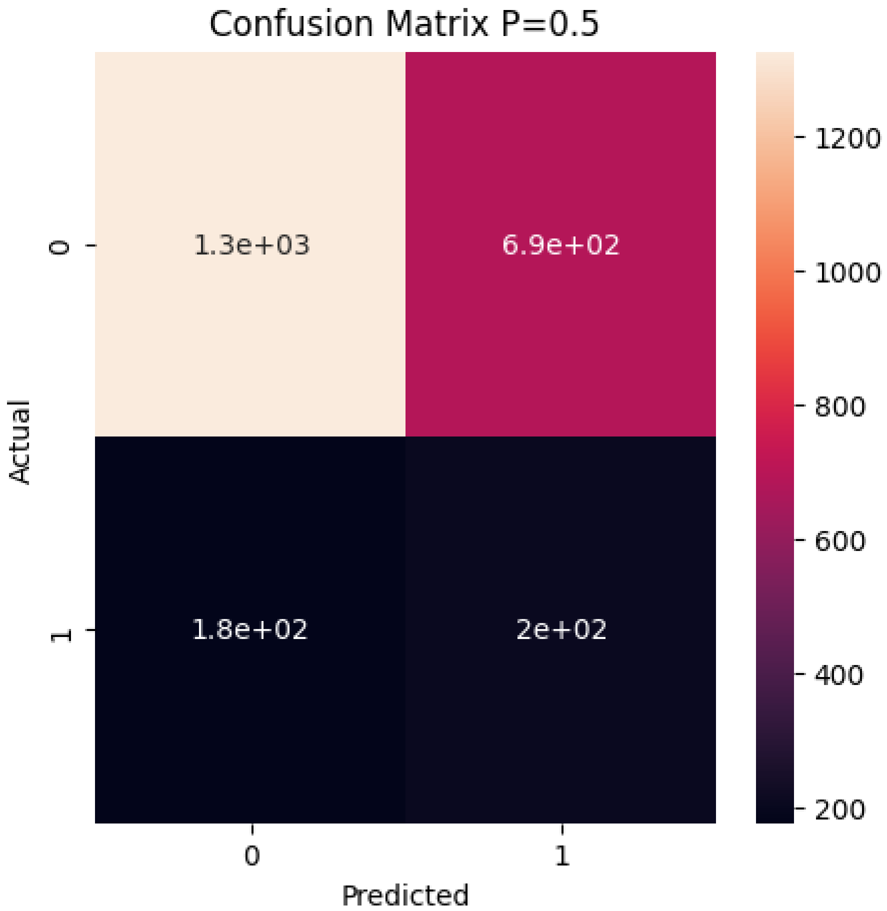

```
AUC
	training         	 (min:    0.575, max:    0.718, cur:    0.718)
	validation       	 (min:    0.572, max:    0.659, cur:    0.636)
Binary_Accuracy
	training         	 (min:    0.635, max:    0.690, cur:    0.656)
	validation       	 (min:    0.583, max:    0.676, cur:    0.639)
False_Negatives
	training         	 (min:  379.000, max:  947.000, cur:  397.000)
	validation       	 (min:  148.000, max:  244.000, cur:  176.000)
False_Positives
	training         	 (min: 1581.000, max: 2220.000, cur: 2072.000)
	validation       	 (min:  541.000, max:  850.000, cur:  688.000)
Precision
	training         	 (min:    0.221, max:    0.269, cur:    0.267)
	validation       	 (min:    0.202, max:    0.248, cur:    0.230)
Precision-Recall
	training         	 (min:    0.213, max:    0.338, cur:    0.337)
	validation       	 (min:    0.190, max:    0.264, cur:    0.241)
Recall
	training         	 (min:    0.382, max:    0.671, cur:    0.655)
	validation       	 (min:    0.360, max:    0.612, cur:    0.538)
True_Negatives
	training         	 (min: 3811.000, max: 5983.000, cur: 3959.000)
	validation       	 (min: 1164.000, max: 1473.000, cur: 1326.000)
True_Positives
	training         	 (min:  509.000, max:  773.000, cur:  755.000)
	validation       	 (min:  137.000, max:  233.000, cur:  205.000)
Loss
	training         	 (min:    0.615, max:    0.742, cur:    0.615)
	validation       	 (min:    0.620, max:    0.684, cur:    0.638)
225/225 [==============================] - 2s 10ms/step - loss: 0.6151 - Binary_Accuracy: 0.6563 - Precision: 0.2671 - Recall: 0.6554 - True_Positives: 755.0000 - True_Negatives: 3959.0000 - False_Positives: 2072.0000 - False_Negatives: 397.0000 - AUC: 0.7175 - Precision-Recall: 0.3368 - val_loss: 0.6377 - val_Binary_Accuracy: 0.6392 - val_Precision: 0.2296 - val_Recall: 0.5381 - val_True_Positives: 205.0000 - val_True_Negatives: 1326.0000 - val_False_Positives: 688.0000 - val_False_Negatives: 176.0000 - val_AUC: 0.6358 - val_Precision-Recall: 0.2411
```

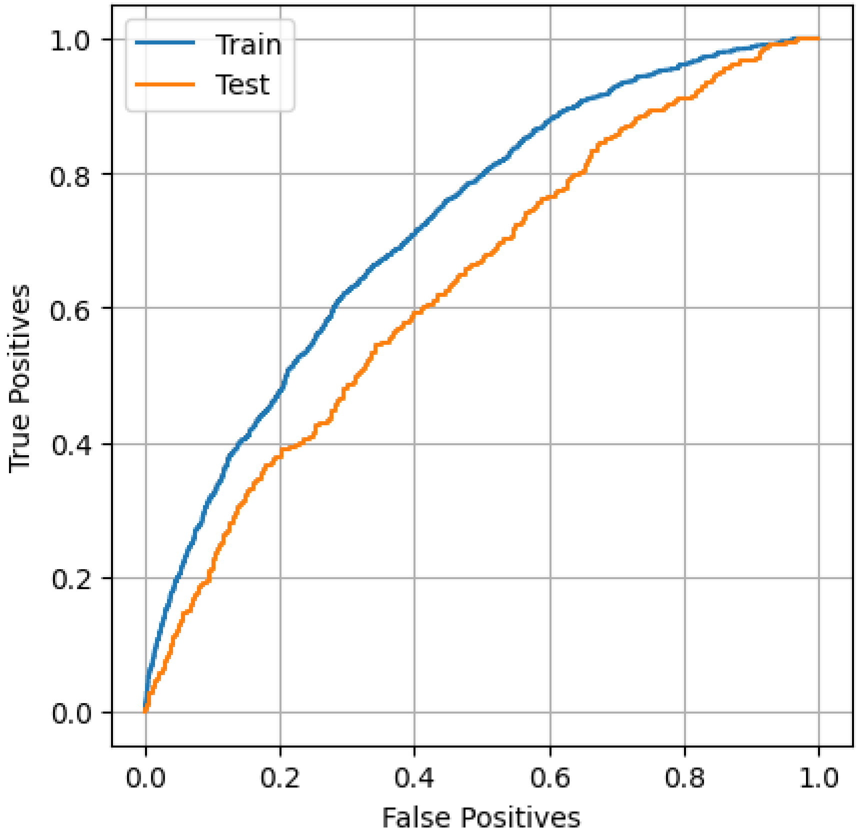
Out[37]: <keras.callbacks.History at 0x13b57feb0>

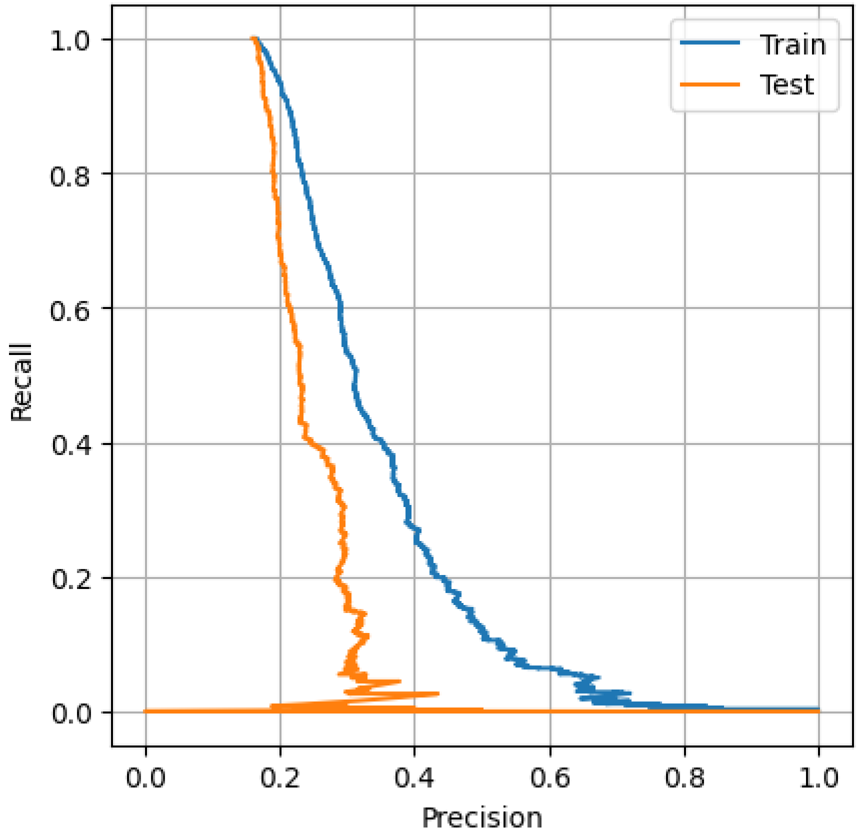
In [38]:
```
evaluate_and_plot(
    model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

```
WARNING:tensorflow:Model was constructed with shape (None, None, 15) for input KerasT
ensor(type_spec=TensorSpec(shape=(None, None, 15), dtype=tf.float32, name='Input_Laye
r'), name='Input_Layer', description="created by layer 'Input_Layer'"), but it was ca
lled on an input with incompatible shape (None, 15).
15/15 [==============================] - 0s 872us/step
5/5 [==============================] - 0s 1ms/step
```

```
loss:   0.6377151012420654
Binary_Accuracy:        0.6392484307289124
Precision:      0.22956326603889465
Recall: 0.5380577445030212
True_Positives: 205.0
True_Negatives: 1326.0
False_Positives:        688.0
False_Negatives:        176.0
AUC:    0.6357935667037964
Precision-Recall:       0.24106895923614502
```







Still not quite what we'd like... Seems like we're now predicting too many fraud cases.

Oversampling

Try oversampling on the positive class to make sure we're identifying potential loans that won't be repaid

In [58]:
```
# Set up split datasets between pos and neg observations to sample at different rates
pos_df = df_dummy[df_dummy['not.fully.paid'] == 1].reset_index()
neg_df = df_dummy[df_dummy['not.fully.paid'] == 0].reset_index()
```

```
# pos_features = pos_df.drop(columns=non_feature_cols, axis=1)
# neg_features = neg_df.drop(columns=non_feature_cols, axis=1)

# pos_labels = pos_df['not.fully.paid']
# neg_labels = neg_df['not.fully.paid']
```

In [61]:
```
pos_df.index
```

Out[61]:
```
RangeIndex(start=0, stop=1533, step=1)
```

In [63]:
```
# Randomly Sample the same number of positive observations as we have in the negative
resampled_pos_df = pos_df.sample(
    n=neg_df.shape[0],
    replace=True,
    random_state=0
)
resampled_pos_df.shape
```

Out[63]:
```
(8045, 20)
```

In [68]:
```
# re-combine the newly resampled dataset
resampled_df = pd.concat([resampled_pos_df, neg_df])

resampled_features = resampled_df.drop(columns=non_feature_cols, axis=1)
resampled_labels = resampled_df['not.fully.paid']

resampled_df.shape
```

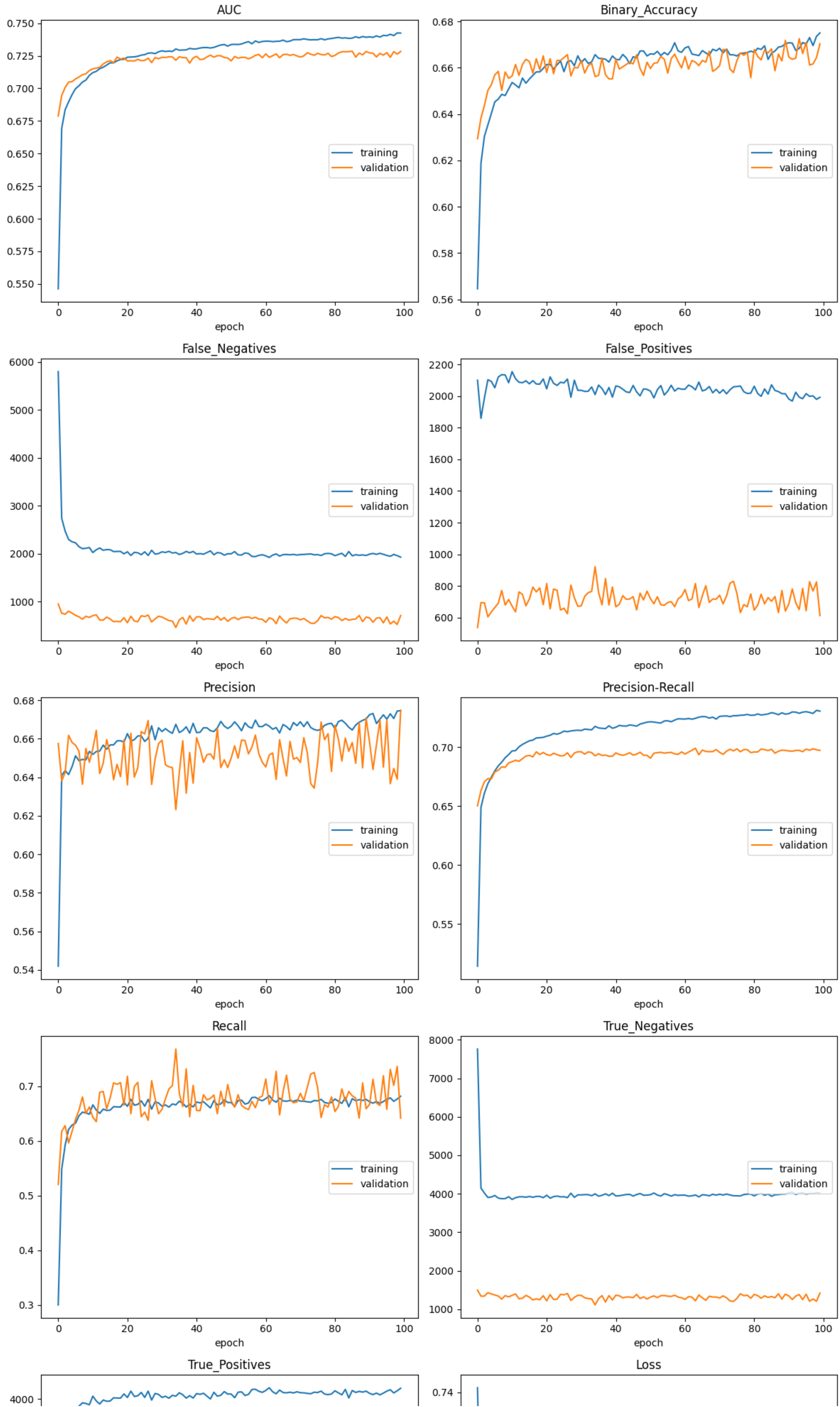
Out[68]:
```
(16090, 20)
```

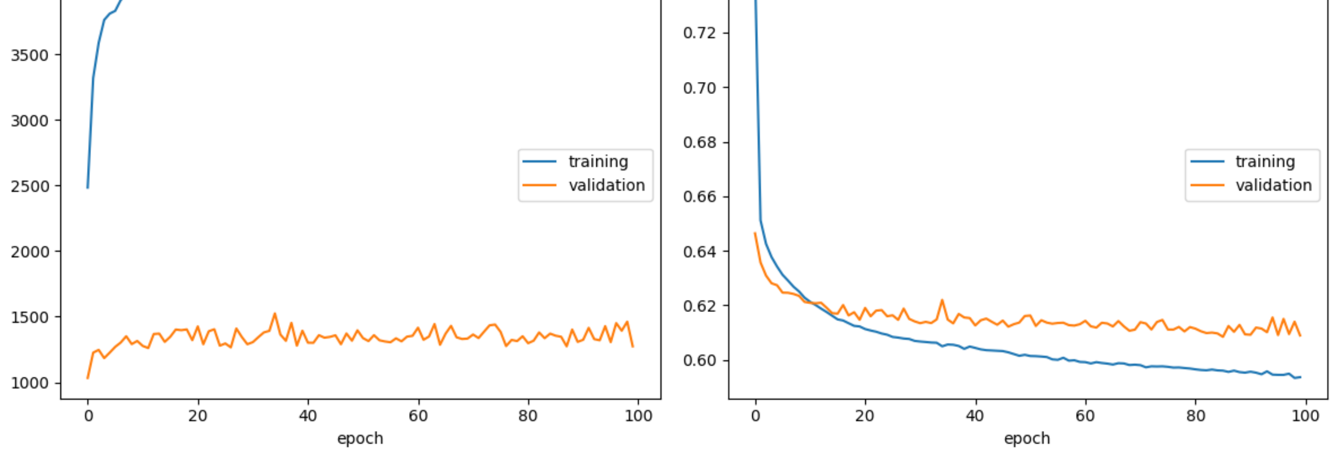
In [69]:
```
# confirm re-balanced classes
resampled_df['not.fully.paid'].value_counts()
```

Out[69]:
```
1    8045
0    8045
Name: not.fully.paid, dtype: int64
```

In [70]:
```
# split the data into train & test
x_train, x_test, y_train, y_test = data_split_standardise(resampled_features,resample
```

In [72]:
```
# re-build the model
# make sure to not add class weights or initial bias because we've rebalanced already
model = make_basic_model(metrics=METRICS, output_bias=initial_bias)

model.fit(
    x_train,
    y_train,
    epochs=100,
    validation_data=(x_test,y_test),
    callbacks=[PlotLossesKerasTF()]
)
```


AUC
epoch
training
validation
Binary_Accuracy
epoch
training
validation
False_Negatives
epoch
training
validation
False_Positives
epoch
training
validation
Precision
epoch
training
validation
Precision-Recall
epoch
training
validation
Recall
epoch
training
validation
True_Negatives
epoch
training
validation
True_Positives
Loss



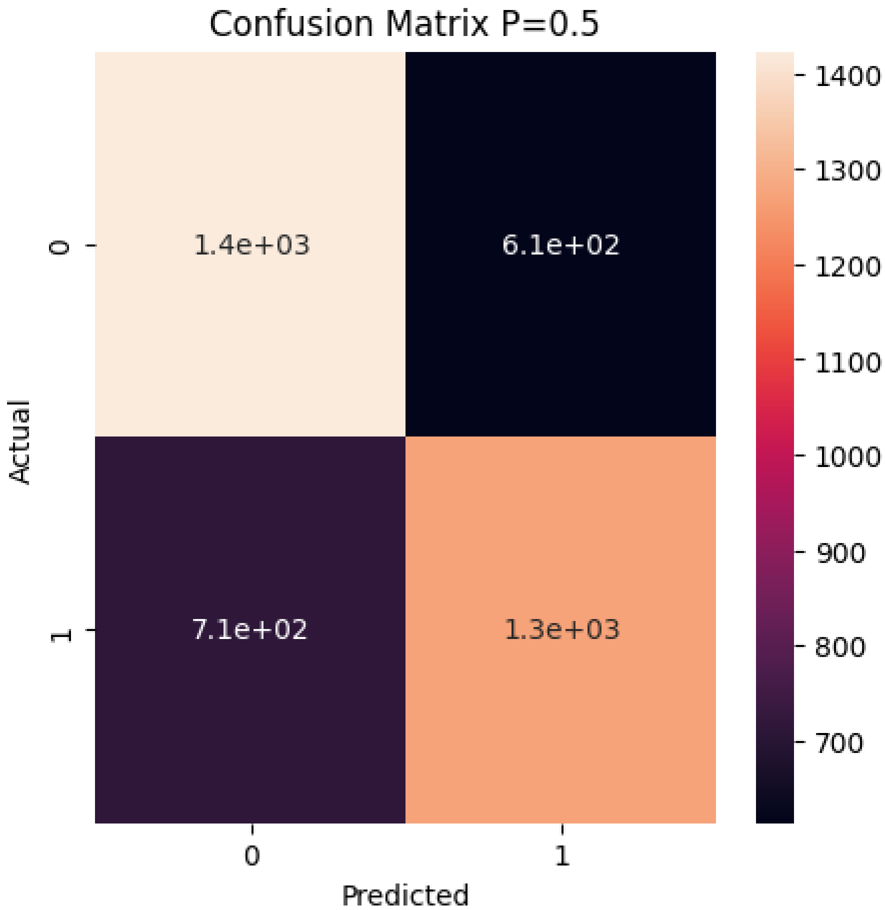

```
AUC
        training                 (min:    0.546, max:    0.743, cur:    0.742)
        validation               (min:    0.679, max:    0.729, cur:    0.728)
Binary_Accuracy
        training                 (min:    0.565, max:    0.675, cur:    0.675)
        validation               (min:    0.629, max:    0.673, cur:    0.670)
False_Negatives
        training                 (min: 1923.000, max: 5799.000, cur: 1929.000)
        validation               (min:  461.000, max:  953.000, cur:  712.000)
False_Positives
        training                 (min: 1859.000, max: 2154.000, cur: 1992.000)
        validation               (min:  538.000, max:  922.000, cur:  614.000)
Precision
        training                 (min:    0.542, max:    0.675, cur:    0.675)
        validation               (min:    0.623, max:    0.675, cur:    0.675)
Precision-Recall
        training                 (min:    0.514, max:    0.731, cur:    0.731)
        validation               (min:    0.650, max:    0.699, cur:    0.697)
Recall
        training                 (min:    0.300, max:    0.683, cur:    0.682)
        validation               (min:    0.520, max:    0.768, cur:    0.641)
True_Negatives
        training                 (min: 3854.000, max: 7759.000, cur: 4016.000)
        validation               (min: 1115.000, max: 1499.000, cur: 1423.000)
True_Positives
        training                 (min: 2484.000, max: 4136.000, cur: 4130.000)
        validation               (min: 1033.000, max: 1525.000, cur: 1274.000)
Loss
        training                 (min:    0.593, max:    0.743, cur:    0.594)
        validation               (min:    0.608, max:    0.646, cur:    0.609)
378/378 [==============================] - 3s 8ms/step - loss: 0.5936 - Binary_Accura
cy: 0.6751 - Precision: 0.6746 - Recall: 0.6816 - True_Positives: 4130.0000 - True_Ne
gatives: 4016.0000 - False_Positives: 1992.0000 - False_Negatives: 1929.0000 - AUC:
0.7424 - Precision-Recall: 0.7307 - val_loss: 0.6089 - val_Binary_Accuracy: 0.6704 -
val_Precision: 0.6748 - val_Recall: 0.6415 - val_True_Positives: 1274.0000 - val_True
_Negatives: 1423.0000 - val_False_Positives: 614.0000 - val_False_Negatives: 712.0000
- val_AUC: 0.7284 - val_Precision-Recall: 0.6974
```

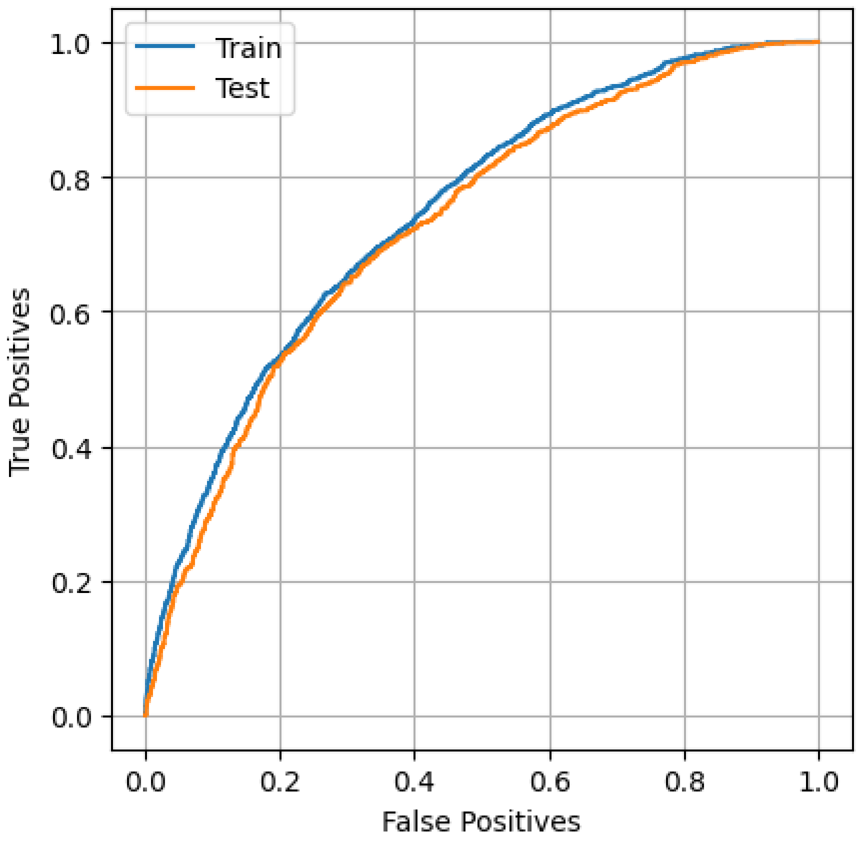
Out[72]: <keras.callbacks.History at 0x140514220>

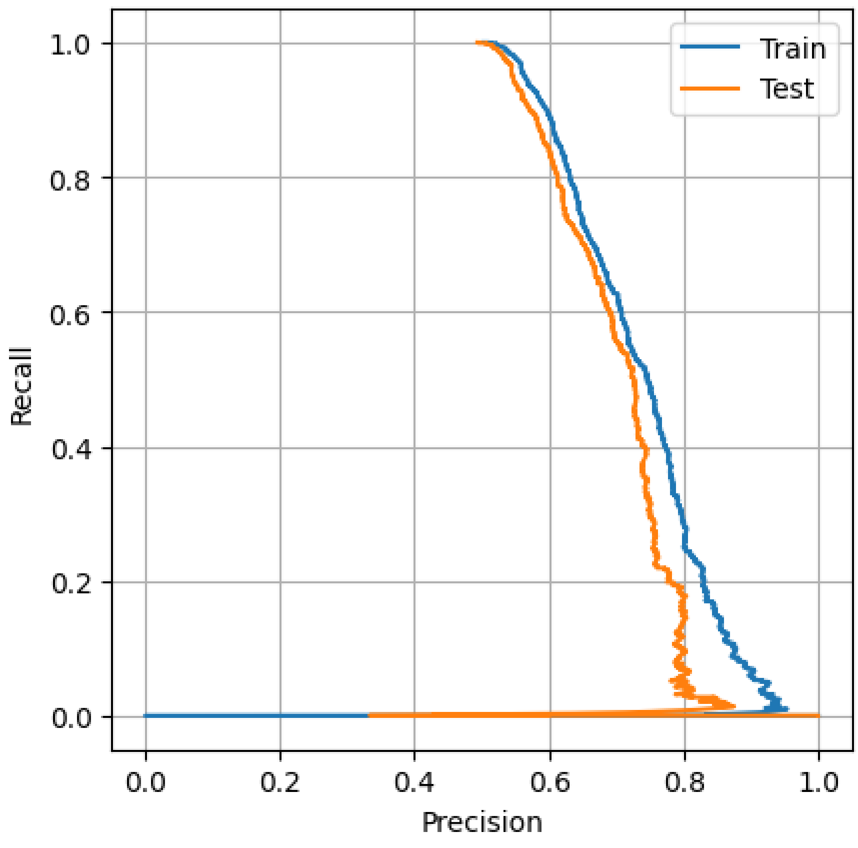
In [73]:
```
evaluate_and_plot(
    model,
    x_train,
    x_test,
    y_train,
    y_test
)
```

```
WARNING:tensorflow:Model was constructed with shape (None, None, 16) for input KerasT
ensor(type_spec=TensorSpec(shape=(None, None, 16), dtype=tf.float32, name='Input_Laye
r'), name='Input_Layer', description="created by layer 'Input_Layer'"), but it was ca
lled on an input with incompatible shape (None, 16).
24/24 [==============================] - 0s 1ms/step
8/8 [==============================] - 0s 1ms/step
```

```
loss:	0.6089179515838623
Binary_Accuracy:	0.6703952550888062
Precision:	0.6747881174087524
Recall: 0.6414904594421387
True_Positives: 1274.0
True_Negatives: 1423.0
False_Positives:	614.0
False_Negatives:	712.0
AUC:	0.7283744215965271
Precision-Recall:	0.6973873376846313
```







Much better results!!