Default Prediction

March 8, 2024

1 Table of Contents

---

[105]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
# import tensorflow as tf
# from tensorflow import keras
import warnings
warnings.filterwarnings('ignore')
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪roc_auc_score, precision_recall_curve
from xgboost import XGBClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
import pickle
import tensorflow as tf
```

```
from keras.layers import Dropout
from keras.callbacks import EarlyStopping
from keras.layers import Activation
from sklearn.base import TransformerMixin
from autogluon.tabular import TabularDataset, TabularPredictor
from sklearn.ensemble import GradientBoostingClassifier, StackingClassifier
import matplotlib.patches as mpatches
import dalex as dx
from sklearn.ensemble import IsolationForest
```

[106]:
```
from utils import you_never_know_where_it_is as ynk
import importlib
importlib.reload(ynk)
from utils.you_never_know_where_it_is import DataPreprocessor,␣
↪PerformanceMetrics, create_metrics_dataframe, create_stacking_classifier
```

2 I. Data Preprocessing

[107]:
```
df = pd.read_csv('./data/loan_train.csv')
```

Let's write a class to write all the data preprocessing steps.

Some plot functions

2.1 I.1 Basic Overview Analysis

Eyeball data, check missing values, duplicates

[108]:
```
# df.info()
```

Check duplicates

[109]:
```
# check duplicates
df.duplicated().sum()

# remove duplicates
df = df.drop_duplicates()
```

Check missing values

[110]:
```
# missing values as a percentage
missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0]
missing_values = missing_values / len(df) * 100
missing_values = missing_values.sort_values(ascending=False)
```

[111]:
```
# number of missing cols containing missing values
print('number of cols containing missing values:', len(missing_values))

# number of missing values in the loan_status
print('number of missing values in the loan_status:', df['loan_status'].
 ↪isnull().sum())
```

```
number of cols containing missing values: 51
number of missing values in the loan_status: 0
```

Apparently, all cols contain missing values except the label. This is quite annoying

Here is the plan to deal with missing values: - if the missing values are less than 1% of the total, we can drop the rows

- if the missing values are less than 10% of the total we can do some further investigation to see if it's appropriate to fill these missing values with median or mode
- If the missing values are greater than 10%, let's investigation to see if there is a reason for the missing values. If not, we can decide to drop some cols wikth extremely high missing values

[112]:
```
# get the index for those have value less than 1% in missing_values
small_missing_values= missing_values[missing_values < 1].index

small_missing_values.tolist()

# drop the rows with missing values in the small_missing_values
df = df.dropna(subset=small_missing_values)
```

[113]:
```
# check the remaining missing values
missing_values = df.isnull().sum()
missing_values = missing_values[missing_values > 0]
missing_values = missing_values / len(df) * 100
missing_values = missing_values.sort_values(ascending=False)
missing_values
```

[113]:
```
next_pymnt_d              92.307172
mths_since_last_record    91.593369
mths_since_last_delinq    63.606225
desc                      31.725304
emp_title                  6.062246
pub_rec_bankruptcies       2.919486
emp_length                 2.547361
dtype: float64
```

These faetures are a bit complicated, and require careful handling. Thus, we will take care of them in **feature engineering**

2.2 I.2 Univariate Analysis

[114]:
```
# initialize the DataPreprocessor
dp = DataPreprocessor()
```

2.2.1 I.2.1 Numerical Features

Visualize histogram of all these numerical features to see the distribution of the data and check for outliers as well as invalid numerical features. Some of them could be potential categorical features instead

[115]:
```
# extract numerical features from the dataset
numerical_features = df.select_dtypes(include=[np.number]).columns.tolist()
```

[116]:
```
ynk.analyze_numerical(df, numerical_features)
```


Distribution of id
Distribution of member_id
Distribution of loan_amnt
Distribution of funded_amnt
Distribution of funded_amnt_inv
Distribution of installment
Distribution of annual_inc
Distribution of dti
Distribution of delinq_2yrs
Distribution of fico_range_low
Distribution of fico_range_high
Distribution of inq_last_6mths
Distribution of mths_since_last_delinq
Distribution of mths_since_last_record
Distribution of open_acc
Distribution of pub_rec
Distribution of revol_bal
Distribution of total_acc
Distribution of out_prncp
Distribution of out_prncp_inv
Distribution of total_rec_late_fee
Distribution of last_pymnt_amnt
Distribution of collections_12_mths_ex_med
Distribution of policy_code
Distribution of acc_now_delinq
Distribution of chargeoff_within_12_mths
Distribution of delinq_amnt
Distribution of pub_rec_bankruptcies
Distribution of tax_liens

It appears that none of them follows a normal distribution. Some distribution is quite suspicious and we need to investigate further.

Some of these variables have just ONE value in the entire column. we can consider removing these columns as they don't provide any information.

- Here is the list of useless numerical cols that don't provide any good information:

[117]:
```
useless_cols = ['chargeoff_within_12_mths', 'collections_12_mths_ex_med',␣
↪'delinq_amnt', 'tax_liens', 'policy_code', 'acc_now_delinq',␣
↪'total_rec_late_fee']
```

Let's just drop them to make our life easier

[118]:
```
df = dp.drop_useless_numerical_cols(df)
```

Some cols are actually numerical but they are in the categorical form. We need to convert them to numerical form

[119]:
```
# df = dp.convert_interest_rate_to_numerical(df)
```

There are still some issues that we need to address, but we can take care of them later in the feature engineering section. Let's just now move on to the categorical section

2.2.2 1.2.2 Categorical Features

[120]:
```
# extract categorical features from the dataset
categorical_features = df.select_dtypes(include=[object]).columns.tolist()

# check the number of levels in these features
for col in categorical_features:
    print(f"{col}: {df[col].nunique()}")
```

```
term: 2
int_rate: 389
grade: 7
sub_grade: 35
emp_title: 21998
emp_length: 11
home_ownership: 5
verification_status: 3
issue_d: 53
loan_status: 2
desc: 20151
purpose: 14
addr_state: 50
earliest_cr_line: 516
```

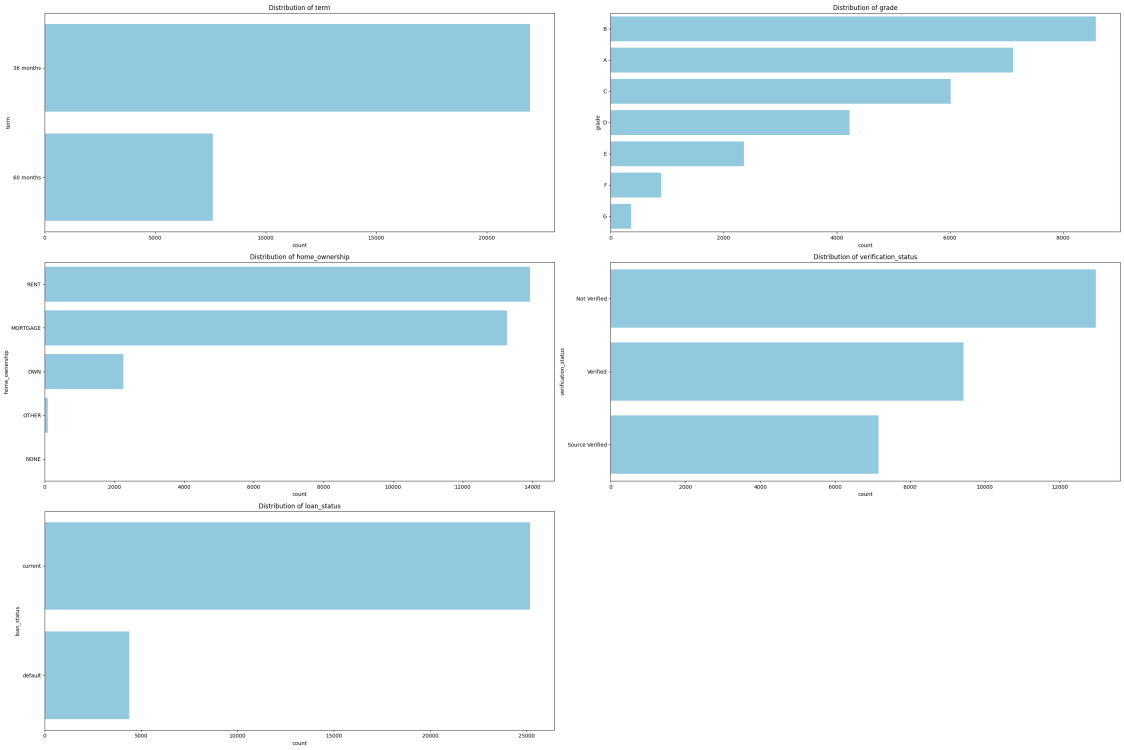
```
revol_util: 1093
next_pymnt_d: 95
last_credit_pull_d: 106
```

Low Cardinality (Less than 10 unique values)

[121]:
```
low_card =  ['term', 'grade', 'home_ownership', 'verification_status',␣
 ↪'loan_status']
```

[122]:
```
ynk.visualize_categorical(df, low_card, figsize=(30, 20))
```

```
term
36 months    74.3%
60 months    25.7%
Name: proportion, dtype: object


grade
B    29.03%
A    24.07%
C    20.35%
D    14.29%
E     7.99%
F     3.02%
G     1.24%
Name: proportion, dtype: object


home_ownership
RENT        47.15%
MORTGAGE     44.9%
OWN          7.64%
OTHER         0.3%
NONE         0.01%
Name: proportion, dtype: object


verification_status
Not Verified       43.87%
Verified            31.9%
Source Verified    24.23%
Name: proportion, dtype: object


loan_status
current    85.18%
default    14.82%
Name: proportion, dtype: object
```



Potential transformation needed: home_ownership

`Med Cardinality`

[123]:
```
med_card = ['sub_grade', 'emp_length', 'purpose', 'addr_state']
```

[124]:
```
ynk.visualize_categorical(df, med_card, figsize=(30, 20))
```

```
sub_grade
B3    7.05%
A4     6.9%
A5     6.6%
B5     6.5%
B4    5.95%
C1    5.38%
B2    5.01%
C2    4.89%
B1    4.52%
A3     4.2%
C3    3.95%
A2    3.72%
D2    3.61%
```

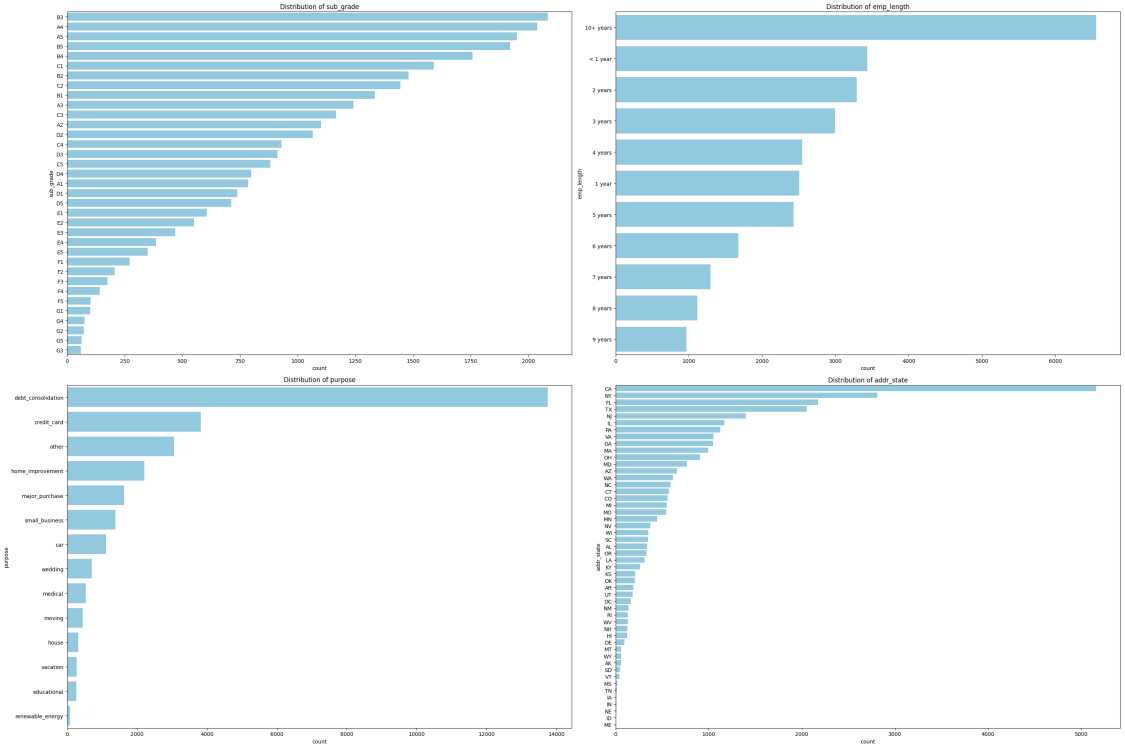
```
C4    3.15%
D3    3.09%
C5    2.98%
D4     2.7%
A1    2.66%
D1     2.5%
D5    2.41%
E1    2.05%
E2    1.86%
E3    1.59%
E4    1.31%
E5    1.18%
F1    0.92%
F2     0.7%
F3    0.59%
F4    0.48%
F5    0.34%
G1    0.34%
G4    0.25%
G2    0.24%
G5    0.21%
G3     0.2%
Name: proportion, dtype: object
```

```
emp_length
10+ years    22.77%
< 1 year     11.92%
2 years      11.42%
3 years      10.39%
4 years       8.83%
1 year         8.7%
5 years       8.43%
6 years       5.81%
7 years        4.5%
8 years       3.88%
9 years       3.35%
Name: proportion, dtype: object
```

```
purpose
debt_consolidation    46.53%
credit_card           12.93%
other                 10.36%
home_improvement       7.48%
major_purchase         5.52%
small_business         4.66%
car                    3.77%
```

```
wedding                2.36%
medical                1.78%
moving                  1.5%
house                  1.05%
vacation               0.93%
educational             0.9%
renewable_energy       0.26%
Name: proportion, dtype: object


addr_state
CA    17.46%
NY     9.52%
FL     7.35%
TX     6.95%
NJ     4.73%
IL     3.95%
PA      3.8%
VA     3.55%
GA     3.55%
MA     3.37%
OH     3.06%
MD     2.59%
AZ     2.24%
WA     2.08%
NC     2.01%
CT     1.94%
CO     1.89%
MI     1.86%
MO     1.83%
MN     1.51%
NV     1.27%
WI      1.2%
SC     1.18%
AL     1.14%
OR     1.12%
LA     1.05%
KY     0.89%
KS     0.71%
OK      0.7%
AR     0.65%
UT     0.63%
DC     0.55%
NM     0.47%
RI     0.45%
WV     0.44%
NH     0.43%
HI     0.41%
```

```
DE      0.32%
MT       0.2%
WY       0.2%
AK      0.19%
SD      0.16%
VT      0.15%
MS      0.06%
TN      0.05%
IA      0.03%
IN      0.03%
NE      0.03%
ID      0.02%
ME       0.0%
Name: proportion, dtype: object
```



Large Cardinality

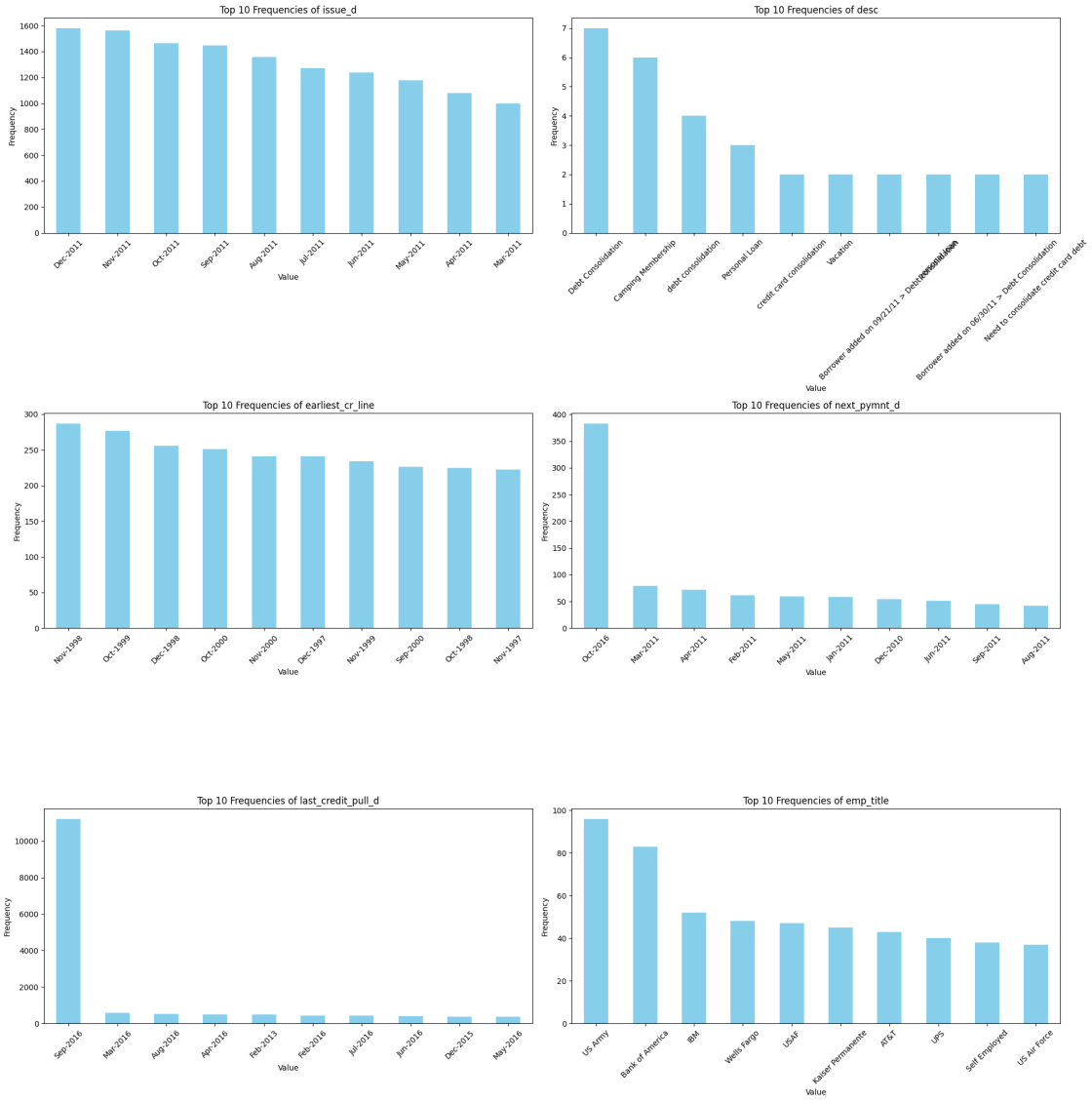
[125]:
```
large_card = ['issue_d', 'desc', 'earliest_cr_line', 'next_pymnt_d',␣
 ↪'last_credit_pull_d', 'emp_title']
```

[126]:
```
# unique values in these features
for col in large_card:
```

```
    print(f"{col}: {df[col].nunique()}")
```

```
issue_d: 53
desc: 20151
earliest_cr_line: 516
next_pymnt_d: 95
last_credit_pull_d: 106
emp_title: 21998
```

[127]:
```
ynk.plot_top_10_frequencies(df, large_card)
```



The majority of title is debt consolidation and debt consolidation loan. On the other hand, Oct 2016 is the most frequent next_pymnt_d. Also, an overwhelming majority of last_credit_pull_d

is Sep 2016

2.3 I.3 Bivariate Analysis

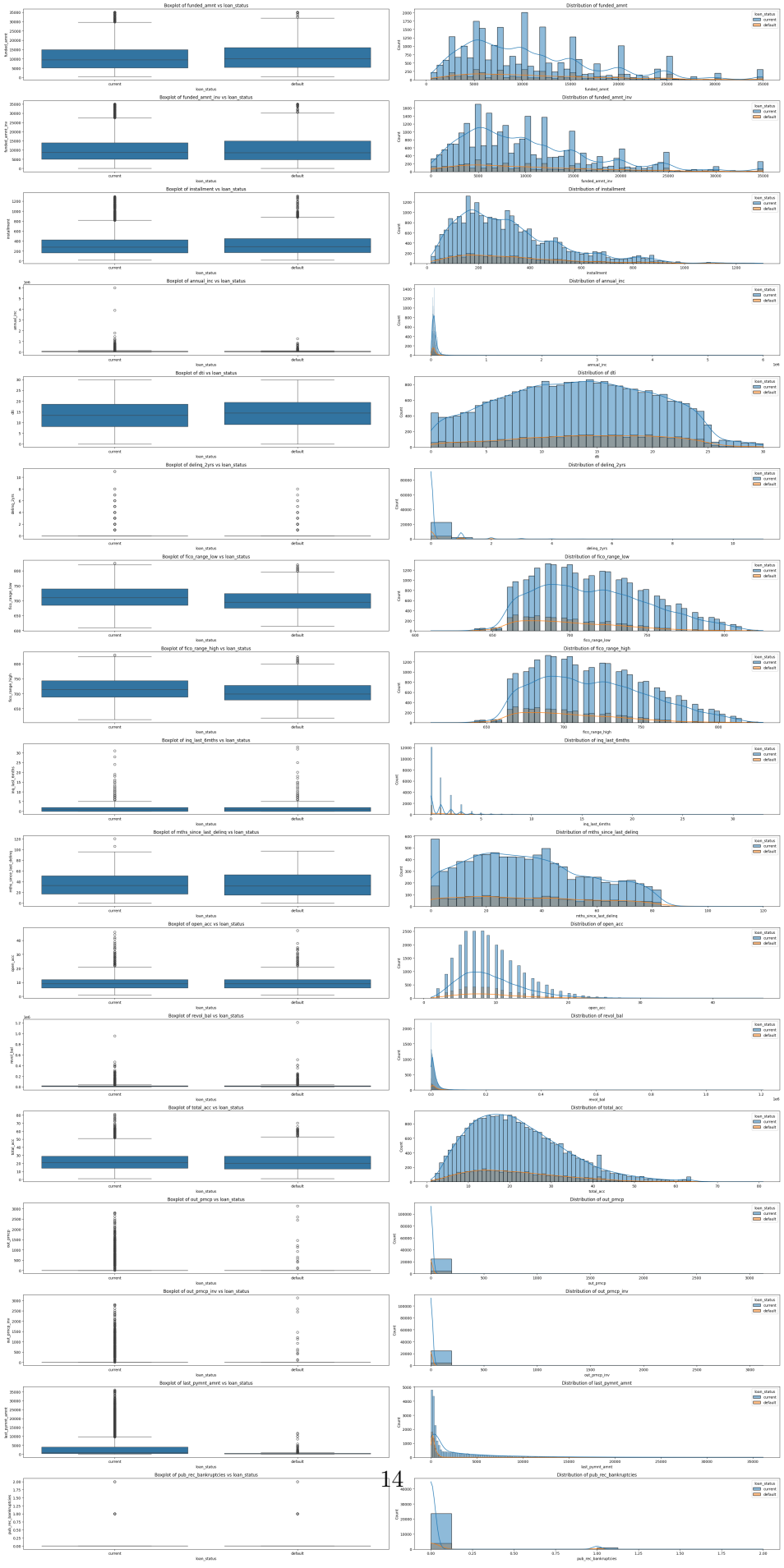
Explore the relationship between the target variable (loan_status) and other features

[128]:
```
# update numerical feature list
numerical_features = df.select_dtypes(include=[np.number]).columns.tolist()
```

The distribution between target variabel and other numerical features is quite similar. However, we can still spot a few difference. For example, the median interest rate for default loan is slightly higher than that of current loan. Meanwhile, fico_range_high is lower for default loan than that of current loan

Numerical Features and Target Variable

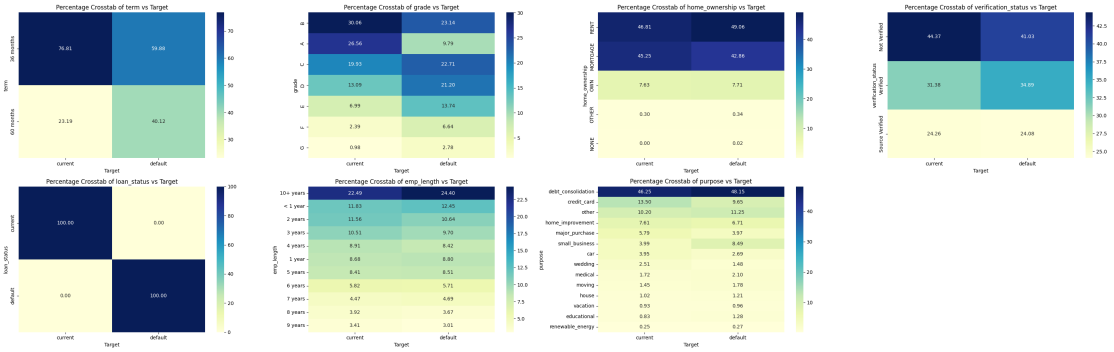
[129]:
```
ynk.plot_boxplot_and_histogram(df, numerical_features, 'loan_status')
```


Boxplot of funded_amnt vs loan_status
Distribution of funded_amnt
Boxplot of funded_amnt_inv vs loan_status
Distribution of funded_amnt_inv
Boxplot of installment vs loan_status
Distribution of installment
Boxplot of annual_inc vs loan_status
Distribution of annual_inc
Boxplot of dti vs loan_status
Distribution of dti
Boxplot of delinq_2yrs vs loan_status
Distribution of delinq_2yrs
Boxplot of fico_range_low vs loan_status
Distribution of fico_range_low
Boxplot of fico_range_high vs loan_status
Distribution of fico_range_high
Boxplot of inq_last_6mths vs loan_status
Distribution of inq_last_6mths
Boxplot of mths_since_last_delinq vs loan_status
Distribution of mths_since_last_delinq
Boxplot of open_acc vs loan_status
Distribution of open_acc
Boxplot of revol_bal vs loan_status
Distribution of revol_bal
Boxplot of total_acc vs loan_status
Distribution of total_acc
Boxplot of out_prncp vs loan_status
Distribution of out_prncp
Boxplot of out_prncp_inv vs loan_status
Distribution of out_prncp_inv
Boxplot of last_pymnt_amnt vs loan_status
Distribution of last_pymnt_amnt
Boxplot of pub_rec_bankruptcies vs loan_status
Distribution of pub_rec_bankruptcies
loan_status
current
default

Categorical Features and Target Variable

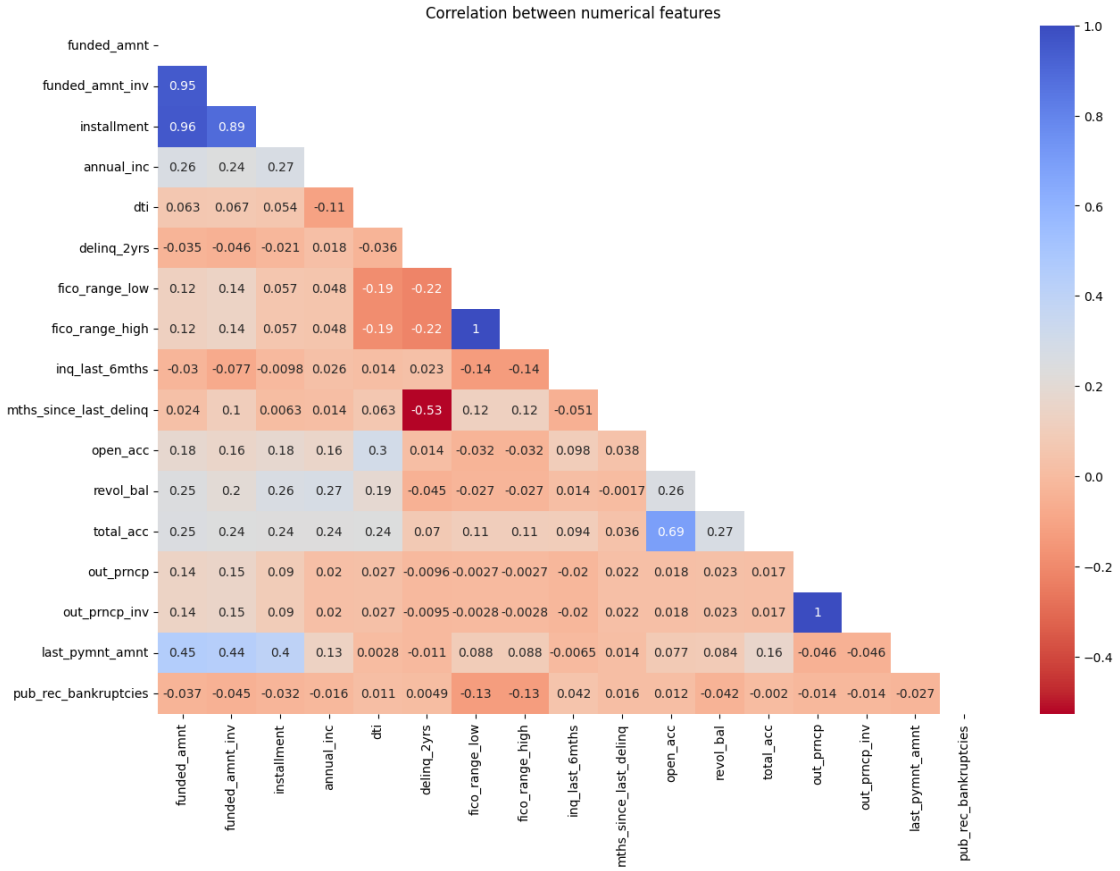
Low Cardinality

[130]:
```
ynk.plot_percentage_crosstab(df, df['loan_status'], low_card +␣
↪['emp_length','purpose'])
```



2.4 I.4 Multivariate Analysis

[131]:
```
# check the correlation between numerical features
correlation = df[numerical_features].corr()

plt.figure(figsize=(15, 10))
mask = np.zeros_like(correlation, dtype=bool)
mask[np.triu_indices_from(mask)] = True  # Mask the upper triangle
sns.heatmap(correlation, annot=True, cmap='coolwarm_r', mask=mask)
plt.title('Correlation between numerical features')
plt.show()
```



Some numerical features are highly correlated with one another. We can consider removing some of them to avoid multicollinearity

3 II. Feature Engineering

[132]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('loan_status',␣
↪axis=1), df['loan_status'], test_size=0.2, random_state=42)
```

[133]:
```
# get the data ready for model building
X_train = dp.feature_engineering_pipeline(X_train)

X_test = dp.feature_engineering_pipeline(X_test)

y_train = dp.convert_target_to_numerical(y_train)

y_test = dp.convert_target_to_numerical(y_test)
```

4 III. Model Building

[134]:
```
# update numerical and categorical features list
numerical_features = X_train.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = X_train.select_dtypes(include=[object]).columns.tolist()
```

[135]:
```
# Create a pipeline to preprocess the data
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# combine
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
```

Before we buid models, let's conduct some anomaly detection to see if there is any abnormal present in our dataset

[136]:
```
X_train_anomaly = X_train.copy()

X = preprocessor.fit_transform(X_train_anomaly)

# Initialize the Isolation Forest model
iso_forest = IsolationForest(n_estimators=100,
                             contamination=0.01, # use 'auto'
                             random_state=42)

# Fit the model
iso_forest.fit(X)

# Predict anomalies (-1 for anomalies, 1 for normal)
anomalies = iso_forest.predict(X)

# Add the anomalies to the original data (optional)
X_train_anomaly['anomaly'] = anomalies

# predict anomalies in the training set
```

```
# Filter the anomalies
anomalies_data = X_train_anomaly[X_train_anomaly['anomaly'] == -1]

# Analyze the anomalies
print(f"Number of anomalies detected {anomalies_data.shape[0]} ", )

anomalies_data.head().T
```

```
Number of anomalies detected 237
```

[136]:

	7764	4130	23226	\
funded_amnt	35000.0	28000.0	20800.0	
funded_amnt_inv	34975.0	28000.0	20775.0	
term	0	1	0	
int_rate	7.49	12.42	11.48	
installment	1088.56	628.81	685.75	
grade	0	1	1	
sub_grade	A4	B4	B2	
emp_title	0	1	1	
emp_length	2.0	4.0	10.0	
home_ownership	1	0	2	
annual_inc	162500.0	130000.0	140000.0	
verification_status	1	2	0	
issue_d	62.966667	59.9	82.2	
desc	1	0	1	
purpose	small_business	major_purchase	home_improvement	
addr_state	CT	PA	CA	
dti	27.92	6.66	21.77	
delinq_2yrs	0.0	0.0	0.0	
earliest_cr_line	333.733333	370.333333	140.033333	
fico_range_low	770.0	770.0	790.0	
fico_range_high	774.0	774.0	794.0	
inq_last_6mths	0.0	4.0	1.0	
mths_since_last_delinq	0	0	0	
open_acc	16.0	11.0	6.0	
revol_bal	2367.0	14506.0	682.0	
revol_util	7.7	16.8	3.3	
total_acc	30.0	37.0	15.0	
out_prncp	0.0	0.0	0.0	
out_prncp_inv	0.0	0.0	0.0	
last_pymnt_amnt	12646.0	21555.69	20999.81	
next_pymnt_d	0	0	0	
last_credit_pull_d	1	0	0	
pub_rec_bankruptcies	0	0	0	
anomaly	-1	-1	-1	

	11439	228
funded_amnt	28000.0	20000.0
funded_amnt_inv	27975.0	18847.373057
term	1	1
int_rate	20.11	12.69
installment	743.55	451.9
grade	6	1
sub_grade	G1	B5
emp_title	0	0
emp_length	6.0	7.0
home_ownership	0	2
annual_inc	140000.0	85000.0
verification_status	1	2
issue_d	66.0	57.866667
desc	0	0
purpose	medical	home_improvement
addr_state	OR	LA
dti	19.86	10.21
delinq_2yrs	0.0	0.0
earliest_cr_line	169.433333	111.6
fico_range_low	660.0	725.0
fico_range_high	664.0	729.0
inq_last_6mths	2.0	0.0
mths_since_last_delinq	0	0
open_acc	17.0	13.0
revol_bal	32101.0	17537.0
revol_util	81.1	54.8
total_acc	55.0	31.0
out_prncp	0.0	1345.54
out_prncp_inv	0.0	1345.54
last_pymnt_amnt	501.49	451.9
next_pymnt_d	0	1
last_credit_pull_d	1	1
pub_rec_bankruptcies	0	0
anomaly	-1	-1

Some anomaly could be detected here: - High income

- Last payment amount for the top 3 is relatively high, higher than its funded amount
- The first borrower has a very high funded amount and high dti

4.1 III. 1 Logisitic Regression

[137]:
```
# execute the pipeline
lr_pipeline = ynk.train_model(LogisticRegression(random_state=0), preprocessor,␣
 ↪ X_train, y_train)
```

```
# obtain prediction
lr_pred, lr_pred_prob = ynk.get_predictions(lr_pipeline, X_test)

# obtain performance
lr_performance = PerformanceMetrics(y_test, lr_pred, lr_pred_prob)

# print out the performance
lr_performance.print_performance_metrics('Logistic Regression')
```

```
Performance Metrics of the model Logistic Regression:

Accuracy: 0.9134
Precision: 0.7438
Recall: 0.6596
ROC AUC: 0.9483
F1 Score: 0.6992
```

4.2 III. 2 Random Forest

[138]:
```
# execute the pipeline
rf_pipeline = ynk.train_model(RandomForestClassifier(random_state=0),␣
 ↪preprocessor,  X_train, y_train)

# obtain prediction
rf_pred, rf_pred_prob = ynk.get_predictions(rf_pipeline, X_test)

# obtain performance
rf_performance = PerformanceMetrics(y_test, rf_pred, rf_pred_prob)

# print out the performance
rf_performance.print_performance_metrics('Base Random Forest')
```

```
Performance Metrics of the model Base Random Forest:

Accuracy: 0.9010
Precision: 0.8042
Recall: 0.4645
ROC AUC: 0.9473
F1 Score: 0.5889
```

4.2.1 III. 2.1 Random Forest Tunning

[139]:
```
# rf_best_params = ynk.tune_rf_rs(rf_pipeline, X_train, y_train)

# print(rf_best_params)
```

With this best params from the random search, we can narrow the search space down, and apply grid search on neighbor values to find the optimal values

[140]:
```
# rf_best_params = ynk.tune_rf_gs(rf_pipeline, X_train, y_train)
```

[141]:
```
# # Create the RandomForestClassifier with the best parameters

# rf_best_params = {k.replace('classifier__', ''): v for k, v in rf_best_params.
 ↪items()}

# tuned_rf_classifier = RandomForestClassifier(random_state=0, **rf_best_params)

# # apply the best hyperparameters to the random forest model
# tuned_rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
#                                     ('classifier', tuned_rf_classifier)])

# # fit the model to the training data
# tuned_rf_pipeline.fit(X_train, y_train)
```

[142]:
```
# # export the model to a pickle file
# tuned_rf_pkl = './models/tuned_rf_pipeline.pkl'
# with open(tuned_rf_pkl, 'wb') as file:
#     pickle.dump(tuned_rf_pipeline, file)
```

[143]:
```
# import the tuned random forest model from the pickle file
with open('./models/tuned_rf_pipeline.pkl', 'rb') as file:
    tuned_rf_pipeline = pickle.load(file)
```

[144]:
```
tuned_rf_pred, tuned_rf_pred_prob = ynk.get_predictions(tuned_rf_pipeline,␣
 ↪X_test)

# obtain performance
tuned_rf_performance = PerformanceMetrics(y_test, tuned_rf_pred,␣
 ↪tuned_rf_pred_prob)

# print out the performance
tuned_rf_performance.print_performance_metrics('Tuned Random Forest')
```

```
Performance Metrics of the model Tuned Random Forest:

Accuracy: 0.8968
Precision: 0.7967
Recall: 0.4346
ROC AUC: 0.9506
F1 Score: 0.5624
```

4.3 III. 3 XGBoost

[145]:
```
# execute the pipeline
xgb_pipeline = ynk.train_model(XGBClassifier(random_state=0), preprocessor,
 ↪X_train, y_train)

# obtain prediction
xgb_pred, xgb_pred_prob = ynk.get_predictions(xgb_pipeline, X_test)

# obtain performance
xgb_performance = PerformanceMetrics(y_test, xgb_pred, xgb_pred_prob)

# print out the performance
xgb_performance.print_performance_metrics('Base XGBoost')
```

```
Performance Metrics of the model Base XGBoost:

Accuracy: 0.9212
Precision: 0.7633
Recall: 0.7007
ROC AUC: 0.9568
F1 Score: 0.7306
```

4.3.1 III. 3.1 XGBoost Tunning

[146]:
```
# xgb_best_params = ynk.tune_xgb_rs(xgb_pipeline, X_train, y_train)
# print(xgb_best_params)
```

With the best params from the random search, we can narrow the search space down, and apply grid search on neighbor values to find the optimal values

[147]:
```
# xgb_best_params = ynk.tune_xgb_gs(xgb_pipeline, X_train, y_train)
```

[148]:
```
# # Create the xgboost with the best parameters

# xgb_best_params = {k.replace('classifier__', ''): v for k, v in
 ↪xgb_best_params.items()}

# tuned_xgb_classifier = XGBClassifier(random_state=0, **xgb_best_params)

# # apply the best hyperparameters to the random forest model
# tuned_xgb_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
#                               ('classifier', tuned_xgb_classifier)])

# # fit the model to the training data
# tuned_xgb_pipeline.fit(X_train, y_train)
```

[149]:
```
# # export the model to a pickle file
# tuned_xgb_pkl = './models/tuned_xgb_pipeline.pkl'
# with open(tuned_xgb_pkl, 'wb') as file:
#     pickle.dump(tuned_xgb_pipeline, file)
```

[150]:
```
# import the tuned xgb model from the pickle file
with open('./models/tuned_xgb_pipeline.pkl', 'rb') as file:
    tuned_xgb_pipeline = pickle.load(file)
```

Now make predictions and obtain eval metrics on the tuned model

[151]:
```
tuned_xgb_pred, tuned_xgb_pred_prob = ynk.get_predictions(tuned_xgb_pipeline,␣
 ↪X_test)

# obtain performance
tuned_xgb_performance = PerformanceMetrics(y_test, tuned_xgb_pred,␣
 ↪tuned_xgb_pred_prob)

# pri_xgbout the performance
tuned_xgb_performance.print_performance_metrics('Tuned XGBoost')
```

```
Performance Metrics of the model Tuned XGBoost:

Accuracy: 0.9254
Precision: 0.7814
Recall: 0.7095
ROC AUC: 0.9603
F1 Score: 0.7438
```

4.4 III. 4 Neural Network

[152]:
```
# get the number of dimension after transform in the pipeline
print('Number of features after one-hot encoding:', len(rf_pipeline.
 ↪named_steps['preprocessor'].transform(X_train).toarray()[0]))
```

```
Number of features after one-hot encoding: 129
```

[153]:
```
# Step 1: Define a function that returns a Keras model

def create_model():
    model = Sequential()
    model.add(Dense(129, input_dim = 129, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
    return model
```

```
# Step 2: Wrap the Keras model with KerasClassifier (you may need to install␣
 ↪scikeras first)
neural_network = KerasClassifier(build_fn=create_model, epochs=100,␣
 ↪batch_size=128, verbose=0)

# Step 3: Include the wrapped model in your pipeline
nn_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('nn', neural_network),
])

# Fit the model
nn_pipeline.fit(X_train, y_train)
```

[153]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['funded_amnt',
                                                   'funded_amnt_inv', 'term',
                                                   'int_rate', 'installment',
                                                   'grade', 'emp_title',
                                                   'emp_length',
                                                   'home_ownership',
                                                   'annual_inc',
                                                   'verification_status',
                                                   'issue_d', 'desc', 'dti',
                                                   'delinq_2yrs',
                                                   '…
                                                   'next_pymnt_d',
                                                   'last_credit_pull_d',
                                                   'pub_rec_bankruptcies']),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                  ('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['sub_grade', 'purpose',
                                                   'addr_state'])])),
                ('nn',
                 KerasClassifier(batch_size=128, build_fn=<function create_model
at 0x000001A367002B60>, epochs=100, verbose=0))])
```

[154]:
```
# get predictions
nn_pred, nn_pred_prob = ynk.get_predictions(nn_pipeline, X_test)

# performance metrics
nn_performance = PerformanceMetrics(y_test, nn_pred, nn_pred_prob)

# print out the performance
nn_performance.print_performance_metrics('Neural Network')
```

```
Performance Metrics of the model Neural Network:

Accuracy: 0.8929
Precision: 0.6675
Recall: 0.5942
ROC AUC: 0.9167
F1 Score: 0.6287
```

4.4.1 III. 4.1 Neural Network Tunning

[155]:
```
# Fit the preprocessor on the training data
preprocessor.fit(X_train)

# Transform the training and test data using the preprocessor (the pipeline␣
 ↪that we defined earlier)
X_train_processed = preprocessor.transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Convert sparse matrices to dense tensors
X_train_processed = X_train_processed.toarray()
X_test_processed = X_test_processed.toarray()
```

[156]:
```
# # Define a function to create a Keras model with dropout layers
# def create_model(hidden_layers=1, neurons=32, activation='relu',␣
 ↪dropout_rate=0.2):
#     model = Sequential()
#     model.add(Dense(128, input_dim=X_train_processed.shape[1]))
#     model.add(Dense(neurons, activation=activation))
#     model.add(Dropout(dropout_rate))
#     for _ in range(hidden_layers):
#         model.add(Dense(neurons, activation=activation))
#         model.add(Dropout(dropout_rate))
#     model.add(Dense(1, activation='sigmoid'))
#     model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy', tf.keras.metrics.Recall()])
#     return model

# # Define early stopping callback
```

```
# early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# # Define the hyperparameters grid including dropout rate and number of epochs
# param_grid = {
#     'hidden_layers': [2, 3, 4],
#     'neurons': [32, 64, 128],
#     'dropout_rate': [0.1, 0.2, 0.3, 0.4],
#     'epochs': [50, 100, 150],
# }

# # Perform grid search manually
# best_score = 0
# best_params = None
# for hidden_layers in param_grid['hidden_layers']:
#     for neurons in param_grid['neurons']:
#         for dropout_rate in param_grid['dropout_rate']:
#             for epochs in param_grid['epochs']:
#                 # Create and compile the model
#                 model = create_model(hidden_layers=hidden_layers,␣
 ↪neurons=neurons, dropout_rate=dropout_rate)

#                 # Train the model
#                 model.fit(X_train_processed, y_train, epochs=epochs,␣
 ↪validation_data=(X_test_processed, y_test), callbacks=[early_stopping],␣
 ↪verbose=2)

#                 # Evaluate the model
#                 _, _, recall = model.evaluate(X_test_processed, y_test,␣
 ↪verbose=0)

#                 # Check if this is the best model so far
#                 if recall > best_score:
#                     best_score = recall
#                     best_params = {'hidden_layers': hidden_layers,
#                                    'neurons': neurons,
#                                    'dropout_rate': dropout_rate,
#                                    'epochs': epochs}

# print("Best Parameters:", best_params)
# print("Best Recall:", best_score)
```

[157]:
```
# optimal_epochs = best_params['epochs']
# # remove the epochs from the best_params
# best_params.pop('epochs')
```

[158]:
```
# tuned_nn = create_model(**best_params)
# # train the the optimal nn model
# tuned_nn.fit(X_train_processed, y_train, epochs=optimal_epochs,␣
 ↪validation_data=(X_test_processed, y_test), callbacks=[early_stopping],␣
 ↪verbose=2)
```

[159]:
```
# # export the model to a pickle file
# tuned_nn_pkl = './models/tuned_nn.pkl'
# with open(tuned_nn_pkl, 'wb') as file:
#     pickle.dump(tuned_nn, file)
```

[160]:
```
# import the tuned xgb model from the pickle file
with open('./models/tuned_nn.pkl', 'rb') as file:
    tuned_nn = pickle.load(file)
```

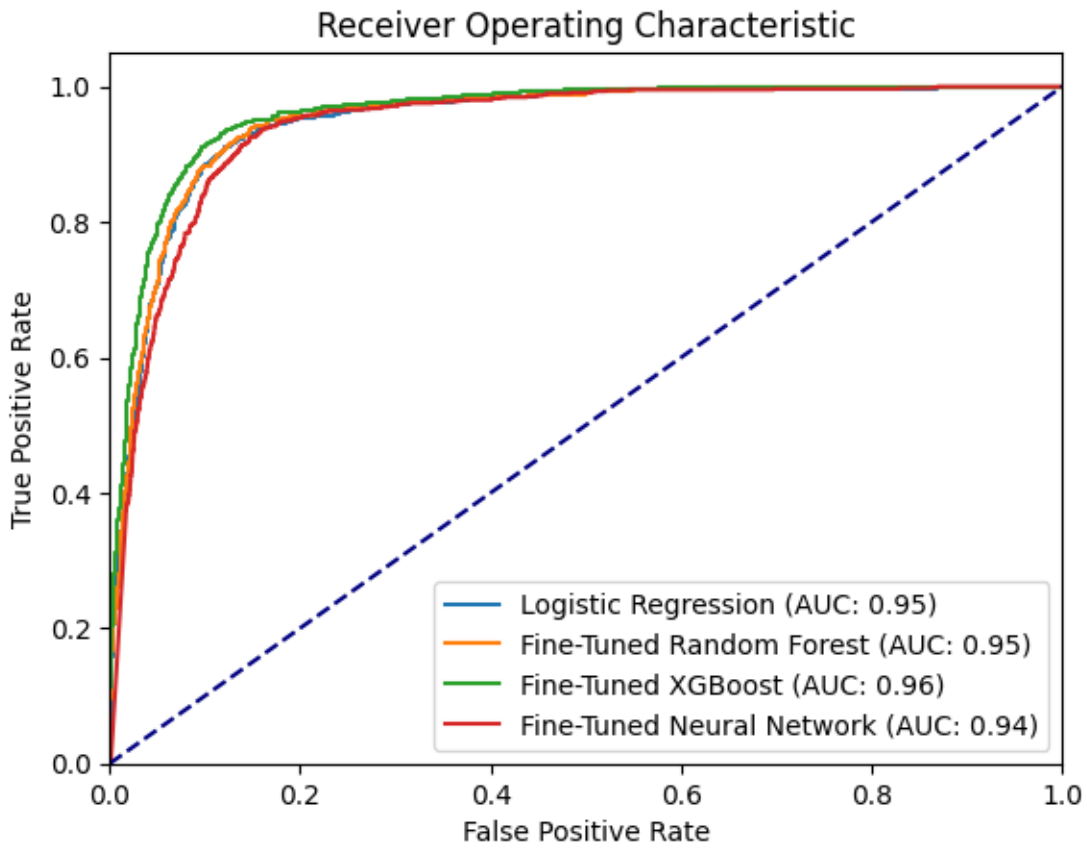
Now make predictions and obtain eval metrics on the tuned model

[161]:
```
# get predictions
tuned_nn_pred_prob = tuned_nn.predict(X_test_processed)
tuned_nn_pred = (tuned_nn_pred_prob > 0.5).astype(int)

# obtain performance
tuned_nn_performance = PerformanceMetrics(y_test, tuned_nn_pred,␣
 ↪tuned_nn_pred_prob)

# print out the performance
tuned_nn_performance.print_performance_metrics('Tuned Neural Network')
```

```
185/185 [==============================] - 0s 807us/step
Performance Metrics of the model Tuned Neural Network:

Accuracy: 0.8895
Precision: 0.5951
Recall: 0.8636
ROC AUC: 0.9414
F1 Score: 0.7047
```

Merge all the performance metrics together

[162]:
```
ynk.plot_roc_curve([lr_performance.y_pred_prob,
                    tuned_rf_performance.y_pred_prob,
                    tuned_xgb_performance.y_pred_prob,
                    tuned_nn_performance.y_pred_prob],
                   ['Logistic Regression',
                    'Fine-Tuned Random Forest',
                    'Fine-Tuned XGBoost',
                    'Fine-Tuned Neural Network'],
                   y_test)
```



4.5 III. 5 Stacking Classifier

In this stacking classifier, we will use the best models from the previous section to build a new model. Also, we included gradient boosting (GBM) as an additional model in the stack. Params used in these model have already been tuned in the previous section

[163]:
```
# stacked_pipeline = create_stacking_classifier(preprocessor=preprocessor)
```

[164]:
```
# stacked_pipeline.fit(X_train, y_train)
```

[165]:
```
# # export the model to a pickle file
# stacked_pkl = './models/stacked.pkl'
# with open(stacked_pkl, 'wb') as file:
#     pickle.dump(stacked_pipeline, file)
```

[166]:
```
# import the stacked model in
with open('./models/stacked.pkl', 'rb') as file:
    stacked_pipeline = pickle.load(file)
```

[167]:
```
# obtain prediction
stacked_pred, stacked_pred_prob = ynk.get_predictions(stacked_pipeline, X_test)

# obtain performance
stacked_performance = PerformanceMetrics(y_test, stacked_pred,␣
 ↪stacked_pred_prob)

# print out the performance
stacked_performance.print_performance_metrics('Stacking Classifier')
```

```
Performance Metrics of the model Stacking Classifier:

Accuracy: 0.9210
Precision: 0.7836
Recall: 0.6663
ROC AUC: 0.9569
F1 Score: 0.7202
```

5 IV. Model Evaluation

5.1 IV. 1 Global Model Explanations

Merge all the eval metrics for all models together

[168]:
```
metrics_tbl = pd.concat([create_metrics_dataframe(lr_performance, 'Logistic␣
 ↪Regression'),
                         create_metrics_dataframe(rf_performance, 'Random␣
 ↪Forest'),
                         create_metrics_dataframe(tuned_rf_performance, 'Tuned␣
 ↪Random Forest'),
                         create_metrics_dataframe(xgb_performance, 'XGBoost'),
                         create_metrics_dataframe(tuned_xgb_performance, 'Tuned␣
 ↪XGBoost'),
                         create_metrics_dataframe(nn_performance, 'Neural␣
 ↪Network'),
                         create_metrics_dataframe(tuned_nn_performance, 'Tuned␣
 ↪Neural Network'),
                         create_metrics_dataframe(stacked_performance,␣
 ↪'Stacking Classifier')])

metrics_tbl.round(4)
```

[168]:

	accuracy	precision	recall	roc_auc	f1_score
Logistic Regression	0.9134	0.7438	0.6596	0.9483	0.6992
Random Forest	0.9010	0.8042	0.4645	0.9473	0.5889
Tuned Random Forest	0.8968	0.7967	0.4346	0.9506	0.5624
XGBoost	0.9212	0.7633	0.7007	0.9568	0.7306

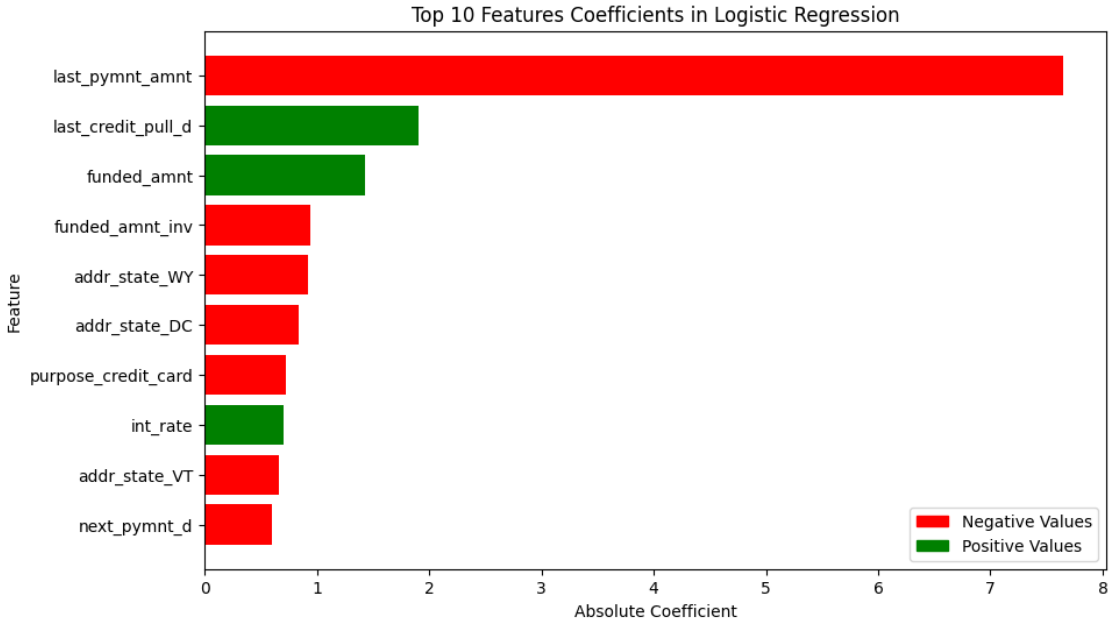
Tuned XGBoost	0.9254	0.7814	0.7095	0.9603	0.7438
Neural Network	0.8929	0.6675	0.5942	0.9167	0.6287
Tuned Neural Network	0.8895	0.5951	0.8636	0.9414	0.7047
Stacking Classifier	0.9210	0.7836	0.6663	0.9569	0.7202

Since we apply standard scaling to the numerical features, we can use the coefficients from the logistic regression to interpret the importance of the features. We can also use the feature importance from the random forest and xgboost to interpret the importance of the features

5.1.1 IV. 1.1 Feature Importance

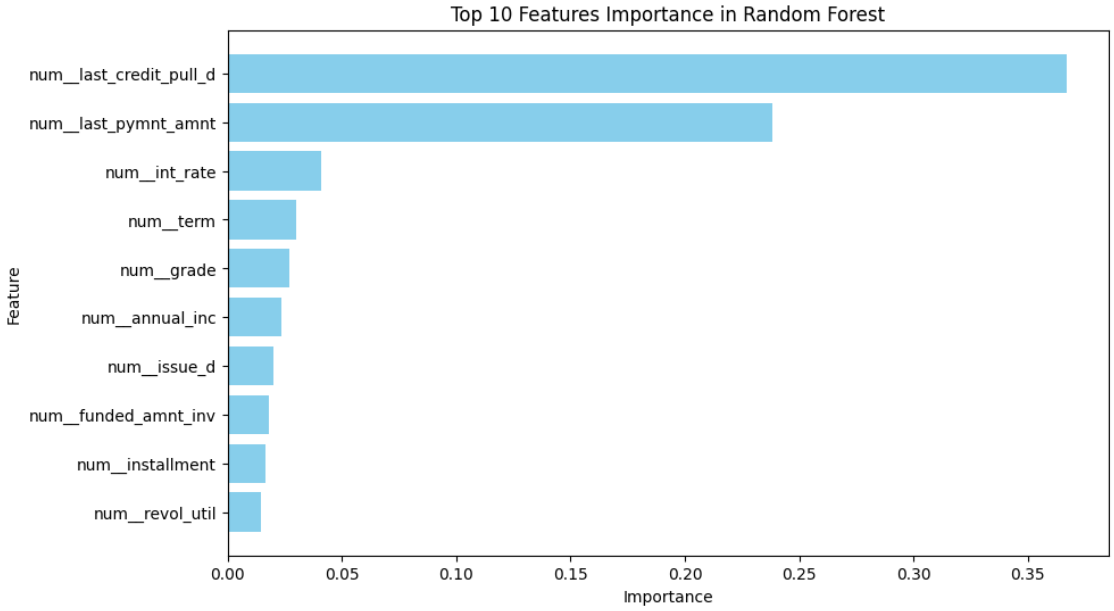
Logisi Regression

[169]:
```
ynk.plot_lr_fi(lr_pipeline)
```



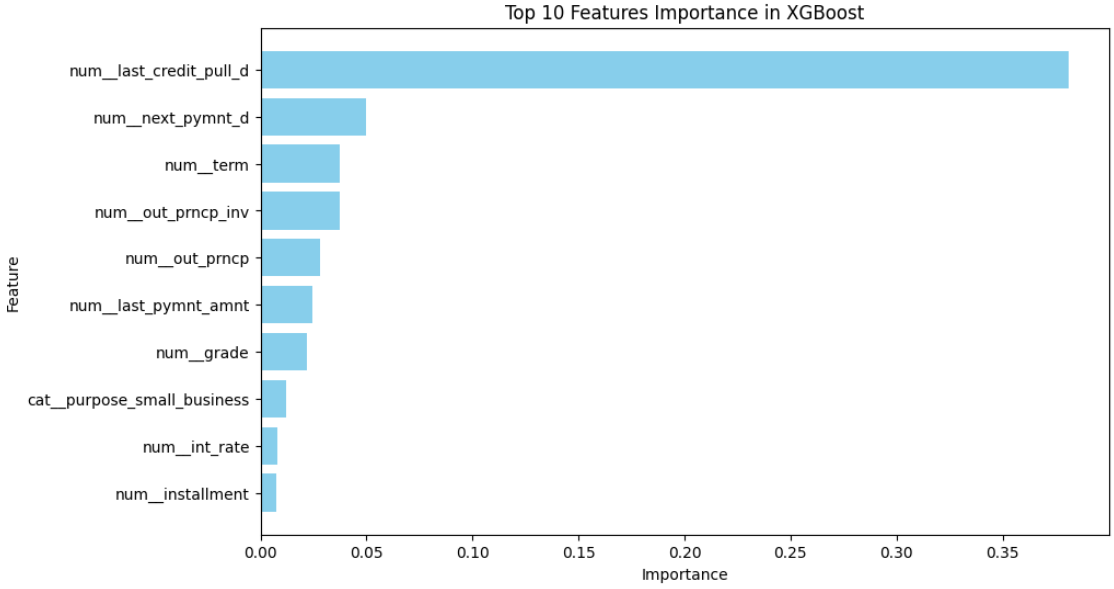
Random Forest

[170]:
```
ynk.plot_tree_fi(tuned_rf_pipeline, title = 'Top 10 Features Importance in␣
↪Random Forest')
```



XGBoost

[171]:
```
ynk.plot_tree_fi(tuned_xgb_pipeline, title = 'Top 10 Features Importance in␣
↪XGBoost')
```



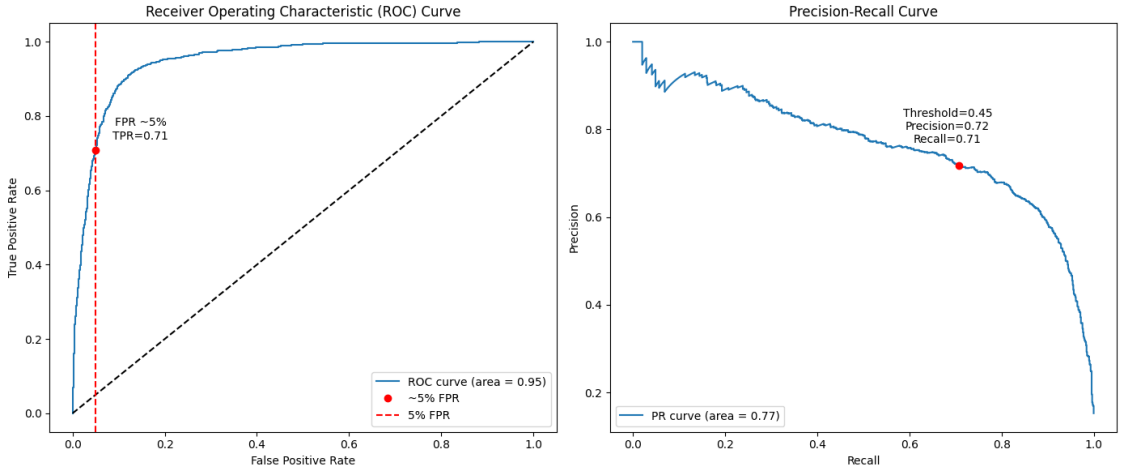
5.1.2 IV. 1.2 ROC Curve & PR Curve

Visualize the roc curve for 5% FPR for our top models

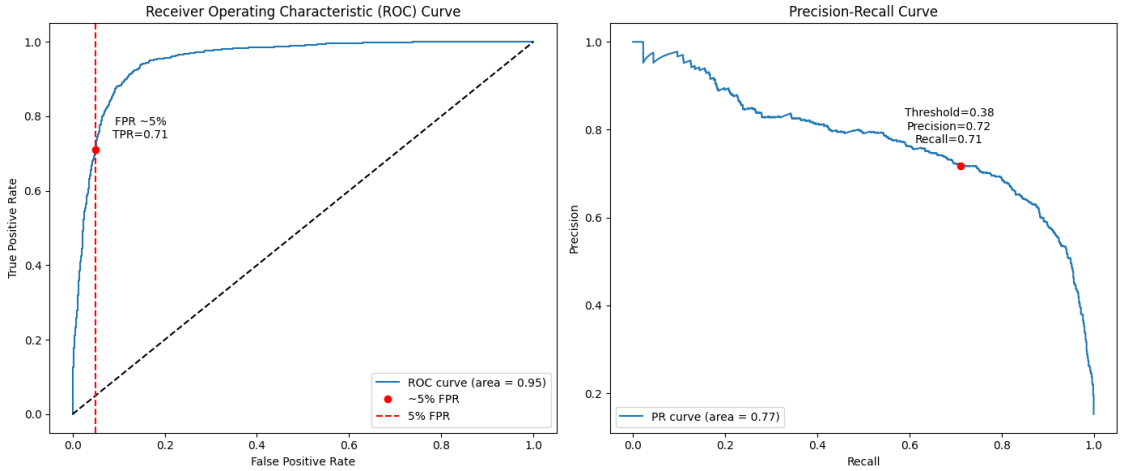
[172]:
```
top_models_pred = {'Logistic Regression': lr_performance.y_pred_prob,
                   'Tuned Random Forest': tuned_rf_performance.y_pred_prob,
                   'Tuned XGBoost': tuned_xgb_performance.y_pred_prob,
                   'Tuned Neural Network': tuned_nn_performance.y_pred_prob,
                   'Stacking Classifier' : stacked_performance.y_pred_prob}
```

[173]:
```
for model, pred in top_models_pred.items():
    print(f'ROC and PR curve for {model}')
    ynk.plot_roc_pr_curves(y_test, pred, fpr_percentile= 5)
    print('\n')
```

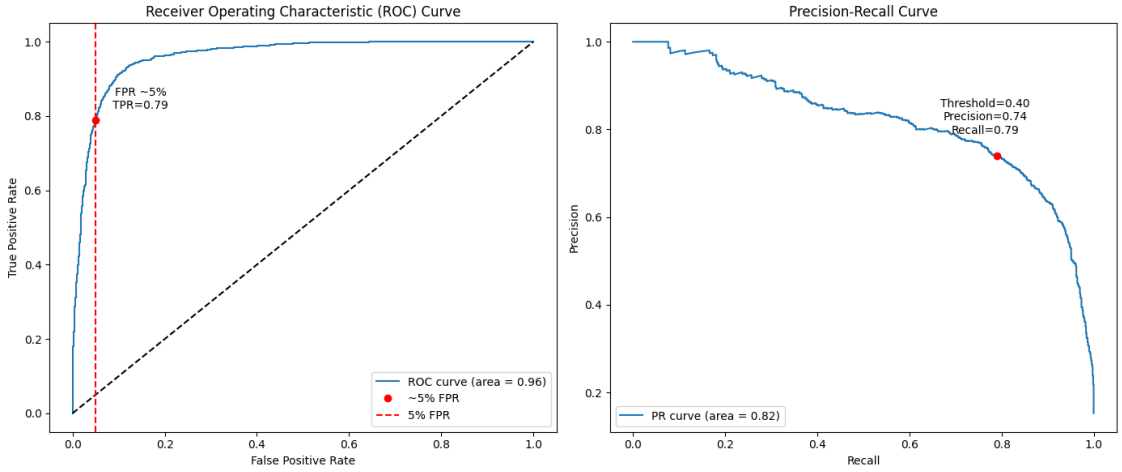
```
ROC and PR curve for Logistic Regression
WARNING:tensorflow:Detecting that an object or model or tf.train.Checkpoint is
being deleted with unrestored values. See the following logs for the specific
values in question. To silence these warnings, use `status.expect_partial()`.
See https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint#restorefor
details about the status object returned by the restore function.
WARNING:tensorflow:Value in checkpoint could not be found in the restored
object: (root).keras_api.metrics.0.total
WARNING:tensorflow:Value in checkpoint could not be found in the restored
object: (root).keras_api.metrics.0.count
WARNING:tensorflow:Value in checkpoint could not be found in the restored
object: (root).keras_api.metrics.1.total
WARNING:tensorflow:Value in checkpoint could not be found in the restored
object: (root).keras_api.metrics.1.count
WARNING:tensorflow:Value in checkpoint could not be found in the restored
object: (root).keras_api.metrics.2.true_positives
WARNING:tensorflow:Value in checkpoint could not be found in the restored
object: (root).keras_api.metrics.2.false_negatives
```



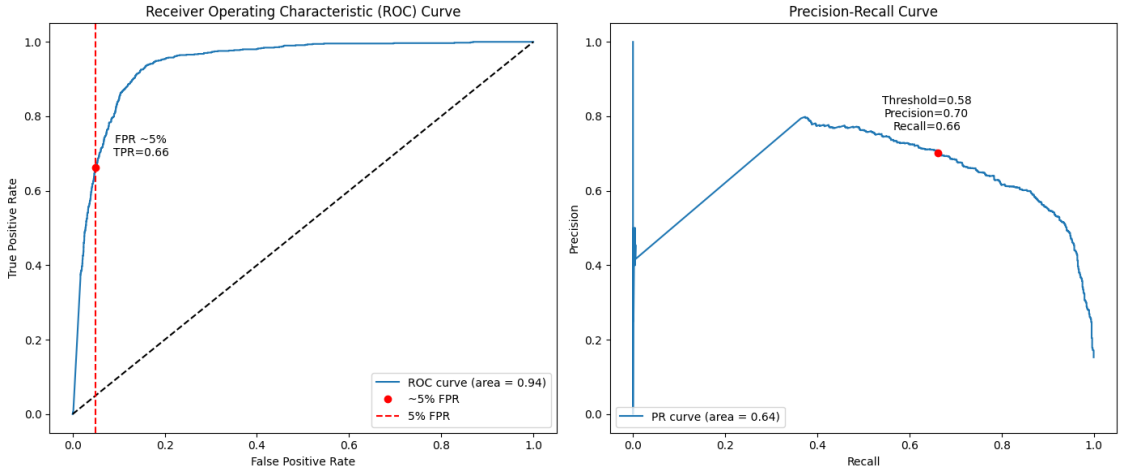
ROC and PR curve for Tuned Random Forest



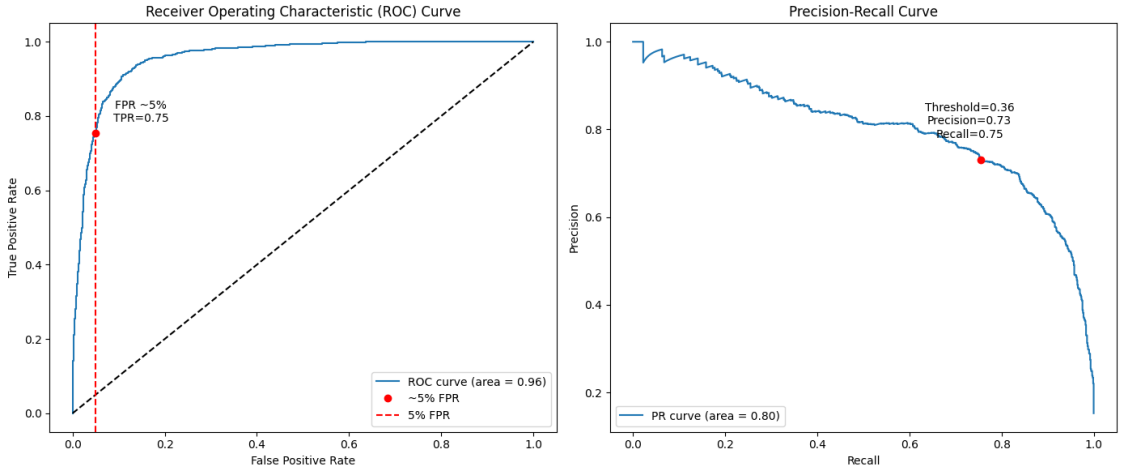
ROC and PR curve for Tuned XGBoost



ROC and PR curve for Tuned Neural Network



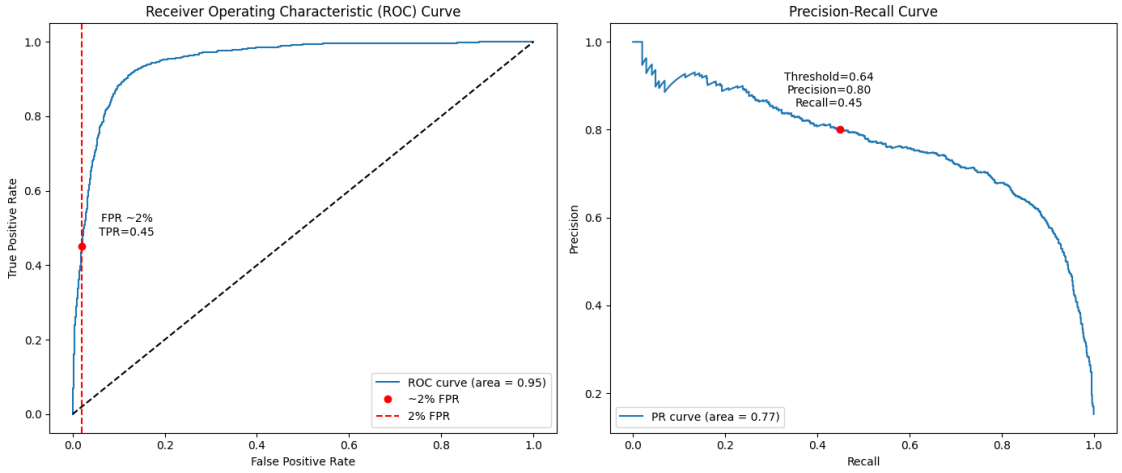
ROC and PR curve for Stacking Classifier



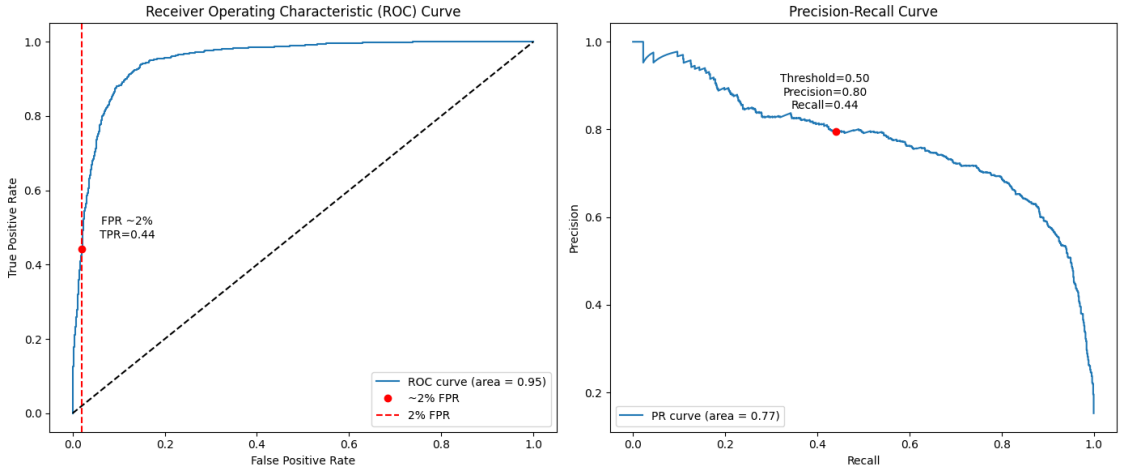
Visualize the roc curve for 2% FPR for our top models

[174]:
```
for model, pred in top_models_pred.items():
    print(f'ROC and PR curve for {model}')
    ynk.plot_roc_pr_curves(y_test, pred, fpr_percentile= 2)
    print('\n')
```

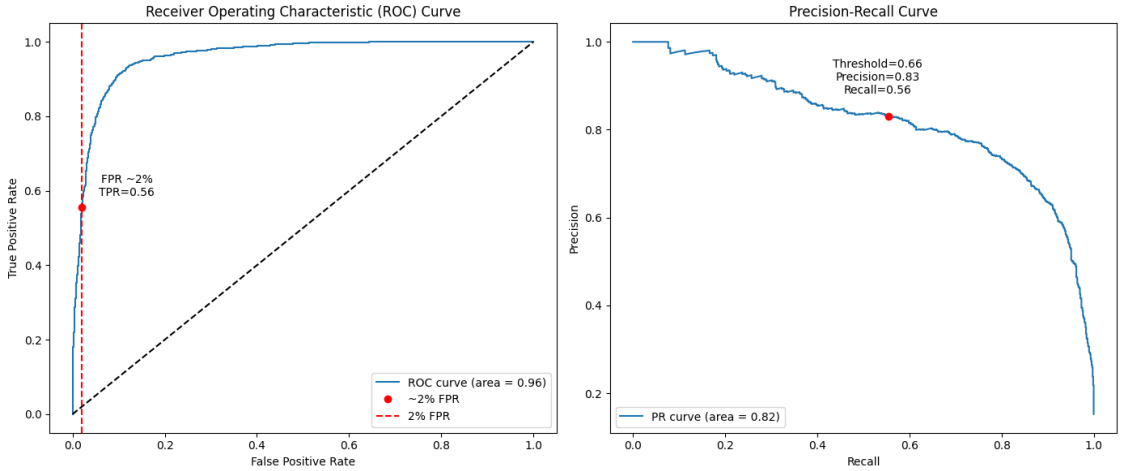
ROC and PR curve for Logistic Regression



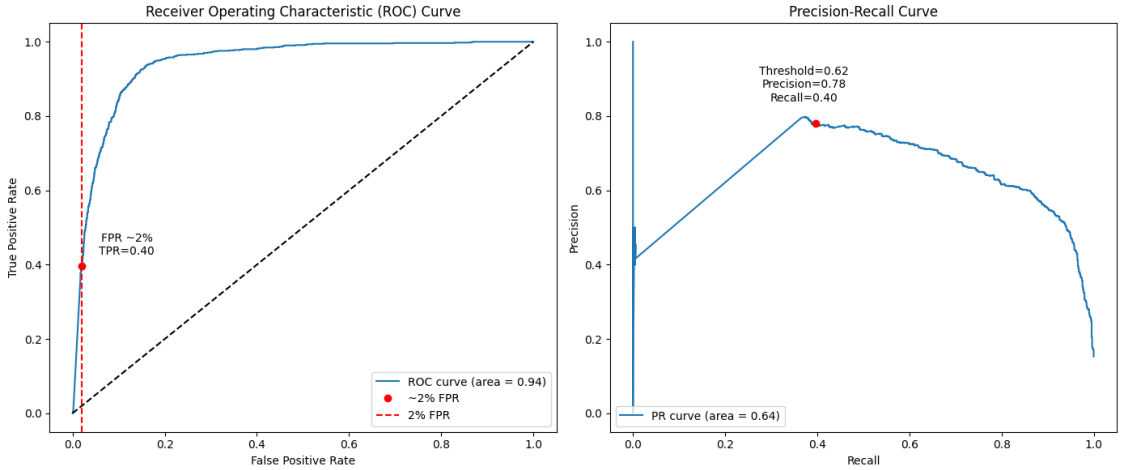
ROC and PR curve for Tuned Random Forest



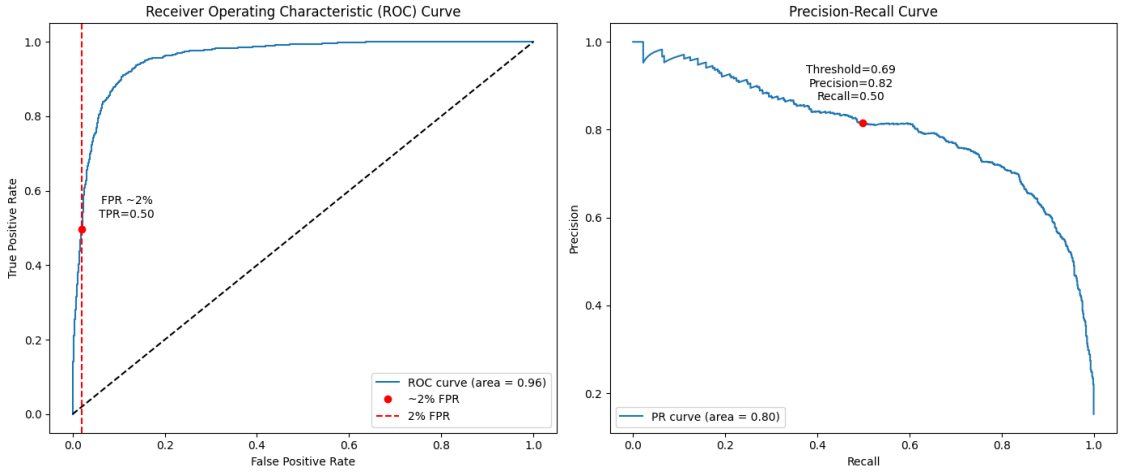
ROC and PR curve for Tuned XGBoost



ROC and PR curve for Tuned Neural Network



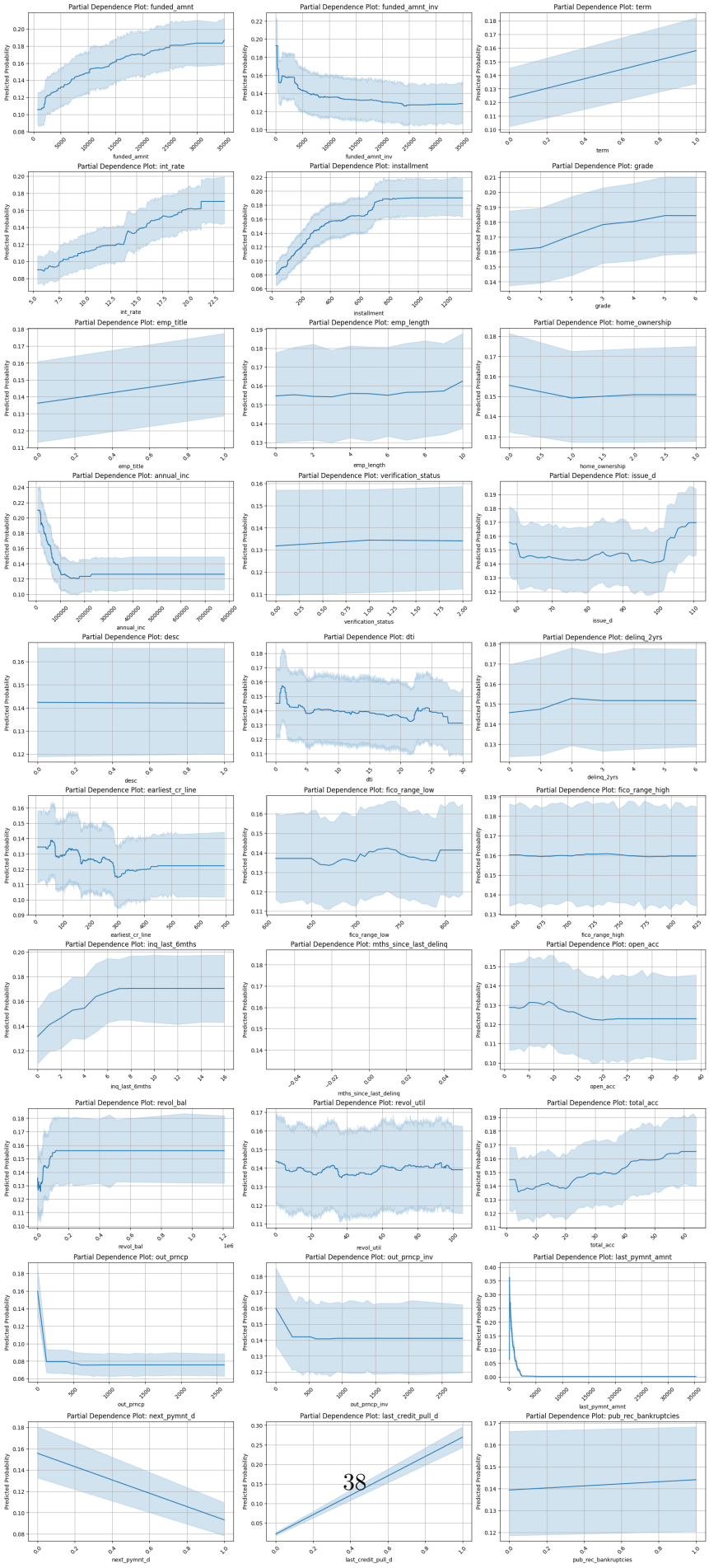
ROC and PR curve for Stacking Classifier



5.1.3 IV. 1.3 Partial Dependence Plot

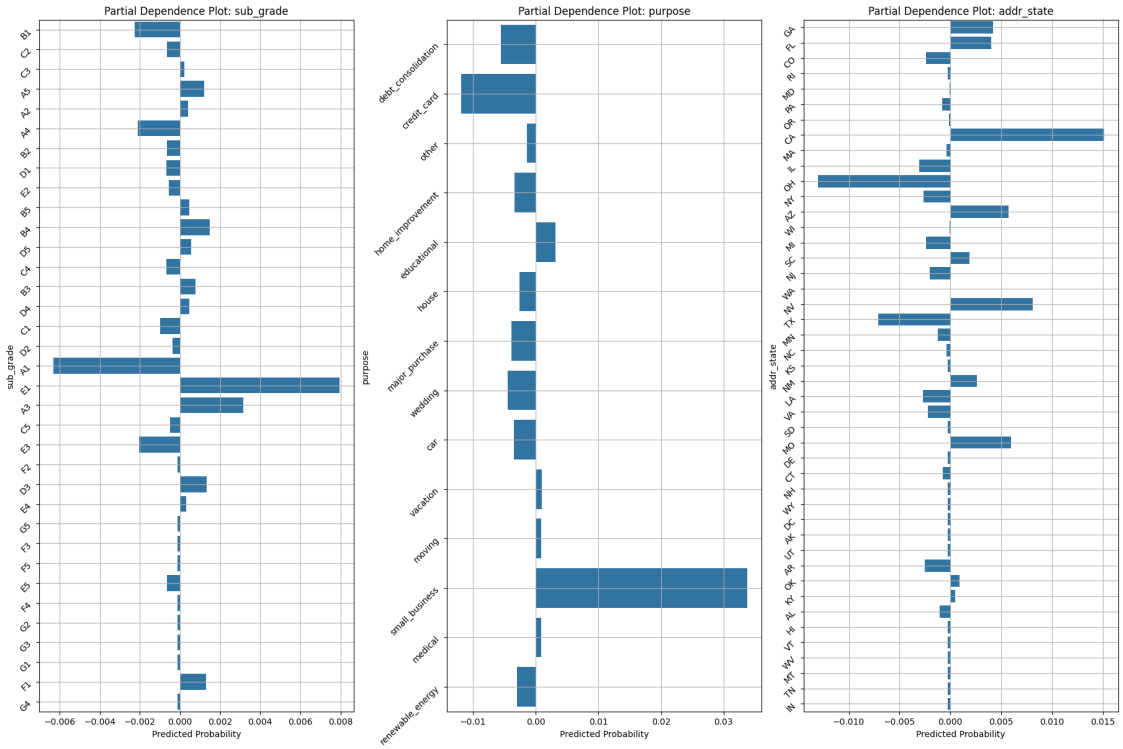
Numerical Features

[175]:
```
num_rows = (len(numerical_features) + 2) // 3

# Create subplots with three columns, creating all subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=3, figsize=(18, 4*num_rows),␣
 ↪constrained_layout=True)

# Plot PDPs for each numerical feature
for i, num in enumerate(numerical_features):
    row = i // 3
    col = i % 3
    ynk.pdp_plot_numeric(axes[row, col], X_train, num, sample_n=500,␣
 ↪pipeline=tuned_xgb_pipeline)

# Hide unused subplots
for i in range(len(numerical_features), num_rows * 3):
    row = i // 3
    col = i % 3
    axes[row, col].axis('off')

plt.show()
```


Partial Dependence Plot: funded_amnt
Partial Dependence Plot: funded_amnt_inv
Partial Dependence Plot: term
Partial Dependence Plot: int_rate
Partial Dependence Plot: installment
Partial Dependence Plot: grade
Partial Dependence Plot: emp_title
Partial Dependence Plot: emp_length
Partial Dependence Plot: home_ownership
Partial Dependence Plot: annual_inc
Partial Dependence Plot: verification_status
Partial Dependence Plot: issue_d
Partial Dependence Plot: desc
Partial Dependence Plot: dti
Partial Dependence Plot: delinq_2yrs
Partial Dependence Plot: earliest_cr_line
Partial Dependence Plot: fico_range_low
Partial Dependence Plot: fico_range_high
Partial Dependence Plot: inq_last_6mths
Partial Dependence Plot: mths_since_last_delinq
Partial Dependence Plot: open_acc
Partial Dependence Plot: revol_bal
Partial Dependence Plot: revol_util
Partial Dependence Plot: total_acc
Partial Dependence Plot: out_prncp
Partial Dependence Plot: out_prncp_inv
Partial Dependence Plot: last_pymnt_amnt
Partial Dependence Plot: next_pymnt_d
Partial Dependence Plot: last_credit_pull_d
Partial Dependence Plot: pub_rec_bankruptcies
Predicted Probability

Categorical Features

[176]:
```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 12),␣
 ↪constrained_layout=True)

# Plot PDPs for each categorical feature
for i, var in enumerate(categorical_features[:3]):
    ynk.pdp_plot_categorical(axes[i], X_train, var, sample_n=500,␣
 ↪pipeline=tuned_xgb_pipeline)
```



5.2 IV. 2 Local Model Explanations

Well, of course XGBoost is the champion model, let's look into the local model explanation generated by this model

[177]:
```
pipeline_explainer = dx.Explainer(tuned_xgb_pipeline, X_test, y_test)
pipeline_explainer
```

```
Preparation of a new explainer is initiated

  -> data              : 5912 rows 33 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a
```

```
numpy.ndarray.
  -> target variable   : 5912 values
  -> model_class       : xgboost.sklearn.XGBClassifier (default)
  -> label             : Not specified, model's class short name will be used.
(default)
  -> predict function  : <function yhat_proba_default at 0x000001A364BFAFC0>
will be used (default)
  -> predict function  : Accepts only pandas.DataFrame, numpy.ndarray causes
problems.
  -> predicted values  : min = 0.000364, mean = 0.144, max = 0.981
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.93, mean = 0.00856, max = 0.997
  -> model_info        : package sklearn

A new explainer has been created!
```

[177]:
```
<dalex._explainer.object.Explainer at 0x1a3059abcd0>
```

5.2.1 IV 2.1 Top 10 best True Positive

[178]:
```
X_test['pred']= tuned_xgb_pred
X_test['pred_proba']= tuned_xgb_pred_prob
X_test['loan_status'] = y_test
```

[179]:
```
top_10_tp = (X_test
             .query('loan_status == pred and loan_status == 1')
             .sort_values(by='pred_proba', ascending=False)
             .head(10)
             .reset_index(drop=True)
)
```

[180]:
```
top_10_tp.head(10).T
```

[180]:

	0	1	2	\
funded_amnt	25000.0	24000.0	12000.0	
funded_amnt_inv	300.005174	274.997104	11975.0	
term	0	0	1	
int_rate	12.04	11.72	19.03	
installment	830.84	793.94	311.49	
grade	2	2	4	
sub_grade	C5	C4	E2	
emp_title	0	0	0	
emp_length	4.0	0.0	2.0	
home_ownership	1	0	0	
annual_inc	105000.0	140000.0	35004.0	
verification_status	0	0	0	

issue_d	105.533333	105.533333	57.866667
desc	0	0	0
purpose	small_business	small_business	debt_consolidation
addr_state	WA	GA	OH
dti	3.73	8.39	12.14
delinq_2yrs	0.0	0.0	0.0
earliest_cr_line	26.366667	176.566667	147.1
fico_range_low	700.0	705.0	725.0
fico_range_high	704.0	709.0	729.0
inq_last_6mths	2.0	8.0	0.0
mths_since_last_delinq	0	0	0
open_acc	2.0	5.0	4.0
revol_bal	1554.0	16298.0	9047.0
revol_util	9.1	80.7	98.3
total_acc	2.0	10.0	13.0
out_prncp	0.0	0.0	0.0
out_prncp_inv	0.0	0.0	0.0
last_pymnt_amnt	830.84	793.94	36.97
next_pymnt_d	0	0	0
last_credit_pull_d	1	1	1
pub_rec_bankruptcies	1	0	0
pred	1	1	1
pred_proba	0.981287	0.979662	0.972376
loan_status	1	1	1

	3	4	5 \
funded_amnt	14400.0	27575.0	15000.0
funded_amnt_inv	14300.0	27575.0	14975.0
term	1	1	1
int_rate	20.3	17.58	21.67
installment	383.92	693.94	411.48
grade	4	3	5
sub_grade	E5	D4	F3
emp_title	1	0	0
emp_length	NaN	10.0	2.0
home_ownership	0	0	2
annual_inc	54000.0	64000.0	33000.0
verification_status	1	1	2
issue_d	59.9	58.866667	57.866667
desc	1	0	0
purpose	credit_card	home_improvement	debt_consolidation
addr_state	AZ	NJ	NY
dti	13.13	10.76	14.36
delinq_2yrs	0.0	0.0	0.0
earliest_cr_line	101.466667	129.9	160.3
fico_range_low	670.0	715.0	660.0
fico_range_high	674.0	719.0	664.0

inq_last_6mths	1.0	2.0	1.0
mths_since_last_delinq	0	0	0
open_acc	21.0	7.0	9.0
revol_bal	11356.0	13087.0	12964.0
revol_util	42.1	72.7	97.1
total_acc	33.0	23.0	20.0
out_prncp	0.0	0.0	0.0
out_prncp_inv	0.0	0.0	0.0
last_pymnt_amnt	383.92	693.94	412.98
next_pymnt_d	0	0	0
last_credit_pull_d	1	1	1
pub_rec_bankruptcies	0	0	0
pred	1	1	1
pred_proba	0.971895	0.971747	0.971676
loan_status	1	1	1

	6	7	\
funded_amnt	30000.0	21000.0	
funded_amnt_inv	29575.0	20975.0	
term	1	1	
int_rate	23.13	19.42	
installment	847.96	549.62	
grade	6	4	
sub_grade	G2	E3	
emp_title	0	1	
emp_length	3.0	1.0	
home_ownership	1	0	
annual_inc	102500.0	42000.0	
verification_status	1	1	
issue_d	58.866667	57.866667	
desc	1	1	
purpose	debt_consolidation	small_business	
addr_state	CA	FL	
dti	22.57	11.26	
delinq_2yrs	0.0	0.0	
earliest_cr_line	201.933333	149.133333	
fico_range_low	670.0	700.0	
fico_range_high	674.0	704.0	
inq_last_6mths	0.0	1.0	
mths_since_last_delinq	0	0	
open_acc	14.0	9.0	
revol_bal	35174.0	18695.0	
revol_util	97.3	72.2	
total_acc	25.0	22.0	
out_prncp	0.0	0.0	
out_prncp_inv	0.0	0.0	
last_pymnt_amnt	100.0	549.62	

next_pymnt_d	0	0
last_credit_pull_d	1	1
pub_rec_bankruptcies	0	0
pred	1	1
pred_proba	0.969432	0.969046
loan_status	1	1

	8	9
funded_amnt	17000.0	26000.0
funded_amnt_inv	16975.0	25975.0
term	1	1
int_rate	17.14	16.77
installment	423.78	642.96
grade	4	3
sub_grade	E3	D2
emp_title	0	0
emp_length	4.0	10.0
home_ownership	1	0
annual_inc	35000.0	53000.0
verification_status	1	1
issue_d	67.033333	57.866667
desc	0	0
purpose	debt_consolidation	home_improvement
addr_state	CA	NC
dti	22.87	11.77
delinq_2yrs	0.0	0.0
earliest_cr_line	248.533333	161.3
fico_range_low	690.0	730.0
fico_range_high	694.0	734.0
inq_last_6mths	0.0	1.0
mths_since_last_delinq	0	0
open_acc	13.0	5.0
revol_bal	6121.0	7560.0
revol_util	41.1	72.7
total_acc	27.0	14.0
out_prncp	0.0	0.0
out_prncp_inv	0.0	0.0
last_pymnt_amnt	423.78	642.96
next_pymnt_d	0	0
last_credit_pull_d	1	1
pub_rec_bankruptcies	1	0
pred	1	1
pred_proba	0.965679	0.965577
loan_status	1	1

[181]:
```
ynk.plot_local_breakdown_interactions(top_10_tp, pipeline_explainer)
```

5.2.2 IV 2.2 Top 10 False Negative

[182]:
```
top_10_fn = (X_test
             .query('loan_status != pred and loan_status == 1')
             .sort_values(by='pred_proba', ascending=True)
             .head(10)
             .reset_index(drop=True)
)
top_10_fn.head(10).T
```

[182]:

	0	1	2	\
funded_amnt	15925.0	16000.0	15000.0	
funded_amnt_inv	15850.0	15900.0	14731.08	
term	1	1	0	
int_rate	20.77	15.58	10.99	
installment	428.77	385.53	491.03	
grade	6	3	1	
sub_grade	G4	D3	B4	
emp_title	0	0	0	
emp_length	6.0	2.0	2.0	
home_ownership	0	0	1	
annual_inc	120000.0	78000.0	45000.0	
verification_status	1	1	0	
issue_d	71.033333	76.133333	78.166667	
desc	0	0	0	
purpose	debt_consolidation	debt_consolidation	credit_card	
addr_state	VA	MN	IL	
dti	9.5	8.57	20.56	
delinq_2yrs	1.0	0.0	0.0	
earliest_cr_line	263.833333	154.2	141.0	
fico_range_low	665.0	700.0	710.0	
fico_range_high	669.0	704.0	714.0	
inq_last_6mths	1.0	1.0	0.0	
mths_since_last_delinq	0	0	0	
open_acc	17.0	9.0	10.0	
revol_bal	29164.0	7265.0	17811.0	
revol_util	51.8	34.3	68.8	
total_acc	45.0	22.0	36.0	
out_prncp	0.0	0.0	0.0	
out_prncp_inv	0.0	0.0	0.0	
last_pymnt_amnt	8480.98	10021.06	491.03	
next_pymnt_d	0	0	0	
last_credit_pull_d	1	1	0	
pub_rec_bankruptcies	0	0	0	
pred	0	0	0	
pred_proba	0.002519	0.003763	0.005517	
loan_status	1	1	1	

	3	4	5	\
funded_amnt	13000.0	4000.0	21600.0	
funded_amnt_inv	12975.0	4000.0	21575.0	
term	0	0	1	
int_rate	7.66	13.99	14.79	
installment	405.34	136.7	511.49	
grade	0	2	2	
sub_grade	A5	C3	C4	
emp_title	0	0	0	
emp_length	3.0	10.0	10.0	
home_ownership	1	0	0	
annual_inc	66000.0	50400.0	32160.0	
verification_status	1	0	1	
issue_d	69.0	63.966667	61.933333	
desc	0	0	1	
purpose	small_business	debt_consolidation	small_business	
addr_state	OR	NV	NC	
dti	11.02	22.4	0.37	
delinq_2yrs	0.0	0.0	0.0	
earliest_cr_line	124.8	293.233333	197.866667	
fico_range_low	735.0	675.0	795.0	
fico_range_high	739.0	679.0	799.0	
inq_last_6mths	1.0	3.0	2.0	
mths_since_last_delinq	0	0	0	
open_acc	21.0	14.0	2.0	
revol_bal	7683.0	9930.0	115.0	
revol_util	14.7	47.1	11.5	
total_acc	34.0	26.0	10.0	
out_prncp	0.0	0.0	0.0	
out_prncp_inv	0.0	0.0	0.0	
last_pymnt_amnt	405.34	1637.09	11328.71	
next_pymnt_d	0	0	0	
last_credit_pull_d	0	1	1	
pub_rec_bankruptcies	0	0	0	
pred	0	0	0	
pred_proba	0.006075	0.006865	0.007457	
loan_status	1	1	1	

	6	7	8	9
funded_amnt	8000.0	3250.0	3600.0	12000.0
funded_amnt_inv	6747.855405	3166.055906	3600.0	12000.0
term	1	1	1	0
int_rate	6.17	6.91	15.58	9.63
installment	155.3	64.22	86.75	385.13
grade	0	0	3	1
sub_grade	A3	A5	D3	B1

emp_title	0	0	0	0
emp_length	6.0	4.0	5.0	2.0
home_ownership	0	0	1	0
annual_inc	69996.0	35000.0	46000.0	180000.0
verification_status	0	0	0	2
issue_d	72.066667	72.066667	74.1	67.033333
desc	0	0	0	0
purpose	other	debt_consolidation	other	house
addr_state	NC	RI	NY	OR
dti	19.34	21.77	19.57	2.16
delinq_2yrs	0.0	0.0	1.0	0.0
earliest_cr_line	148.133333	198.866667	119.7	100.4
fico_range_low	765.0	725.0	675.0	725.0
fico_range_high	769.0	729.0	679.0	729.0
inq_last_6mths	1.0	0.0	0.0	2.0
mths_since_last_delinq	0	0	0	0
open_acc	15.0	15.0	12.0	11.0
revol_bal	10834.0	20122.0	7826.0	6056.0
revol_util	21.1	46.6	52.2	30.1
total_acc	38.0	50.0	22.0	20.0
out_prncp	0.0	0.0	0.0	0.0
out_prncp_inv	0.0	0.0	0.0	0.0
last_pymnt_amnt	155.3	1468.0	1911.36	385.13
next_pymnt_d	0	0	0	0
last_credit_pull_d	0	1	1	0
pub_rec_bankruptcies	0	0	0	0
pred	0	0	0	0
pred_proba	0.008447	0.008796	0.008897	0.010072
loan_status	1	1	1	1

[183]:
```
ynk.plot_local_breakdown_interactions(top_10_fn, pipeline_explainer)
```

5.2.3 IV 2.3 Top 10 False Positive

[184]:
```
# top 10 false positive
top_10_fp = (X_test
             .query('loan_status != pred and loan_status == 0')
             .sort_values(by='pred_proba', ascending=False)
             .head(10)
             .reset_index(drop=True)
)
top_10_fp.head(10).T
```

[184]:

	0	1	2	\
funded_amnt	20000.0	29600.0	7600.0	
funded_amnt_inv	1510.13	29475.0	7575.0	
term	0	1	1	

int_rate	12.61	19.69	16.77
installment	670.13	779.13	187.95
grade	3	4	3
sub_grade	D1	E5	D2
emp_title	0	0	0
emp_length	3.0	NaN	1.0
home_ownership	1	1	1
annual_inc	38000.0	110000.0	33996.0
verification_status	0	1	1
issue_d	104.5	61.933333	57.866667
desc	0	0	0
purpose	small_business	small_business	other
addr_state	TX	CA	TX
dti	4.04	4.65	16.52
delinq_2yrs	0.0	0.0	0.0
earliest_cr_line	118.766667	160.3	144.033333
fico_range_low	685.0	690.0	685.0
fico_range_high	689.0	694.0	689.0
inq_last_6mths	0.0	0.0	2.0
mths_since_last_delinq	0	0	0
open_acc	3.0	9.0	6.0
revol_bal	4110.0	16498.0	4292.0
revol_util	91.3	58.7	72.7
total_acc	13.0	16.0	21.0
out_prncp	0.0	0.0	0.0
out_prncp_inv	0.0	0.0	0.0
last_pymnt_amnt	678.78	208.28	53.19
next_pymnt_d	0	0	0
last_credit_pull_d	1	1	1
pub_rec_bankruptcies	0	0	0
pred	1	1	1
pred_proba	0.930232	0.928819	0.914703
loan_status	0	0	0

	3	4	5 \
funded_amnt	2825.0	5000.0	2400.0
funded_amnt_inv	1700.0	5000.0	2400.0
term	0	1	1
int_rate	9.64	15.7	17.19
installment	90.68	120.8	59.9
grade	1	3	4
sub_grade	B4	D4	E3
emp_title	0	0	0
emp_length	3.0	0.0	0.0
home_ownership	0	0	1
annual_inc	70000.0	60000.0	12000.0
verification_status	0	0	0

issue_d	108.6	77.166667	73.066667	
desc	0	0	0	
purpose	small_business	home_improvement	car	
addr_state	TN	WA	MI	
dti	18.0	17.28	7.8	
delinq_2yrs	0.0	0.0	0.0	
earliest_cr_line	122.733333	181.6	62.933333	
fico_range_low	730.0	690.0	660.0	
fico_range_high	734.0	694.0	664.0	
inq_last_6mths	2.0	2.0	2.0	
mths_since_last_delinq	0	0	0	
open_acc	11.0	9.0	4.0	
revol_bal	55720.0	5834.0	2630.0	
revol_util	10.0	72.9	50.6	
total_acc	41.0	27.0	5.0	
out_prncp	0.0	0.0	0.0	
out_prncp_inv	0.0	0.0	0.0	
last_pymnt_amnt	96.65	102.68	67.56	
next_pymnt_d	0	0	0	
last_credit_pull_d	1	1	1	
pub_rec_bankruptcies	0	0	0	
pred	1	1	1	
pred_proba	0.889747	0.887084	0.883561	
loan_status	0	0	0	

	6	7	8	\
funded_amnt	2500.0	25000.0	10700.0	
funded_amnt_inv	2489.95523	24804.773795	10700.0	
term	0	1	1	
int_rate	18.78	13.72	17.99	
installment	91.37	578.09	271.66	
grade	5	2	4	
sub_grade	F3	C5	E1	
emp_title	0	0	0	
emp_length	0.0	10.0	1.0	
home_ownership	1	0	1	
annual_inc	65000.0	56000.0	50000.0	
verification_status	0	1	1	
issue_d	83.2	70.033333	60.9	
desc	0	1	0	
purpose	small_business	home_improvement	small_business	
addr_state	MN	GA	WA	
dti	2.79	15.92	10.32	
delinq_2yrs	1.0	0.0	0.0	
earliest_cr_line	124.833333	247.566667	96.4	
fico_range_low	660.0	760.0	685.0	
fico_range_high	664.0	764.0	689.0	

inq_last_6mths	2.0	4.0	1.0
mths_since_last_delinq	0	0	0
open_acc	4.0	5.0	8.0
revol_bal	2601.0	6314.0	8484.0
revol_util	40.6	52.2	94.3
total_acc	7.0	14.0	8.0
out_prncp	0.0	0.0	0.0
out_prncp_inv	0.0	0.0	0.0
last_pymnt_amnt	96.14	577.55	270.83
next_pymnt_d	0	0	0
last_credit_pull_d	1	1	1
pub_rec_bankruptcies	0	0	0
pred	1	1	1
pred_proba	0.8823	0.880714	0.879419
loan_status	0	0	0

	9
funded_amnt	10000.0
funded_amnt_inv	10000.0
term	1
int_rate	17.14
installment	249.28
grade	4
sub_grade	E3
emp_title	0
emp_length	10.0
home_ownership	2
annual_inc	65000.0
verification_status	0
issue_d	67.966667
desc	0
purpose	major_purchase
addr_state	VA
dti	4.67
delinq_2yrs	2.0
earliest_cr_line	164.4
fico_range_low	675.0
fico_range_high	679.0
inq_last_6mths	1.0
mths_since_last_delinq	0
open_acc	17.0
revol_bal	7206.0
revol_util	34.8
total_acc	22.0
out_prncp	0.0
out_prncp_inv	0.0
last_pymnt_amnt	140.29

```
next_pymnt_d                        0
last_credit_pull_d                  1
pub_rec_bankruptcies                0
pred                                1
pred_proba                   0.878064
loan_status                         0
```

[185]:
```
ynk.plot_local_breakdown_interactions(top_10_fp, pipeline_explainer)
```

6 V. Prediction on Holdout Data

[186]:
```
# import the holdout data
holdout = pd.read_csv('./data/loan_holdout.csv')
holdout_id = holdout['id']
```

[187]:
```
# preprocess the holdout data
holdout = dp.feature_engineering_pipeline(holdout)
```

[188]:
```
# predictions on the holdout data using tuned xgboost model
holdout_pred, holdout_pred_prob = ynk.get_predictions(tuned_xgb_pipeline,␣
 ↪holdout)
```

[189]:
```
# create a dataframe for the holdout predictions
holdout_pred_df = pd.DataFrame({
    'id': holdout_id,
    'P_DEFAULT ': holdout_pred_prob
})

# save the holdout predictions to a csv file
holdout_pred_df.to_csv('./data/holdout_predictions.csv', index=False)
```