Interpretable Modelling of Credit Risk

> Daniela Jiménez Lara

> Tianji Rao

As detailed in Cynthia Rudin's excellent commentary on interpretability (ArXiV version here), there are a plethora of reasons to avoid the use of black box models when models are being used to make high stakes decisions to may have life-altering effects on real people. Efforts to develop "explainable black box models," while appealing for their potential to let us continuing using the same tools we always have and to creation explanations after the fact, are inherently flawed. As Rudin notes in my single favorite passage from her paper:

> Explainable ML methods provide explanations that are not faithful to what the original model computes. Explanations must be wrong. They cannot have perfect fidelity with respect to the original model. If the explanation was completely faithful to what the original model computes, the explanation would equal the original model, and one would not need the original model in the first place, only the explanation. (In other words, this is a case where the original model would be interpretable.) This leads to the danger that any explanation method for a black box model can be an inaccurate representation of the original model in parts of the feature space.
>
> An inaccurate (low-fidelity) explanation model limits trust in the explanation, and by extension, trust in the black box that it is trying to explain. An explainable model that has a 90% agreement with the original model indeed explains the original model most of the time. However, an explanation model that is correct 90% of the time is wrong 10% of the time. If a tenth of the explanations are incorrect, one cannot trust the explanations, and thus one cannot trust the original black box. If we cannot know for certain whether our explanation is correct, we cannot know whether to trust either the explanation or the original model.

With this motivation in mind, in this exercise, we will use a cutting edge interpretable modeling framework to model credit risk using data from the 14th Pacific-Asia Knowledge Discovery and Data Mining conference (PAKDD 2010). This data covers the period of 2006 to 2009, and "comes from a private label credit card operation of a Brazilian credit company and its partner shops." (The competition was won by TIMi, who purely by coincidence helped me complete my PhD dissertation research!).

We will be working with Generalized Additive Models (GAMs) (not to be confused with Generalized *Linear* Models (GLMs) — GLMs are a special case of GAMs). In particular, we will be using the pyGAM, though this is far from the only GAM implementation out there. mvgam in R is probably considered the gold standard, as it was developed by a pioneering researcher of GAMs. `statsmodels` also has an implementation, and GAM is also hiding in plain sight behind many other tools, like Meta's Prophet time series forecasting library (which is GAM-based).

In [ ]:
```
# import xlrd
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from pygam import LogisticGAM, s, f
from pygam.datasets import default
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

import warnings

warnings.filterwarnings("ignore")
```

Data Prep

Exercise 1

The PADD 2010 data is in this repository. You can find column names in `PAKDD2010_VariablesList.XLS` and the actual data in `PAKDD2010_Modeling_Data.txt`.

Note: you may run into a string-encoding issue loading the `PAKDD2010_Modeling_Data.txt` data. All I'll say is that most latin-based languages used `latin8` as a text encoding prior to broad adoption of UTF-8. (Don't know about UTF? Check out this video!)

Load the data (including column names).

In [ ]:
```
# column names
columns_file_path = "https://raw.githubusercontent.com/nickeubank/MIDS_Data/master/PAKDD%202010/PAKDD2010_Varia
df_columns = pd.read_excel(columns_file_path, header=None)
column_names = df_columns[1].tolist()
```

In [ ]:
```
# find duplicate
for i in range(len(column_names)):
    if column_names[i] in column_names[:i]:
        print(column_names[i])

# Two EDUCATION_LEVEL
# Keep the first one to be EDUCATION_LEVEL
# Change the seconde one to MATE_EDUCATION_LEVEL

second_edu_level_idx = [
    i for i, value in enumerate(column_names) if value == "EDUCATION_LEVEL"
][-1]

column_names[second_edu_level_idx] = "MATE_EDUCATION_LEVEL"
```

```
EDUCATION_LEVEL
```

In [ ]:
```
# data
data_file_path = "https://media.githubusercontent.com/media/nickeubank/MIDS_Data/master/PAKDD%202010/PAKDD2010_
df_data = pd.read_csv(
    data_file_path, encoding="latin1", sep="\t", header=None, names=column_names[1:]
)
```

In [ ]:
```
df_data.head()
```

Out[ ]:

	ID_CLIENT	CLERK_TYPE	PAYMENT_DAY	APPLICATION_SUBMISSION_TYPE	QUANT_ADDITIONAL_CARDS	POSTAL_ADDRES
0	1	C	5	Web	0	
1	2	C	15	Carga	0	
2	3	C	5	Web	0	
3	4	C	20	Web	0	
4	5	C	10	Web	0	

5 rows × 54 columns

In [ ]:
```
df_columns.head()
```

Out[ ]:

	0	1	2	3
0	Var_Id	Var_Title	Var_Description	Field_Content
1	1	ID_CLIENT	Sequential number for the applicant (to be use...	1-50000, 50001-70000, 70001-90000
2	2	CLERK_TYPE	Not informed	C
3	3	PAYMENT_DAY	Day of the month for bill payment, chosen by t...	1,5,10,15,20,25
4	4	APPLICATION_SUBMISSION_TYPE	Indicates if the application was submitted via...	Web, Carga

Exercise 2

There are a few variables with a lot of missing values (more than half missing). Given the limited documentation for this data it's a little hard to be sure why, but given the effect on sample size and what variables are missing, let's go ahead and drop them. You you end up dropping 6 variables.

Hint: Some variables have missing values that aren't immediately obviously.

(This is not strictly necessary at this stage, given we'll be doing more feature selection down the line, but keeps things easier knowing we don't have to worry about missingness later.)

In [ ]:
```
# df_data.describe()
```

In [ ]:
```
df_data["APPLICATION_SUBMISSION_TYPE"] = df_data["APPLICATION_SUBMISSION_TYPE"].replace(
    "0", np.nan
)
df_data["SEX"] = df_data["SEX"].replace(["N", " "], np.nan)
df_data["MARITAL_STATUS"] = df_data["MARITAL_STATUS"].replace(0, np.nan)
df_data["OCCUPATION_TYPE"] = df_data["OCCUPATION_TYPE"].replace(0.0, np.nan)
df_data["RESIDENCE_TYPE"] = df_data["RESIDENCE_TYPE"].replace(0.0, np.nan)
df_data["RESIDENCIAL_ZIP_3"] = df_data["RESIDENCIAL_ZIP_3"].replace("#DIV/0!", np.nan)
df_data["PROFESSIONAL_STATE"] = df_data["PROFESSIONAL_STATE"].replace(" ", np.nan)
df_data["PROFESSIONAL_PHONE_AREA_CODE"] = df_data[
    "PROFESSIONAL_PHONE_AREA_CODE"
].replace(" ", np.nan)
```

In [ ]:
```
# df_data.isna().sum().sort_values(ascending=False)
```

In [ ]:
```
missing = df_data.isna().mean()
missing_50_col = list(missing[missing > 0.5].index)

missing_50_col
```

Out[ ]:
```
['PROFESSIONAL_STATE',
 'PROFESSIONAL_CITY',
 'PROFESSIONAL_BOROUGH',
 'PROFESSIONAL_PHONE_AREA_CODE',
 'MATE_PROFESSION_CODE',
 'MATE_EDUCATION_LEVEL']
```

In [ ]:
```
# drop
df1 = df_data.drop(columns=missing_50_col)

# df1
```

Exercise 3

Let's start off by fitting a model that uses the following variables:

```
"QUANT_DEPENDANTS",
"QUANT_CARS",
"MONTHS_IN_RESIDENCE",
"PERSONAL_MONTHLY_INCOME",
"QUANT_BANKING_ACCOUNTS",
"AGE",
"SEX",
"MARITAL_STATUS",
"OCCUPATION_TYPE",
"RESIDENCE_TYPE",
"RESIDENCIAL_STATE",
"RESIDENCIAL_CITY",
"RESIDENCIAL_BOROUGH",
"RESIDENCIAL_ZIP_3"
```

(GAMs don't have any automatic feature selection methods, so these are based on my own sense of features that are likely to matter. A fully analysis would entail a few passes at feature refinement)

Plot and otherwise characterize the distributions of all the variables we may use. If you see anything bananas, adjust how terms enter your model. Yes, pyGAM has flexible functional forms, but giving the model features that are engineered to be more substantively meaningful (e.g., taking log of income) will aid model estimation.

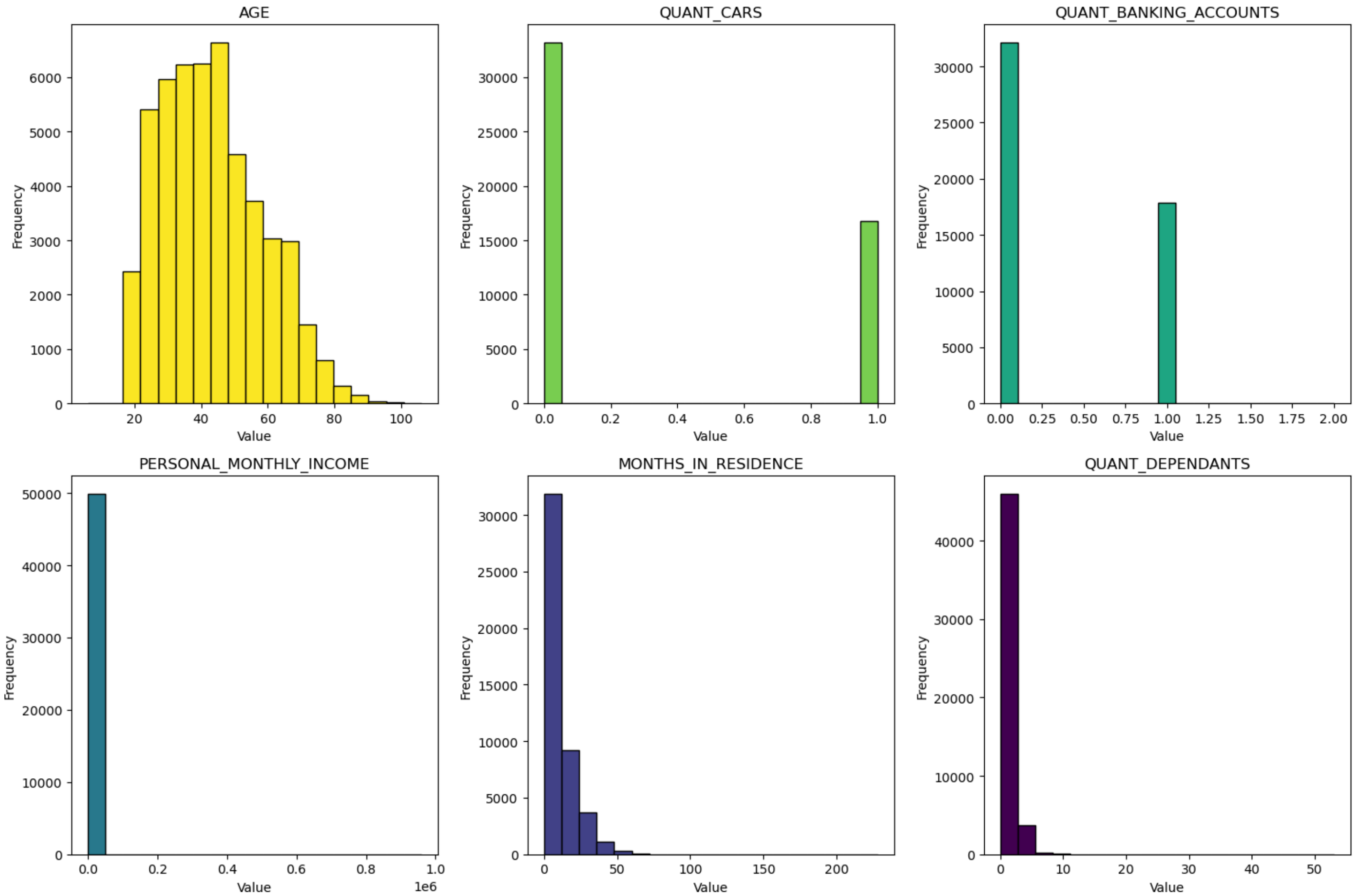
You should probably do something about the functional form of *at least* `PERSONAL_MONTHLY_INCOME`, and `QUANT_DEPENDANTS`.

In [ ]:
```
df1 = df1[
    [
        "QUANT_DEPENDANTS",
        "QUANT_CARS",
        "MONTHS_IN_RESIDENCE",
        "PERSONAL_MONTHLY_INCOME",
        "QUANT_BANKING_ACCOUNTS",
        "AGE",
        "SEX",
        "MARITAL_STATUS",
        "OCCUPATION_TYPE",
        "RESIDENCE_TYPE",
        "RESIDENCIAL_STATE",
        "RESIDENCIAL_CITY",
        "RESIDENCIAL_BOROUGH",
        "RESIDENCIAL_ZIP_3",
        "TARGET_LABEL_BAD=1",
    ]
]
```

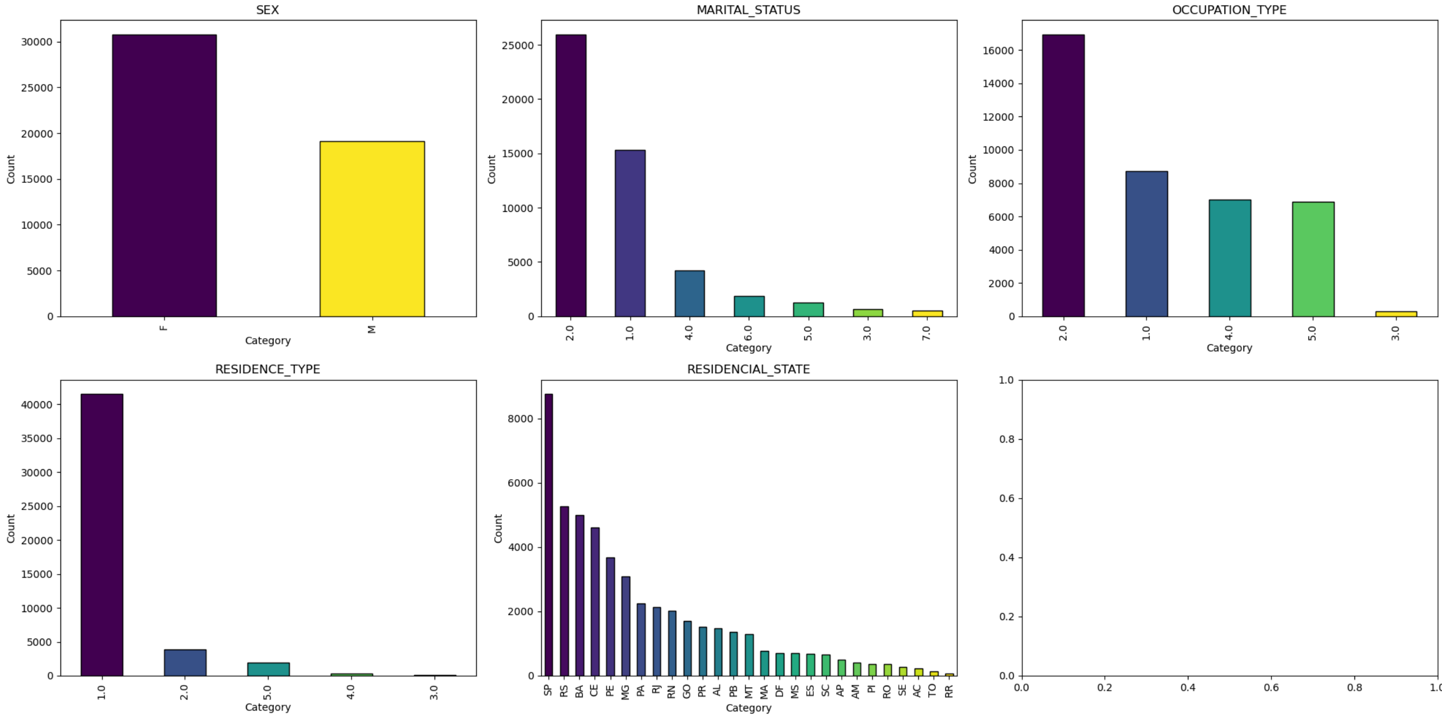
In [ ]:
```
# df1.isna().sum()
```

In [ ]:
```
# df1.info()
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np


quant_vars = [
    "AGE",
    "QUANT_CARS",
    "QUANT_BANKING_ACCOUNTS",
    "PERSONAL_MONTHLY_INCOME",
    "MONTHS_IN_RESIDENCE",
    "QUANT_DEPENDANTS",
]

colors = ["#fde725", "#7ad151", "#22a884", "#2a788e", "#414487", "#440154"]

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

for i, (column, color) in enumerate(zip(quant_vars, colors)):
    ax = axes[i]
    ax.hist(
        df1[column],
        bins=np.linspace(df1[column].min(), df1[column].max(), 20),
        color=color,
        edgecolor="black",
    )
    ax.set_title(column)
    ax.set_xlabel("Value")
    ax.set_ylabel("Frequency")

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



In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

qual_vars2 = [
    "SEX",
    "MARITAL_STATUS",
    "OCCUPATION_TYPE",
    "RESIDENCE_TYPE",
    "RESIDENCIAL_STATE",
    # "RESIDENCIAL_BOROUGH",
]

fig, axes = plt.subplots(2, 3, figsize=(20, 10))

axes = axes.flatten()

cmap = plt.get_cmap("viridis")

for i, var in enumerate(qual_vars2):
    ax = axes[i]
    var_counts = df1[var].value_counts()
    categories = var_counts.index
    colors = cmap(np.linspace(0, 1, len(categories)))

    var_counts.plot(kind="bar", ax=ax, color=colors, edgecolor="black")
    ax.set_title(var)
    ax.set_xlabel("Category")
    ax.set_ylabel("Count")

plt.tight_layout()

plt.show()
```



In [ ]:
```python
# Change to categorical variable
for i in ["MARITAL_STATUS", "OCCUPATION_TYPE", "RESIDENCE_TYPE"]:
    df1[i] = df1[i].astype("Int64").astype("category")

for i in ["SEX", "RESIDENCIAL_STATE", "RESIDENCIAL_CITY", "RESIDENCIAL_ZIP_3"]:
    df1[i] = df1[i].astype("category")
```

In [ ]:
```python
#  functional form of `PERSONAL_MONTHLY_INCOME` and `QUANT_DEPENDANTS`
df1["LOG_PERSONAL_MONTHLY_INCOME"] = df1["PERSONAL_MONTHLY_INCOME"].apply(
    lambda x: np.log(x + 1)
)

df1["LOG_MONTHS_IN_RESIDENCE"] = df1["MONTHS_IN_RESIDENCE"].apply(
    lambda x: np.log(x + 1)
)

# QUANT_DEPENDANTS
df1["QUANT_DEPENDANTS_BINARY"] = df1["QUANT_DEPENDANTS"].apply(
    lambda x: 1 if x > 2 else 0
)
```

In [ ]:
```python
df1.drop(columns=["PERSONAL_MONTHLY_INCOME", "MONTHS_IN_RESIDENCE"], inplace=True)
```

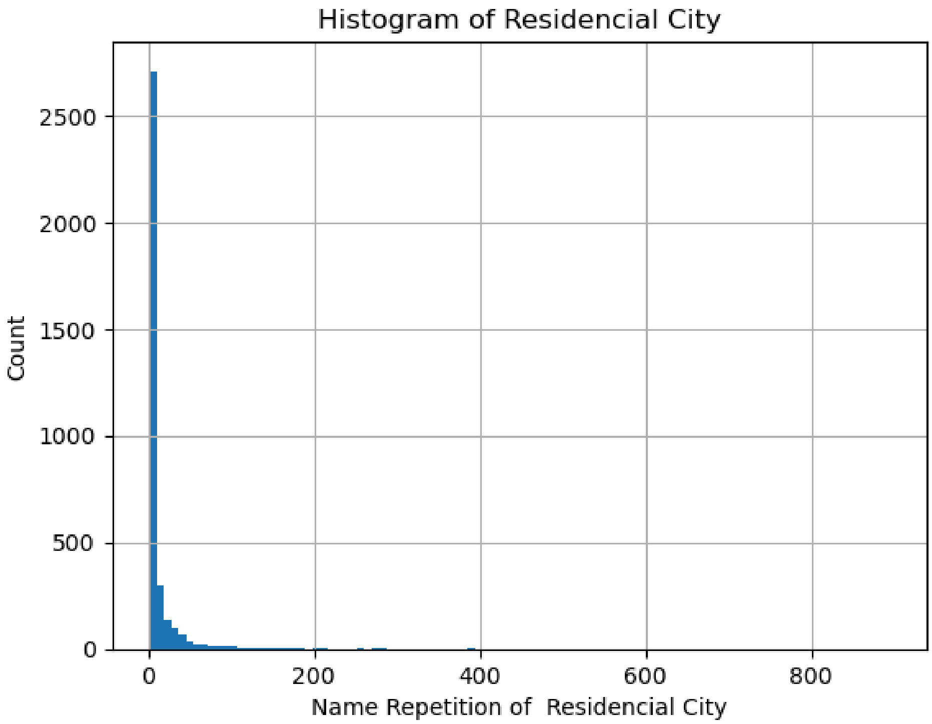
In [ ]:
```python
# df1.info()
```

Exercise 4

Geographic segregation means residency data often contains LOTS of information. But there's a problem with `RESIDENCIAL_CITY` and `RESIDENCIAL_BOROUGH`. What is the problem?

In any real project, this would be something absolutely worth resolving, but for this exercise, we'll just drop all three string `RESIDENCIAL_` variables.

In [ ]:
```python
df1["RESIDENCIAL_CITY"].value_counts().hist(bins=100)
plt.title("Histogram of Residencial City")
plt.xlabel("Name Repetition of  Residencial City")
plt.ylabel("Count")
plt.show()
```



In [ ]:
```
qual_vars = [
    "SEX",
    "MARITAL_STATUS",
    "OCCUPATION_TYPE",
    "RESIDENCE_TYPE",
    "RESIDENCIAL_STATE",
    "RESIDENCIAL_CITY",
    "RESIDENCIAL_BOROUGH",
    "RESIDENCIAL_ZIP_3",
]

for column_name in qual_vars:
    total_unique = df1[column_name].nunique()
    print(
        f"Total number of unique observations in the '{column_name}' variable:",
        total_unique,
    )
```

```
Total number of unique observations in the 'SEX' variable: 2
Total number of unique observations in the 'MARITAL_STATUS' variable: 7
Total number of unique observations in the 'OCCUPATION_TYPE' variable: 5
Total number of unique observations in the 'RESIDENCE_TYPE' variable: 5
Total number of unique observations in the 'RESIDENCIAL_STATE' variable: 27
Total number of unique observations in the 'RESIDENCIAL_CITY' variable: 3529
Total number of unique observations in the 'RESIDENCIAL_BOROUGH' variable: 14511
Total number of unique observations in the 'RESIDENCIAL_ZIP_3' variable: 1480
```

> The total number of unique observations in the 'RESIDENCIAL_CITY' is 3529, and for 'RESIDENCIAL_BOROUGH' is 14511. By doing a review of these unique values, we can see that most of them point to the same borough but with different spelling, use of caps or use of complete name. These lack of consistency on spelling results in different observations in the dataset that refer to the same city or borough.

In [ ]:
```
# drop RESIDENCIAL_ variable
new_df = df1.filter(regex=r"^(?!RESIDENCIAL_).*")
# new_df
```

In [ ]:
```
# drop missing values
new_df = new_df.drop(columns="QUANT_DEPENDANTS").dropna()
```

In [ ]:
```
# solve categorical data
for var in [
    "QUANT_CARS",
```

```
    "QUANT_BANKING_ACCOUNTS",
    "QUANT_DEPENDANTS_BINARY",
]:
    new_df[var] = new_df[var].astype("category")


categorical_variables = [
    "QUANT_CARS",
    "QUANT_BANKING_ACCOUNTS",
    "SEX",
    "MARITAL_STATUS",
    "OCCUPATION_TYPE",
    "RESIDENCE_TYPE",
    "QUANT_DEPENDANTS_BINARY",
]

for var in categorical_variables:
    new_df[var] = new_df[var].cat.codes
```

In [ ]:
```
# new_df.info()
```

Model Fitting

Exercise 5

First, use `train_test_split` to do an 80/20 split of your data. Then, using the `TARGET_LABEL_BAD` variable, fit a classification model on this data. Optimize with `gridsearch`. Use splines for continuous variables and factors for categoricals.

At this point we'd *ideally* be working with 11 variables. However pyGAM can get a little slow with factor features with lots of values + lots of unique values (e.g., 50,000 observations and the *many* values of `RESIDENCIAL_ZIP` takes about 15 minutes on my computer). In that configuration, you should get a model fit in 10-15 seconds.

So let's start by fitting a model that also excludes `RESIDENCIAL_ZIP`.

In [ ]:
```
# Train test split\
X = new_df.loc[:, new_df.columns != "TARGET_LABEL_BAD=1"]
y = new_df["TARGET_LABEL_BAD=1"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

In [ ]:
```
X_train_arr = X_train.to_numpy()
y_train_arr = y_train.to_numpy()

# lambdas
lams = np.logspace(-3, 3, 4)

gam = LogisticGAM(f(0) + f(1) + s(2) + f(3) + f(4) + f(5) + f(6) + s(7) + s(8) + f(9))
gam.gridsearch(X_train_arr, y_train_arr, lam=lams)
```

```
  0% (0 of 4) |                          | Elapsed Time: 0:00:00 ETA:  --:--:--
 25% (1 of 4) |######                    | Elapsed Time: 0:00:01 ETA:   0:00:04
 50% (2 of 4) |#############             | Elapsed Time: 0:00:02 ETA:   0:00:02
 75% (3 of 4) |###################       | Elapsed Time: 0:00:03 ETA:   0:00:01
100% (4 of 4) |##########################| Elapsed Time: 0:00:04 Time:  0:00:04
```

Out[ ]:
```
LogisticGAM(callbacks=[Deviance(), Diffs(), Accuracy()],
   fit_intercept=True, max_iter=100,
   terms=f(0) + f(1) + s(2) + f(3) + f(4) + f(5) + f(6) + s(7) + s(8) + f(9) + intercept,
   tol=0.0001, verbose=False)
```

In [ ]:
```
print("The best lambda (grid search) is:")
print(gam.lam[0][0])
```

```
The best lambda (grid search) is:
10.0
```

In [ ]:
```
y_pred = gam.predict(X_test.to_numpy())
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7404054054054054
```

Exercise 6

Create a (naive) confusion matrix using the predicted values you get with `predict()` on your test data. Our stakeholder cares about two things:

- maximizing the number of people to whom they extend credit, and
- the false negative rate (the share of people identified as "safe bets" who aren't, and who thus default).

How many "good bets" does the model predict (true negatives), and what is the False Omission Rate (the share of predicted negatives that are false negatives)?

Looking at the confusion matrix, how did the model maximize accuracy?

In [ ]:
```
def cm_for(y_test, y_pred, process=True):
    ConfuMat = confusion_matrix(y_test, y_pred)
    TN, FP, FN, TP = ConfuMat.ravel()
    FOR = FN / (FN + TN) if (FN + TN) > 0 else 0
    if process:
        print(f"TN, FP, FN, TP: {TN, FP, FN, TP}")
        print(f"False Omission Rate: {FOR} ")

    return (TN, FP, FN, TP), FOR
```

In [ ]:
```
col, FOR = cm_for(y_test, y_pred)
```

```
TN, FP, FN, TP: (5474, 1, 1920, 5)
False Omission Rate: 0.2596700027048959
```

> There are 5474 "good bets" or True Negatives (people who would not default and thus we would extend credit)

> The False Omission Rate, or predicted negatives that are false negatives, is around 0.2597. Thus the proportion that were 'safe' bets but that would default over all the predicted negatives.

> The model maximizes the accuracy through predicting more people who would not default and minimizing the false positives. In the training set, there are more people whou would not default (TN+FN=5474+1920=7394)than would default, so the model predicts more majority class in the testing period and less minority class (7394 vs. 6).

Exercise 7

Suppose your stakeholder wants to minimize false negative rates. How low of a False Omission Rate (the share of predicted negatives that are false negatives) can you get (assuming more than, say, 10 true negatives), and how many "good bets" (true negatives) do they get at that risk level?

Hint: use `predict_proba()`

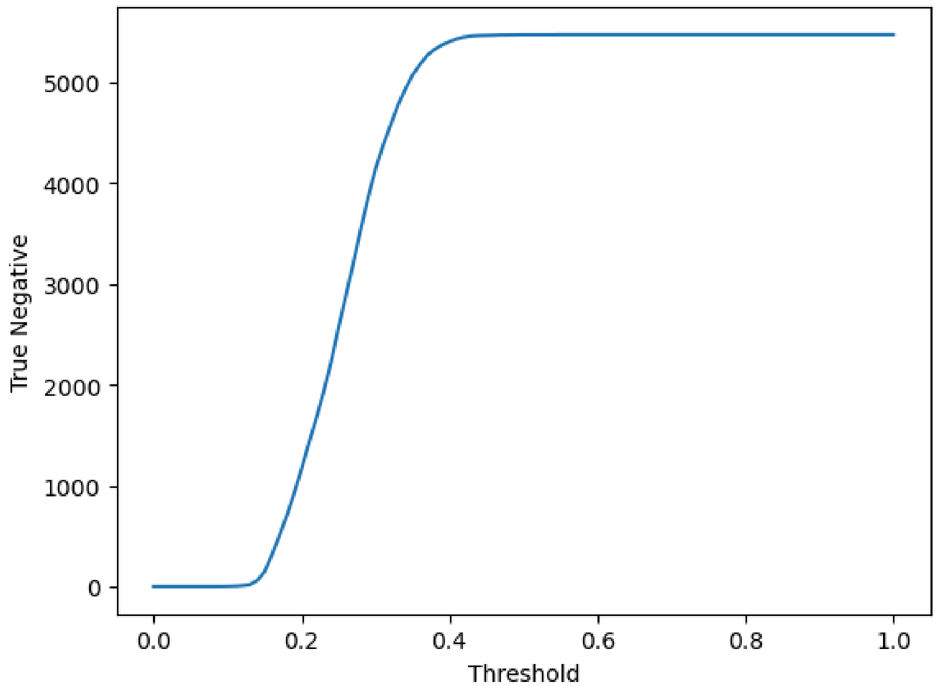
Note: One *can* use class weights to shift the emphasis of the original model fitting, but for the moment let's just play with `predict_proba()` and thresholds.

In [ ]:
```
# predict probability
y_prob = gam.predict_proba(X_test)

# Find the best threshold to minimize FOR and maximize Good Bets
# from 0% to 100%
thre_range = np.linspace(0, 1, 101)
GB = []
FORS = []

for thre in thre_range:
    y_pred_thre = (y_prob > thre).astype("int")

    (TN, FP, FN, TP), FOR = cm_for(y_test, y_pred_thre, process=False)
```

```
    GB.append(TN)
    FORS.append(FOR)
```

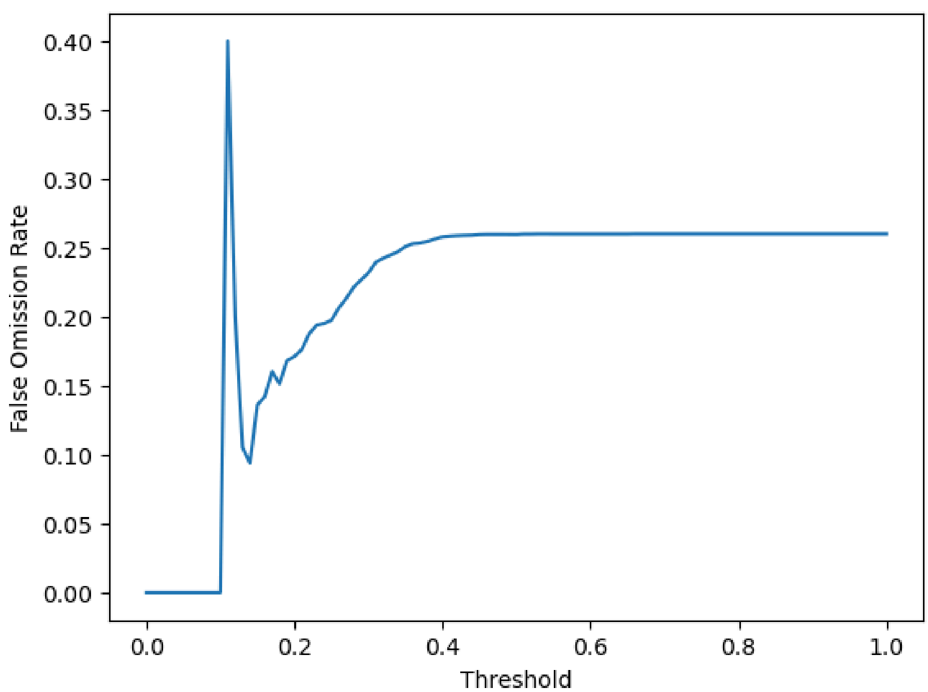
In [ ]:

```
# good bets
plt.plot(thre_range, GB)
plt.xlabel("Threshold")
plt.ylabel("True Negative")
plt.show()
```



In [ ]:

```
plt.plot(thre_range, FORS)
plt.xlabel("Threshold")
plt.ylabel("False Omission Rate")
plt.show()
```



In [ ]:

```
# Find min FOR and corresponding good bets
min_FOR = min(x for x in FORS if x > 0.0)

min_indices = [i for i, x in enumerate(FORS) if x == min_FOR]
```

```
corresponding_GB = [GB[i] for i in min_indices]

print(f"Best threshold: {thre_range[min_indices][0]}")
print(f"Minimum FOR: {min_FOR}")
print(f"Corresponding GB values: {corresponding_GB[0]}")
```

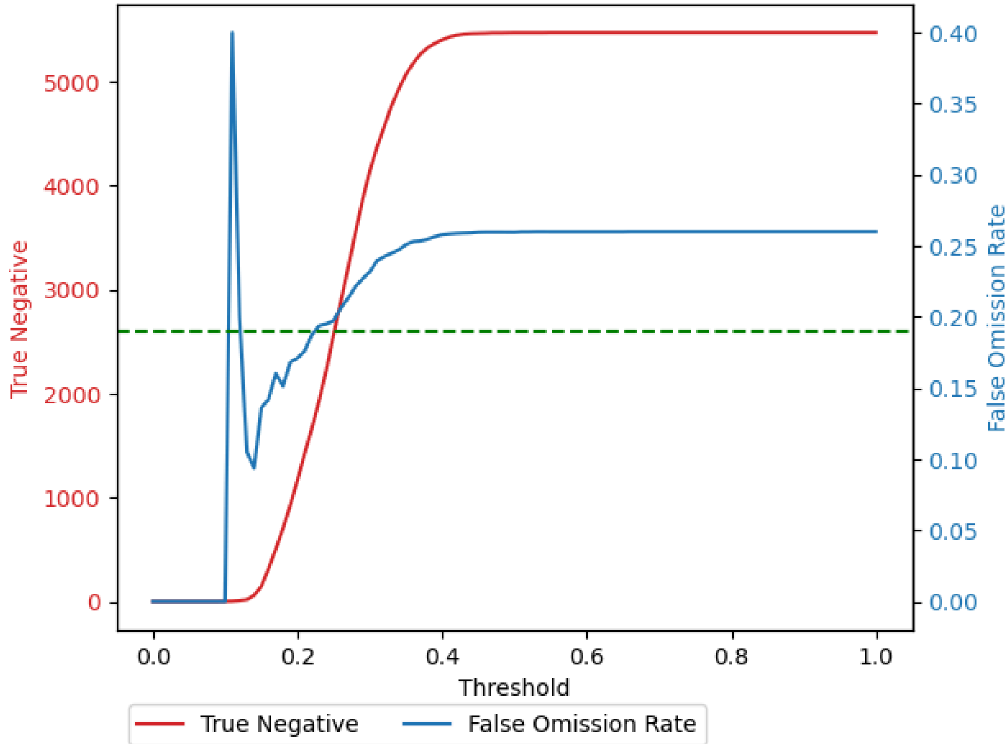
```
Best threshold: 0.14
Minimum FOR: 0.09375
Corresponding GB values: 58
```

> We can minimize the False Omission Rate through changing the threshold value, the best threshold is 0.14. According to the result, the minimum False Omission Rate is 0.09375, where the corresponding "good bets" or True Negatives is 58.

Exercise 8

If the stakeholder wants to maximize true negatives and can tolerate a false omission rate of 19%, how many true negatives will they be able to enroll?

In [ ]:
```
fig, ax1 = plt.subplots()

color = "tab:red"
ax1.set_xlabel("Threshold")
ax1.set_ylabel("True Negative", color=color)
ax1.plot(thre_range, GB, color=color, label="True Negative")
ax1.tick_params(axis="y", labelcolor=color)

ax2 = ax1.twinx()
color = "tab:blue"
ax2.set_ylabel("False Omission Rate", color=color)
ax2.plot(thre_range, FORS, color=color, label="False Omission Rate")
ax2.tick_params(axis="y", labelcolor=color)
ax2.axhline(y=0.19, color="green", linestyle="--")

lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(
    lines + lines2, labels + labels2, loc="upper left", bbox_to_anchor=(0, -0.1), ncol=3
)

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



In [ ]:
```
sele_FOR = [x for x in FORS if x > 0.0 and x < 0.19]

sele_indices = [i for i, x in enumerate(FORS) if x in sele_FOR]

sele_corresponding_GB = [GB[i] for i in sele_indices]

max_sele_gb = max(sele_corresponding_GB)
```

In [ ]:
```
print(f"The max true negatives needed to be enrolled: {max_sele_gb}")
```

```
The max true negatives needed to be enrolled: 1670
```

Let's See This Interpretability!

We're using GAMs for their interpretability, so let's use it!

Exercise 9

Plot the partial dependence plots for all your continuous factors with 95% confidence intervals (I have three, at this stage).

If you get an error like this when generating `partial_dependence` errors:

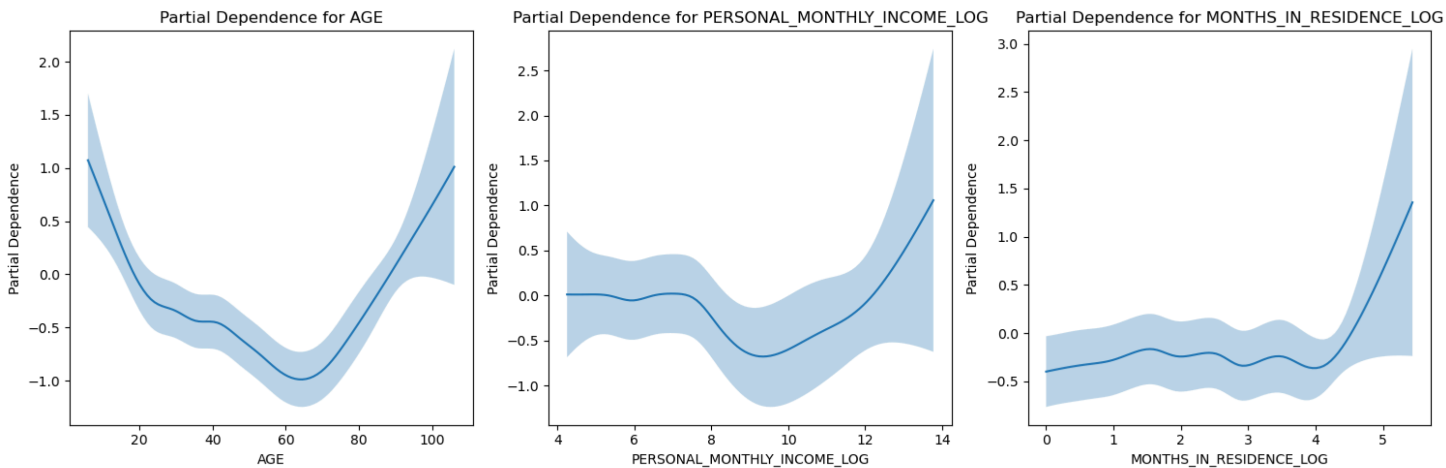
```
----> pdep, confi = gam.partial_dependence(term=i, X=XX, width=0.95)

...
ValueError: X data is out of domain for categorical feature 4. Expected data on [1.0, 2.0], but
found data on [0.0, 0.0]
```

it's because you have a variable set as a factor that doesn't have values of `0`. pyGAM is assuming `0` is the excluded category. Just recode the variable to ensure 0 is used to identify one of the categories.

In [ ]:
```
# check continuous variables
# new_df.info()

# for column in new_df.columns:
#     unique_values = new_df[column].unique()
#     print(f"Column: {column}")
#     print(f"Unique Values: {unique_values}")
#     print()
```

```
# continuous (float)
cont_vars = {"AGE": 2, "PERSONAL_MONTHLY_INCOME_LOG": 7, "MONTHS_IN_RESIDENCE_LOG": 8}
```

In [ ]:
```
# Plot the partial dependence plots
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

for ax, (feature, i) in zip(axs, cont_vars.items()):
    XX = gam.generate_X_grid(term=i)
    pdep, confi = gam.partial_dependence(term=i, X=XX, width=0.95)
    ax.plot(XX[:, i], pdep)
    ax.fill_between(XX[:, i], confi[:, 0], confi[:, 1], alpha=0.3)
    ax.set_xlabel(feature)
    ax.set_ylabel("Partial Dependence")
    ax.set_title(f"Partial Dependence for {feature}")

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



Exercise 10

How does the partial correlation with respect to age look?

> Based on the plot of the partial dependence of age, we can find that the partial correlation is "U-shape". It reveals that as the age increase, the likelihood of default decreases. At the age around 65, the result reaches the lowest likelihood to default. After reaching 65, the trend reverses, and the likelihood of default starts to increase with further increases in age.

Exercise 11

Refit your model, but this time impose monotonicity or concavity/convexity on the relationship between age and credit risk (which makes more sense to you?). Fit the model and plot the new partial dependence.

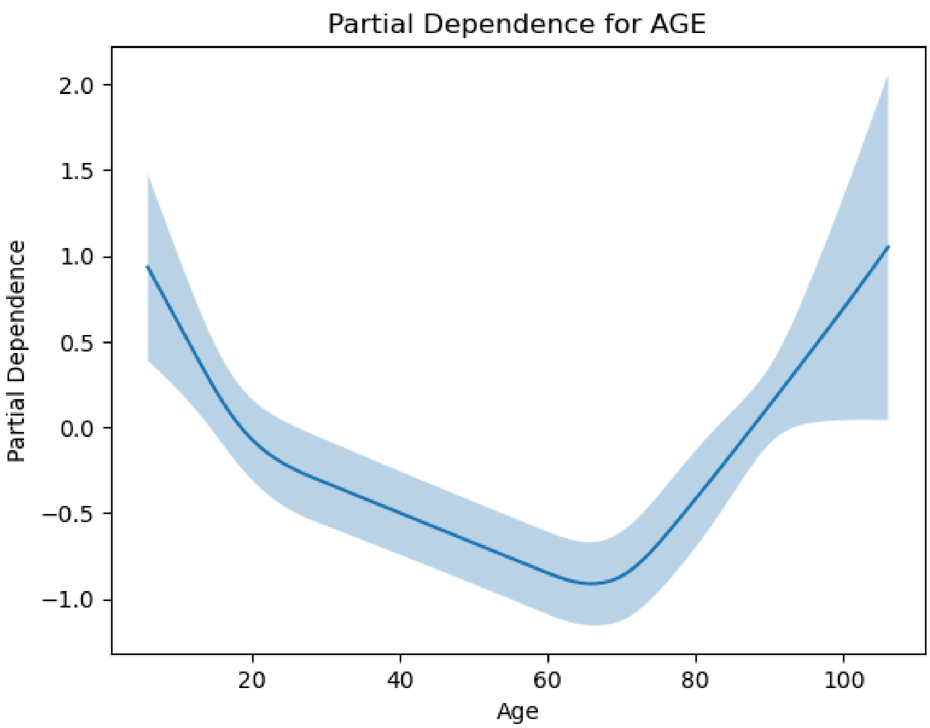
> According to the "U-shape" curve, we should use convexity.

In [ ]:
```
# fit the new model
gam_1 = LogisticGAM(
    f(0)
    + f(1)
    + s(2, constraints="convex")
    + f(3)
    + f(4)
    + f(5)
    + f(6)
    + s(7)
    + s(8)
    + f(9)
)
gam_1.gridsearch(X_train_arr, y_train_arr, lam=lams)
```

```
  0% (0 of 4) |                          | Elapsed Time: 0:00:00 ETA:  --:--:--
 25% (1 of 4) |######                    | Elapsed Time: 0:00:02 ETA:   0:00:08
 50% (2 of 4) |#############             | Elapsed Time: 0:00:03 ETA:   0:00:03
 75% (3 of 4) |###################       | Elapsed Time: 0:00:04 ETA:   0:00:01
100% (4 of 4) |##########################| Elapsed Time: 0:00:05 Time:  0:00:05
```

Out[ ]:
```
LogisticGAM(callbacks=[Deviance(), Diffs(), Accuracy()],
   fit_intercept=True, max_iter=100,
   terms=f(0) + f(1) + s(2) + f(3) + f(4) + f(5) + f(6) + s(7) + s(8) + f(9) + intercept,
   tol=0.0001, verbose=False)
```

In [ ]:
```python
XX = gam.generate_X_grid(term=2)
pdep, confi = gam_1.partial_dependence(term=2, X=XX, width=0.95)
plt.plot(XX[:, 2], pdep)
plt.fill_between(XX[:, 2], confi[:, 0], confi[:, 1], alpha=0.3)
plt.xlabel("Age")
plt.ylabel("Partial Dependence")
plt.title(f"Partial Dependence for AGE")
plt.show()
```



Exercise 12

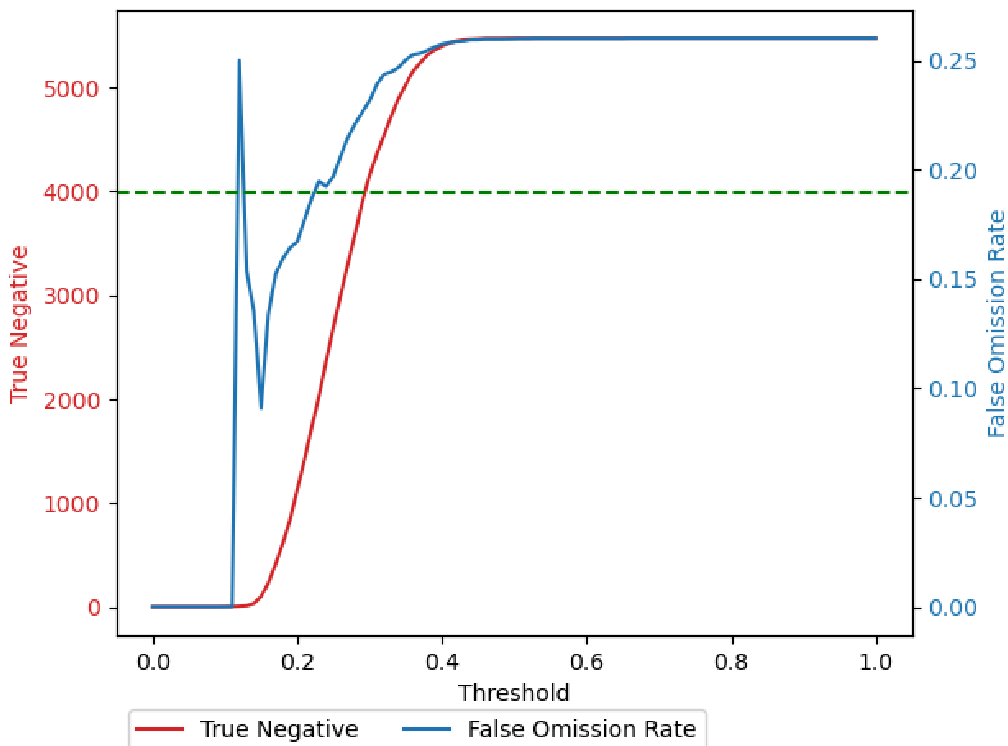
Functional form constraints are often about fairness or meeting regulatory requirements, but they can also prevent overfitting.

Does this change the number of "true negatives" you can enroll below a false omission rate of 19%?

In [ ]:
```python
# predict
y_prob1 = gam_1.predict_proba(X_test)

# Find the best threshold to minimize FOR and maximize Good Bets
# from 0% to 100%
thre_range = np.linspace(0, 1, 101)
GB1 = []
FORS1 = []

for thre in thre_range:
    y_pred_thre = (y_prob1 > thre).astype("int")

    (TN, FP, FN, TP), FOR = cm_for(y_test, y_pred_thre, process=False)

    GB1.append(TN)
    FORS1.append(FOR)
```

In [ ]:

```python
fig, ax1 = plt.subplots()

color = "tab:red"
ax1.set_xlabel("Threshold")
ax1.set_ylabel("True Negative", color=color)
ax1.plot(thre_range, GB1, color=color, label="True Negative")
ax1.tick_params(axis="y", labelcolor=color)

ax2 = ax1.twinx()
color = "tab:blue"
ax2.set_ylabel("False Omission Rate", color=color)
ax2.plot(thre_range, FORS1, color=color, label="False Omission Rate")
ax2.tick_params(axis="y", labelcolor=color)
ax2.axhline(y=0.19, color="green", linestyle="--")

lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(
    lines + lines2, labels + labels2, loc="upper left", bbox_to_anchor=(0, -0.1), ncol=3
)

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



In [ ]:

```python
sele_FOR1 = [x for x in FORS1 if x > 0.0 and x < 0.19]

sele_indices1 = [i for i, x in enumerate(FORS1) if x in sele_FOR1]

sele_corresponding_GB1 = [GB1[i] for i in sele_indices1]

max_sele_gb1 = max(sele_corresponding_GB1)

print(f"The max true negatives needed to be enrolled: {max_sele_gb1}")
```

```
The max true negatives needed to be enrolled: 1727
```

> Yes. The number of "true negatives" we can enroll below FOR of 19% changes from 1670 to 1727.

Exercise 13

In the preceding exercises, we allowed pyGAM to choose its own smoothing parameters / coefficient penalties. This makes life easy, but it isn't always optimal, especially because when it does so, it picks the same smoothing penalty (the `lambda` in `.summary()`) for all terms.

(If you haven't seen them let, penalities are designed to limit overfitting by, basically, "penalizing" big coefficients on different terms. This tends to push models towards smoother fits.)

To get around this, we can do a grid or random search. This is definitely a little slow, but let's give it a try!

Then following the model given in the docs linked above, let's do a random search. Make sure your initial random points has a shape of `100 x (the number of terms in your model)`.

In [ ]:
```
# 10 terms
num_terms = 10
init_lams = 10 ** (np.random.rand(100, num_terms) * 6 - 3)
```

In [ ]:
```
# modeling
loggam = LogisticGAM(
    f(0)
    + f(1)
    + s(2, constraints="convex")
    + f(3)
    + f(4)
    + f(5)
    + f(6)
    + s(7)
    + s(8)
    + f(9)
).gridsearch(X_train_arr, y_train_arr, lam=init_lams)
```

```
  0% (0 of 100) |                        | Elapsed Time: 0:00:00 ETA:  --:--:--
  1% (1 of 100) |                        | Elapsed Time: 0:00:01 ETA:   0:02:24
  2% (2 of 100) |                        | Elapsed Time: 0:00:02 ETA:   0:01:55
  3% (3 of 100) |                        | Elapsed Time: 0:00:03 ETA:   0:01:52
  4% (4 of 100) |                        | Elapsed Time: 0:00:04 ETA:   0:01:48
  5% (5 of 100) |#                       | Elapsed Time: 0:00:05 ETA:   0:01:50
  6% (6 of 100) |#                       | Elapsed Time: 0:00:06 ETA:   0:01:46
  7% (7 of 100) |#                       | Elapsed Time: 0:00:07 ETA:   0:01:45
  8% (8 of 100) |#                       | Elapsed Time: 0:00:08 ETA:   0:01:43
  9% (9 of 100) |##                      | Elapsed Time: 0:00:09 ETA:   0:01:39
 10% (10 of 100) |##                     | Elapsed Time: 0:00:10 ETA:   0:01:37
 11% (11 of 100) |##                     | Elapsed Time: 0:00:11 ETA:   0:01:35
 12% (12 of 100) |##                     | Elapsed Time: 0:00:12 ETA:   0:01:33
 13% (13 of 100) |##                     | Elapsed Time: 0:00:14 ETA:   0:01:34
 14% (14 of 100) |###                    | Elapsed Time: 0:00:15 ETA:   0:01:34
 15% (15 of 100) |###                    | Elapsed Time: 0:00:16 ETA:   0:01:33
 16% (16 of 100) |###                    | Elapsed Time: 0:00:17 ETA:   0:01:34
 17% (17 of 100) |###                    | Elapsed Time: 0:00:18 ETA:   0:01:32
 18% (18 of 100) |####                   | Elapsed Time: 0:00:19 ETA:   0:01:30
 19% (19 of 100) |####                   | Elapsed Time: 0:00:21 ETA:   0:01:32
 20% (20 of 100) |####                   | Elapsed Time: 0:00:22 ETA:   0:01:30
 21% (21 of 100) |####                   | Elapsed Time: 0:00:23 ETA:   0:01:27
 22% (22 of 100) |#####                  | Elapsed Time: 0:00:24 ETA:   0:01:25
 23% (23 of 100) |#####                  | Elapsed Time: 0:00:25 ETA:   0:01:24
 24% (24 of 100) |#####                  | Elapsed Time: 0:00:26 ETA:   0:01:22
 25% (25 of 100) |#####                  | Elapsed Time: 0:00:27 ETA:   0:01:22
 26% (26 of 100) |#####                  | Elapsed Time: 0:00:28 ETA:   0:01:20
 27% (27 of 100) |######                 | Elapsed Time: 0:00:29 ETA:   0:01:19
 28% (28 of 100) |######                 | Elapsed Time: 0:00:30 ETA:   0:01:17
 29% (29 of 100) |######                 | Elapsed Time: 0:00:31 ETA:   0:01:16
 30% (30 of 100) |######                 | Elapsed Time: 0:00:31 ETA:   0:01:14
 31% (31 of 100) |#######                | Elapsed Time: 0:00:32 ETA:   0:01:13
 32% (32 of 100) |#######                | Elapsed Time: 0:00:34 ETA:   0:01:12
 33% (33 of 100) |#######                | Elapsed Time: 0:00:35 ETA:   0:01:11
 34% (34 of 100) |#######                | Elapsed Time: 0:00:36 ETA:   0:01:10
 35% (35 of 100) |########               | Elapsed Time: 0:00:37 ETA:   0:01:09
 36% (36 of 100) |########               | Elapsed Time: 0:00:38 ETA:   0:01:08
 37% (37 of 100) |########               | Elapsed Time: 0:00:39 ETA:   0:01:07
 38% (38 of 100) |########               | Elapsed Time: 0:00:40 ETA:   0:01:05
 39% (39 of 100) |########               | Elapsed Time: 0:00:41 ETA:   0:01:04
 40% (40 of 100) |#########              | Elapsed Time: 0:00:42 ETA:   0:01:04
 41% (41 of 100) |#########              | Elapsed Time: 0:00:43 ETA:   0:01:02
 42% (42 of 100) |#########              | Elapsed Time: 0:00:45 ETA:   0:01:02
 43% (43 of 100) |#########              | Elapsed Time: 0:00:46 ETA:   0:01:01
 44% (44 of 100) |##########             | Elapsed Time: 0:00:47 ETA:   0:00:59
 45% (45 of 100) |##########             | Elapsed Time: 0:00:48 ETA:   0:00:58
 46% (46 of 100) |##########             | Elapsed Time: 0:00:49 ETA:   0:00:57
 47% (47 of 100) |##########             | Elapsed Time: 0:00:50 ETA:   0:00:56
 48% (48 of 100) |###########            | Elapsed Time: 0:00:50 ETA:   0:00:55
 49% (49 of 100) |###########            | Elapsed Time: 0:00:51 ETA:   0:00:53
 50% (50 of 100) |###########            | Elapsed Time: 0:00:52 ETA:   0:00:52
 51% (51 of 100) |###########            | Elapsed Time: 0:00:53 ETA:   0:00:51
 52% (52 of 100) |###########            | Elapsed Time: 0:00:54 ETA:   0:00:49
 53% (53 of 100) |############           | Elapsed Time: 0:00:55 ETA:   0:00:48
 54% (54 of 100) |############           | Elapsed Time: 0:00:56 ETA:   0:00:47
 55% (55 of 100) |############           | Elapsed Time: 0:00:56 ETA:   0:00:46
 56% (56 of 100) |############           | Elapsed Time: 0:00:57 ETA:   0:00:45
 57% (57 of 100) |#############          | Elapsed Time: 0:00:58 ETA:   0:00:44
 58% (58 of 100) |#############          | Elapsed Time: 0:00:59 ETA:   0:00:43
 59% (59 of 100) |#############          | Elapsed Time: 0:01:00 ETA:   0:00:42
 60% (60 of 100) |#############          | Elapsed Time: 0:01:01 ETA:   0:00:41
 61% (61 of 100) |##############         | Elapsed Time: 0:01:03 ETA:   0:00:40
 62% (62 of 100) |##############         | Elapsed Time: 0:01:04 ETA:   0:00:39
 63% (63 of 100) |##############         | Elapsed Time: 0:01:04 ETA:   0:00:38
 64% (64 of 100) |##############         | Elapsed Time: 0:01:05 ETA:   0:00:36
 65% (65 of 100) |##############         | Elapsed Time: 0:01:06 ETA:   0:00:35
 66% (66 of 100) |###############        | Elapsed Time: 0:01:07 ETA:   0:00:34
 67% (67 of 100) |###############        | Elapsed Time: 0:01:08 ETA:   0:00:33
 68% (68 of 100) |###############        | Elapsed Time: 0:01:09 ETA:   0:00:32
 69% (69 of 100) |###############        | Elapsed Time: 0:01:09 ETA:   0:00:31
 70% (70 of 100) |################       | Elapsed Time: 0:01:11 ETA:   0:00:30
 71% (71 of 100) |################       | Elapsed Time: 0:01:12 ETA:   0:00:29
```

```
 72% (72 of 100) |################       | Elapsed Time: 0:01:13 ETA:   0:00:28
 73% (73 of 100) |################       | Elapsed Time: 0:01:14 ETA:   0:00:27
 74% (74 of 100) |#################      | Elapsed Time: 0:01:14 ETA:   0:00:26
 75% (75 of 100) |#################      | Elapsed Time: 0:01:16 ETA:   0:00:25
 76% (76 of 100) |#################      | Elapsed Time: 0:01:17 ETA:   0:00:24
 77% (77 of 100) |#################      | Elapsed Time: 0:01:18 ETA:   0:00:23
 78% (78 of 100) |#################      | Elapsed Time: 0:01:20 ETA:   0:00:22
 79% (79 of 100) |##################     | Elapsed Time: 0:01:21 ETA:   0:00:21
 80% (80 of 100) |##################     | Elapsed Time: 0:01:22 ETA:   0:00:20
 81% (81 of 100) |##################     | Elapsed Time: 0:01:23 ETA:   0:00:19
 82% (82 of 100) |##################     | Elapsed Time: 0:01:24 ETA:   0:00:18
 83% (83 of 100) |###################    | Elapsed Time: 0:01:26 ETA:   0:00:17
 84% (84 of 100) |###################    | Elapsed Time: 0:01:27 ETA:   0:00:16
 85% (85 of 100) |###################    | Elapsed Time: 0:01:28 ETA:   0:00:15
 86% (86 of 100) |###################    | Elapsed Time: 0:01:29 ETA:   0:00:14
 87% (87 of 100) |####################   | Elapsed Time: 0:01:31 ETA:   0:00:13
 88% (88 of 100) |####################   | Elapsed Time: 0:01:33 ETA:   0:00:12
 89% (89 of 100) |####################   | Elapsed Time: 0:01:34 ETA:   0:00:11
 90% (90 of 100) |####################   | Elapsed Time: 0:01:35 ETA:   0:00:10
 91% (91 of 100) |####################   | Elapsed Time: 0:01:36 ETA:   0:00:09
 92% (92 of 100) |#####################  | Elapsed Time: 0:01:39 ETA:   0:00:08
 93% (93 of 100) |#####################  | Elapsed Time: 0:01:40 ETA:   0:00:07
 94% (94 of 100) |#####################  | Elapsed Time: 0:01:41 ETA:   0:00:06
 95% (95 of 100) |#####################  | Elapsed Time: 0:01:42 ETA:   0:00:05
 96% (96 of 100) |###################### | Elapsed Time: 0:01:44 ETA:   0:00:04
 97% (97 of 100) |###################### | Elapsed Time: 0:01:45 ETA:   0:00:03
 98% (98 of 100) |###################### | Elapsed Time: 0:01:46 ETA:   0:00:02
 99% (99 of 100) |###################### | Elapsed Time: 0:01:47 ETA:   0:00:01
100% (100 of 100) |######################| Elapsed Time: 0:01:48 Time:  0:01:48
```

In [ ]:
```python
# model summary
loggam.summary()
```

```
LogisticGAM
=============================================== ==========================================================
Distribution:                      BinomialDist Effective DoF:                                      36.2386
Link Function:                        LogitLink Log Likelihood:                                 -16631.1563
Number of Samples:                        29599 AIC:                                             33334.7899
                                                AICc:                                            33334.8862
                                                UBRE:                                                3.1272
                                                Scale:                                                  1.0
                                                Pseudo R-Squared:                                    0.0185
==========================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
f(0)                              [0.2025]             2            2.0          2.73e-01
f(1)                              [151.0413]           3            0.8          1.64e-01
s(2)                              [0.0015]             20           4.7          0.00e+00     ***
f(3)                              [0.0014]             2            1.0          1.17e-06     ***
f(4)                              [0.0748]             7            6.0          6.75e-12     ***
f(5)                              [0.0491]             5            3.9          2.02e-09     ***
f(6)                              [0.4062]             5            4.0          1.22e-02     *
s(7)                              [55.0576]            20           4.3          4.65e-03     **
s(8)                              [9.9562]             20           8.6          3.25e-03     **
f(9)                              [0.0065]             2            1.0          3.48e-04     ***
intercept                                              1            0.0          4.88e-01
==========================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or models with
         known smoothing parameters, but when smoothing parameters have been estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the null too readily.
```

Exercise 14

How many true negatives can you get now at a less than 19% False Omission Rate?

In [ ]:
```python
def find_tn(model, X_test, threshold):
    y_prob1 = model.predict_proba(X_test)
```

```
    thre_range = np.linspace(0, 1, 101)
    GB1 = []
    FORS1 = []

    for thre in thre_range:
        y_pred_thre = (y_prob1 > thre).astype("int")
        (TN, FP, FN, TP), FOR = cm_for(y_test, y_pred_thre, process=False)
        GB1.append(TN)
        FORS1.append(FOR)

    sele_FOR1 = [x for x in FORS1 if x > 0.0 and x < threshold]
    sele_indices1 = [i for i, x in enumerate(FORS1) if x in sele_FOR1]
    sele_corresponding_GB1 = [GB1[i] for i in sele_indices1]
    max_sele_gb1 = max(sele_corresponding_GB1)
    print(f"The max true negatives needed to be enrolled: {max_sele_gb1}")
```

In [ ]:
```
find_tn(loggam, X_test, 0.19)
```

```
The max true negatives needed to be enrolled: 1717
```

Exercise 15

Add an interaction term between age and personal income.

In [ ]:
```
from pygam import LogisticGAM, s, te

# 12 terms
num_terms = 12
init_lams = 10 ** (np.random.rand(100, num_terms) * 6 - 3)


# fit interaction
gam_2 = LogisticGAM(
    f(0)
    + f(1)
    + s(2, constraints="convex")
    + f(3)
    + f(4)
    + f(5)
    + f(6)
    + s(7)
    + s(8)
    + f(9)
    + te(2, 7)
)
gam_2.gridsearch(X_train_arr, y_train_arr, lam=init_lams)
```

```
 0% (0 of 100) |                        | Elapsed Time: 0:00:00 ETA:  --:--:--
 1% (1 of 100) |                        | Elapsed Time: 0:00:03 ETA:   0:06:03
 2% (2 of 100) |                        | Elapsed Time: 0:00:06 ETA:   0:05:09
 3% (3 of 100) |                        | Elapsed Time: 0:00:09 ETA:   0:05:14
 4% (4 of 100) |                        | Elapsed Time: 0:00:12 ETA:   0:04:58
 5% (5 of 100) |#                       | Elapsed Time: 0:00:16 ETA:   0:05:07
 6% (6 of 100) |#                       | Elapsed Time: 0:00:18 ETA:   0:04:54
 7% (7 of 100) |#                       | Elapsed Time: 0:00:22 ETA:   0:04:58
 8% (8 of 100) |#                       | Elapsed Time: 0:00:24 ETA:   0:04:46
 9% (9 of 100) |##                      | Elapsed Time: 0:00:28 ETA:   0:04:44
10% (10 of 100) |##                     | Elapsed Time: 0:00:30 ETA:   0:04:36
11% (11 of 100) |##                     | Elapsed Time: 0:00:32 ETA:   0:04:26
12% (12 of 100) |##                     | Elapsed Time: 0:00:35 ETA:   0:04:20
13% (13 of 100) |##                     | Elapsed Time: 0:00:38 ETA:   0:04:15
14% (14 of 100) |###                    | Elapsed Time: 0:00:41 ETA:   0:04:14
15% (15 of 100) |###                    | Elapsed Time: 0:00:44 ETA:   0:04:11
16% (16 of 100) |###                    | Elapsed Time: 0:00:47 ETA:   0:04:09
17% (17 of 100) |###                    | Elapsed Time: 0:00:50 ETA:   0:04:07
18% (18 of 100) |####                   | Elapsed Time: 0:00:52 ETA:   0:04:00
19% (19 of 100) |####                   | Elapsed Time: 0:00:55 ETA:   0:03:54
20% (20 of 100) |####                   | Elapsed Time: 0:00:57 ETA:   0:03:49
21% (21 of 100) |####                   | Elapsed Time: 0:01:01 ETA:   0:03:49
22% (22 of 100) |#####                  | Elapsed Time: 0:01:03 ETA:   0:03:45
23% (23 of 100) |#####                  | Elapsed Time: 0:01:06 ETA:   0:03:43
24% (24 of 100) |#####                  | Elapsed Time: 0:01:09 ETA:   0:03:41
25% (25 of 100) |#####                  | Elapsed Time: 0:01:12 ETA:   0:03:36
26% (26 of 100) |#####                  | Elapsed Time: 0:01:14 ETA:   0:03:33
27% (27 of 100) |######                 | Elapsed Time: 0:01:17 ETA:   0:03:29
28% (28 of 100) |######                 | Elapsed Time: 0:01:19 ETA:   0:03:24
29% (29 of 100) |######                 | Elapsed Time: 0:01:22 ETA:   0:03:22
30% (30 of 100) |######                 | Elapsed Time: 0:01:24 ETA:   0:03:18
31% (31 of 100) |#######                | Elapsed Time: 0:01:28 ETA:   0:03:16
32% (32 of 100) |#######                | Elapsed Time: 0:01:30 ETA:   0:03:12
33% (33 of 100) |#######                | Elapsed Time: 0:01:33 ETA:   0:03:10
34% (34 of 100) |#######                | Elapsed Time: 0:01:36 ETA:   0:03:06
35% (35 of 100) |########               | Elapsed Time: 0:01:39 ETA:   0:03:04
36% (36 of 100) |########               | Elapsed Time: 0:01:41 ETA:   0:03:00
37% (37 of 100) |########               | Elapsed Time: 0:01:44 ETA:   0:02:57
38% (38 of 100) |########               | Elapsed Time: 0:01:47 ETA:   0:02:56
39% (39 of 100) |########               | Elapsed Time: 0:01:50 ETA:   0:02:53
40% (40 of 100) |#########              | Elapsed Time: 0:01:54 ETA:   0:02:51
41% (41 of 100) |#########              | Elapsed Time: 0:01:56 ETA:   0:02:47
42% (42 of 100) |#########              | Elapsed Time: 0:01:58 ETA:   0:02:44
43% (43 of 100) |#########              | Elapsed Time: 0:02:01 ETA:   0:02:40
44% (44 of 100) |##########             | Elapsed Time: 0:02:04 ETA:   0:02:39
45% (45 of 100) |##########             | Elapsed Time: 0:02:08 ETA:   0:02:36
46% (46 of 100) |##########             | Elapsed Time: 0:02:13 ETA:   0:02:36
47% (47 of 100) |##########             | Elapsed Time: 0:02:16 ETA:   0:02:33
48% (48 of 100) |###########            | Elapsed Time: 0:02:18 ETA:   0:02:30
49% (49 of 100) |###########            | Elapsed Time: 0:02:21 ETA:   0:02:27
50% (50 of 100) |###########            | Elapsed Time: 0:02:24 ETA:   0:02:24
51% (51 of 100) |###########            | Elapsed Time: 0:02:27 ETA:   0:02:21
52% (52 of 100) |###########            | Elapsed Time: 0:02:31 ETA:   0:02:19
53% (53 of 100) |############           | Elapsed Time: 0:02:34 ETA:   0:02:16
54% (54 of 100) |############           | Elapsed Time: 0:02:37 ETA:   0:02:14
55% (55 of 100) |############           | Elapsed Time: 0:02:39 ETA:   0:02:10
56% (56 of 100) |############           | Elapsed Time: 0:02:42 ETA:   0:02:07
57% (57 of 100) |#############          | Elapsed Time: 0:02:46 ETA:   0:02:05
58% (58 of 100) |#############          | Elapsed Time: 0:02:48 ETA:   0:02:02
59% (59 of 100) |#############          | Elapsed Time: 0:02:51 ETA:   0:01:58
60% (60 of 100) |#############          | Elapsed Time: 0:02:53 ETA:   0:01:55
61% (61 of 100) |##############         | Elapsed Time: 0:02:56 ETA:   0:01:52
62% (62 of 100) |##############         | Elapsed Time: 0:02:59 ETA:   0:01:50
63% (63 of 100) |##############         | Elapsed Time: 0:03:02 ETA:   0:01:47
64% (64 of 100) |##############         | Elapsed Time: 0:03:05 ETA:   0:01:44
65% (65 of 100) |##############         | Elapsed Time: 0:03:07 ETA:   0:01:41
66% (66 of 100) |###############        | Elapsed Time: 0:03:11 ETA:   0:01:38
67% (67 of 100) |###############        | Elapsed Time: 0:03:14 ETA:   0:01:35
68% (68 of 100) |###############        | Elapsed Time: 0:03:17 ETA:   0:01:32
69% (69 of 100) |###############        | Elapsed Time: 0:03:19 ETA:   0:01:29
70% (70 of 100) |################       | Elapsed Time: 0:03:22 ETA:   0:01:26
71% (71 of 100) |################       | Elapsed Time: 0:03:24 ETA:   0:01:23
```

```
 72% (72 of 100) |################       | Elapsed Time: 0:03:26 ETA:   0:01:20
 73% (73 of 100) |################       | Elapsed Time: 0:03:29 ETA:   0:01:17
 74% (74 of 100) |#################      | Elapsed Time: 0:03:31 ETA:   0:01:14
 75% (75 of 100) |#################      | Elapsed Time: 0:03:34 ETA:   0:01:11
 76% (76 of 100) |#################      | Elapsed Time: 0:03:37 ETA:   0:01:08
 77% (77 of 100) |#################      | Elapsed Time: 0:03:40 ETA:   0:01:05
 78% (78 of 100) |#################      | Elapsed Time: 0:03:43 ETA:   0:01:03
 79% (79 of 100) |##################     | Elapsed Time: 0:03:45 ETA:   0:01:00
 80% (80 of 100) |##################     | Elapsed Time: 0:03:48 ETA:   0:00:57
 81% (81 of 100) |##################     | Elapsed Time: 0:03:51 ETA:   0:00:54
 82% (82 of 100) |##################     | Elapsed Time: 0:03:56 ETA:   0:00:51
 83% (83 of 100) |###################    | Elapsed Time: 0:03:58 ETA:   0:00:48
 84% (84 of 100) |###################    | Elapsed Time: 0:04:00 ETA:   0:00:45
 85% (85 of 100) |###################    | Elapsed Time: 0:04:03 ETA:   0:00:42
 86% (86 of 100) |###################    | Elapsed Time: 0:04:06 ETA:   0:00:40
 87% (87 of 100) |####################   | Elapsed Time: 0:04:09 ETA:   0:00:37
 88% (88 of 100) |####################   | Elapsed Time: 0:04:11 ETA:   0:00:34
 89% (89 of 100) |####################   | Elapsed Time: 0:04:14 ETA:   0:00:31
 90% (90 of 100) |####################   | Elapsed Time: 0:04:17 ETA:   0:00:28
 91% (91 of 100) |####################   | Elapsed Time: 0:04:19 ETA:   0:00:25
 92% (92 of 100) |#####################  | Elapsed Time: 0:04:23 ETA:   0:00:22
 93% (93 of 100) |#####################  | Elapsed Time: 0:04:27 ETA:   0:00:20
 94% (94 of 100) |#####################  | Elapsed Time: 0:04:30 ETA:   0:00:17
 95% (95 of 100) |#####################  | Elapsed Time: 0:04:33 ETA:   0:00:14
 96% (96 of 100) |###################### | Elapsed Time: 0:04:37 ETA:   0:00:11
 97% (97 of 100) |###################### | Elapsed Time: 0:04:41 ETA:   0:00:08
 98% (98 of 100) |###################### | Elapsed Time: 0:04:44 ETA:   0:00:05
 99% (99 of 100) |###################### | Elapsed Time: 0:04:47 ETA:   0:00:02
100% (100 of 100) |######################| Elapsed Time: 0:04:49 Time:  0:04:49
```

Out[ ]:
```
LogisticGAM(callbacks=[Deviance(), Diffs(), Accuracy()],
   fit_intercept=True, max_iter=100,
   terms=f(0) + f(1) + s(2) + f(3) + f(4) + f(5) + f(6) + s(7) + s(8) + f(9) + te(2, 7) + intercept,
   tol=0.0001, verbose=False)
```

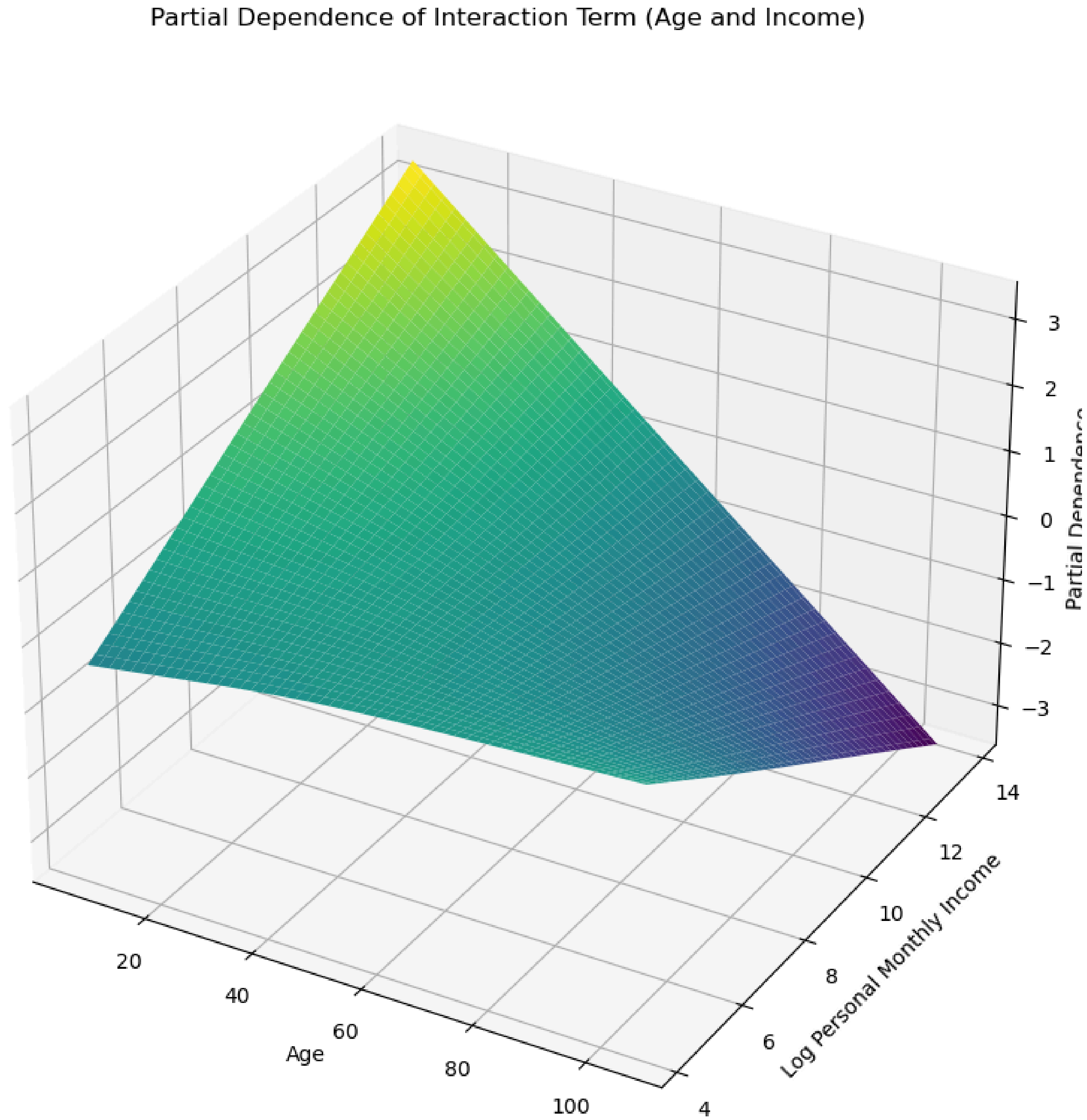
In [ ]:
```
# model summary
gam_2.summary()
```

```
LogisticGAM
=============================================== ==========================================================
Distribution:                      BinomialDist Effective DoF:                                     32.8102
Link Function:                        LogitLink Log Likelihood:                               -16622.4071
Number of Samples:                        29599 AIC:                                           33310.4347
                                                AICc:                                          33310.5143
                                                UBRE:                                              3.1263
                                                Scale:                                                1.0
                                                Pseudo R-Squared:                                   0.019
==========================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code   
================================= ==================== ============ ============ ============ ============
f(0)                              [0.0015]             2            2.0          4.35e-01                 
f(1)                              [0.1617]             3            2.0          1.79e-01                 
s(2)                              [0.8744]             20           5.3          0.00e+00     ***         
f(3)                              [5.0818]             2            1.0          9.24e-07     ***         
f(4)                              [148.2355]           7            3.5          1.06e-12     ***         
f(5)                              [0.008]              5            4.0          1.85e-09     ***         
f(6)                              [0.0013]             5            4.0          3.03e-02     *           
s(7)                              [34.0676]            20           4.9          1.34e-03     **          
s(8)                              [523.0063]           20           4.0          2.71e-03     **          
f(9)                              [1.473]              2            1.0          6.64e-04     ***         
te(2, 7)                          [44.91   38.8613]    100          1.2          1.26e-05     ***         
intercept                                              1            0.0          3.90e-01                 
==========================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or models with
         known smoothing parameters, but when smoothing parameters have been estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the null too readily.
```

Exercise 16

Now visualize the partial dependence interaction term.

In [ ]:
```
plt.rcParams["figure.figsize"] = (12, 8)
XX = gam_2.generate_X_grid(term=10, meshgrid=True)
Z = gam_2.partial_dependence(term=10, X=XX, meshgrid=True)
ax = plt.axes(projection="3d")
ax.plot_surface(XX[0], XX[1], Z, cmap="viridis")
ax.set_ylabel("Log Personal Monthly Income")
ax.set_xlabel("Age")
ax.set_zlabel("Partial Dependence", labelpad=1)
ax.set_title("Partial Dependence of Interaction Term (Age and Income)")
plt.tight_layout()
plt.show()
```



Exercise 17

Finally, another popular interpretable model is the `ExplainableBoostingClassifier`. You can learn more about it here, though how much sense it will make to you may be limited if you aren't familiar with gradient boosting yet. Still, at least one of your classmates prefers it to pyGAM, so give it a try using this code:

```
from interpret.glassbox import ExplainableBoostingClassifier
from interpret import show
import warnings
```

```python
ebm = ExplainableBoostingClassifier()
ebm.fit(X_train, y_train)

with warnings.catch_warnings():
   warnings.simplefilter("ignore")

   ebm_global = ebm.explain_global()
   show(ebm_global)

   ebm_local = ebm.explain_local(X_train, y_train)
   show(ebm_local)
```

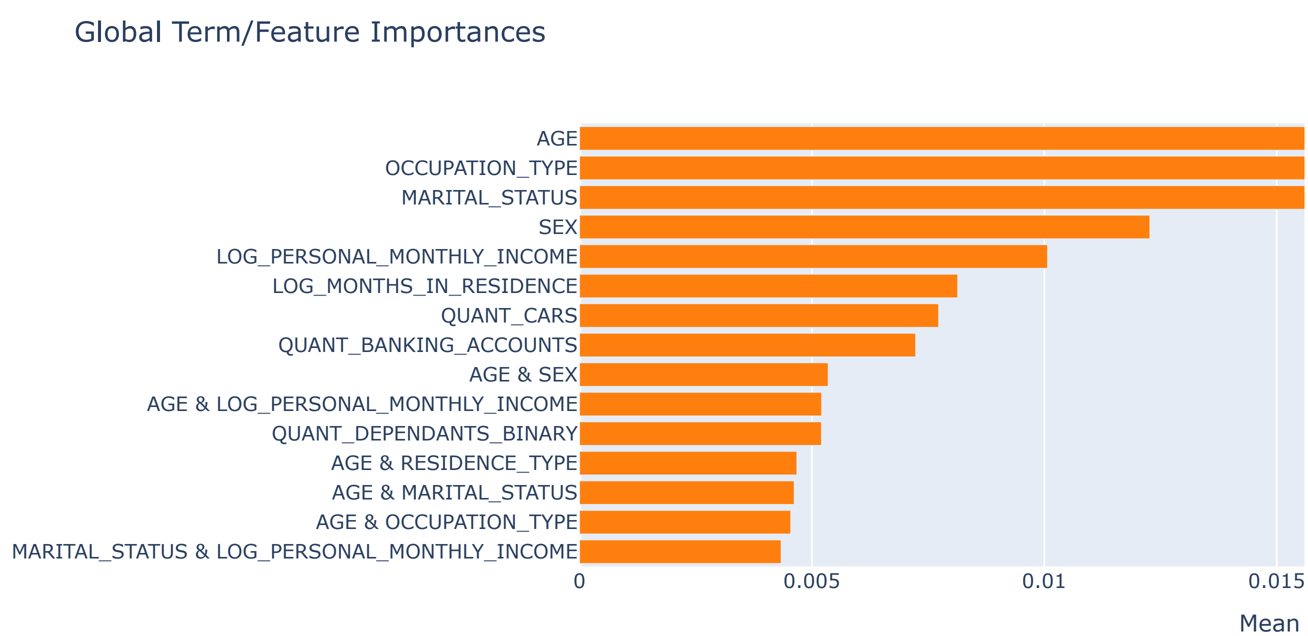
In [ ]:

```python
from interpret.glassbox import ExplainableBoostingClassifier
from interpret import show
import warnings

ebm = ExplainableBoostingClassifier()
ebm.fit(X_train, y_train)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    ebm_global = ebm.explain_global()
    show(ebm_global)

    ebm_local = ebm.explain_local(X_train, y_train)
    show(ebm_local)
```

Select Component to Graph

Summary

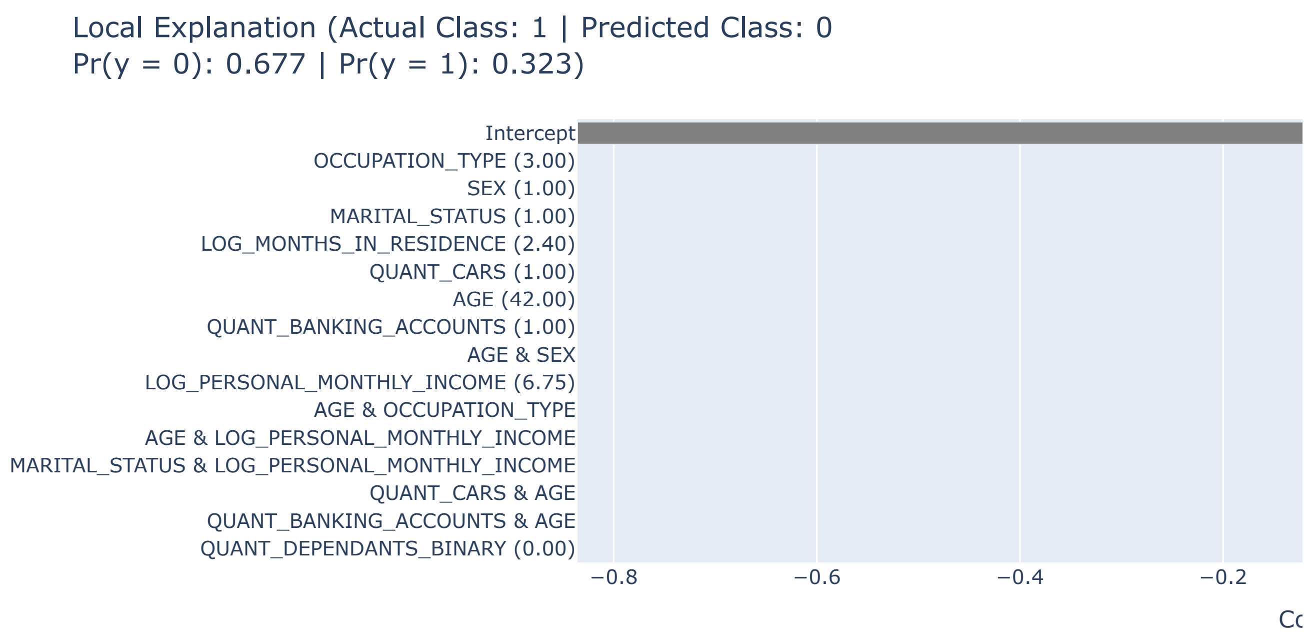
ExplainableBoostingClassifier_0 (Overall)



Select Component to Graph

0 : Actual (1) | Predicted (0) | PrScore (0.677)

ExplainableBoostingClassifier_1 [0]



In [ ]: