index

May 23, 2021

1 Data Science Project

- Name: Author Name
- Email:

1.1 TABLE OF CONTENTS

2 INTRODUCTION

Explain the point of your project and what question you are trying to answer with your modeling.

[1]:
```
from distutils.util import strtobool
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import ticker
import seaborn as sns
import missingno as msno

sns.set_theme(font_scale=1, style='darkgrid')
sns.set_palette("deep", desat=0.85, color_codes=True)
%matplotlib inline
```

[2]:
```
%load_ext autoreload
%autoreload 2
from tools import cleaning, outliers, plotting, utils
from tools.modeling import diagnostics
```

3 OBTAIN

[3]:
```
df = pd.read_csv("data/bank-additional-full.csv", sep=";")
df.head()
```

[3]:
```
   age        job  marital    education  default housing loan    contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  month day_of_week  …  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  …         1    999         0  nonexistent          1.1
1   may         mon  …         1    999         0  nonexistent          1.1
2   may         mon  …         1    999         0  nonexistent          1.1
3   may         mon  …         1    999         0  nonexistent          1.1
4   may         mon  …         1    999         0  nonexistent          1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
```

```
 13   previous          41188 non-null   int64
 14   poutcome          41188 non-null   object
 15   emp.var.rate      41188 non-null   float64
 16   cons.price.idx    41188 non-null   float64
 17   cons.conf.idx     41188 non-null   float64
 18   euribor3m         41188 non-null   float64
 19   nr.employed       41188 non-null   float64
 20   y                 41188 non-null   object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

[5]:
```
df.nunique()
```

[5]:
```
age                 78
job                 12
marital              4
education            8
default              3
housing              3
loan                 3
contact              2
month               10
day_of_week          5
duration          1544
campaign            42
pdays               27
previous             8
poutcome             3
emp.var.rate        10
cons.price.idx      26
cons.conf.idx       26
euribor3m          316
nr.employed         11
y                    2
dtype: int64
```

4 SCRUB

I rename some features to make them a little easier to interpret. Every variable prefixed with "contact" has to do with the last contact of the current campaign.

[6]:
```
df.columns = df.columns.str.replace(".", "_", regex=False)
rename = {"y": "invested",
        "poutcome": "prev_outcome",
        "pdays": "days_since_prev",
        "previous": "prev_contact_count",
        "campaign": "contact_count",
```

```python
        "month": "contact_month",
        "day_of_week": "contact_weekday",
        "duration": "contact_duration",
        "contact": "contact_type",
        "nr_employed": "n_employed",
        "euribor3m": "euribor_3m"}
df.rename(columns=rename, inplace=True)
del rename
df.columns
```

[6]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact_type', 'contact_month', 'contact_weekday', 'contact_duration',
       'contact_count', 'days_since_prev', 'prev_contact_count',
       'prev_outcome', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx',
       'euribor_3m', 'n_employed', 'invested'],
      dtype='object')
```

[7]:
```python
df["days_since_prev"].replace(999, np.NaN, inplace=True)
df.replace(["unknown", "nonexistent"], np.NaN, inplace=True)
cleaning.info(df)
```

[7]:

	nan	nan_%	uniq	uniq_%	dup	dup_%
days_since_prev	39673	96.32	26	0.06	12	0.03
prev_outcome	35563	86.34	2	0.00	12	0.03
default	8597	20.87	2	0.00	12	0.03
education	1731	4.20	7	0.02	12	0.03
housing	990	2.40	2	0.00	12	0.03
loan	990	2.40	2	0.00	12	0.03
job	330	0.80	11	0.03	12	0.03
marital	80	0.19	3	0.01	12	0.03
age	0	0.00	78	0.19	12	0.03
n_employed	0	0.00	11	0.03	12	0.03
euribor_3m	0	0.00	316	0.77	12	0.03
cons_conf_idx	0	0.00	26	0.06	12	0.03
cons_price_idx	0	0.00	26	0.06	12	0.03
emp_var_rate	0	0.00	10	0.02	12	0.03
contact_duration	0	0.00	1544	3.75	12	0.03
prev_contact_count	0	0.00	8	0.02	12	0.03
contact_count	0	0.00	42	0.10	12	0.03
contact_weekday	0	0.00	5	0.01	12	0.03
contact_month	0	0.00	10	0.02	12	0.03
contact_type	0	0.00	2	0.00	12	0.03
invested	0	0.00	2	0.00	12	0.03

[8]:
```python
display(df.loc[df.duplicated()])
df.drop_duplicates(inplace=True)
```

	age	job	marital	education	default	housing	loan	\
1266	39	blue-collar	married	basic.6y	no	no	no	
12261	36	retired	married	NaN	no	no	no	
14234	27	technician	single	professional.course	no	no	no	
16956	47	technician	divorced	high.school	no	yes	no	
18465	32	technician	single	professional.course	no	yes	no	
20216	55	services	married	high.school	NaN	no	no	
20534	41	technician	married	professional.course	no	yes	no	
25217	39	admin.	married	university.degree	no	no	no	
28477	24	services	single	high.school	no	yes	no	
32516	35	admin.	married	university.degree	no	yes	no	
36951	45	admin.	married	university.degree	no	no	no	
38281	71	retired	single	university.degree	no	no	no	

	contact_type	contact_month	contact_weekday	…	contact_count	\
1266	telephone	may	thu	…	1	
12261	telephone	jul	thu	…	1	
14234	cellular	jul	mon	…	2	
16956	cellular	jul	thu	…	3	
18465	cellular	jul	thu	…	1	
20216	cellular	aug	mon	…	1	
20534	cellular	aug	tue	…	1	
25217	cellular	nov	tue	…	2	
28477	cellular	apr	tue	…	1	
32516	cellular	may	fri	…	4	
36951	cellular	jul	thu	…	1	
38281	telephone	oct	tue	…	1	

	days_since_prev	prev_contact_count	prev_outcome	emp_var_rate	\
1266	NaN	0	NaN	1.1	
12261	NaN	0	NaN	1.4	
14234	NaN	0	NaN	1.4	
16956	NaN	0	NaN	1.4	
18465	NaN	0	NaN	1.4	
20216	NaN	0	NaN	1.4	
20534	NaN	0	NaN	1.4	
25217	NaN	0	NaN	-0.1	
28477	NaN	0	NaN	-1.8	
32516	NaN	0	NaN	-1.8	
36951	NaN	0	NaN	-2.9	
38281	NaN	0	NaN	-3.4	

	cons_price_idx	cons_conf_idx	euribor_3m	n_employed	invested
1266	93.994	-36.4	4.855	5191.0	no
12261	93.918	-42.7	4.966	5228.1	no
14234	93.918	-42.7	4.962	5228.1	no
16956	93.918	-42.7	4.962	5228.1	no
18465	93.918	-42.7	4.968	5228.1	no

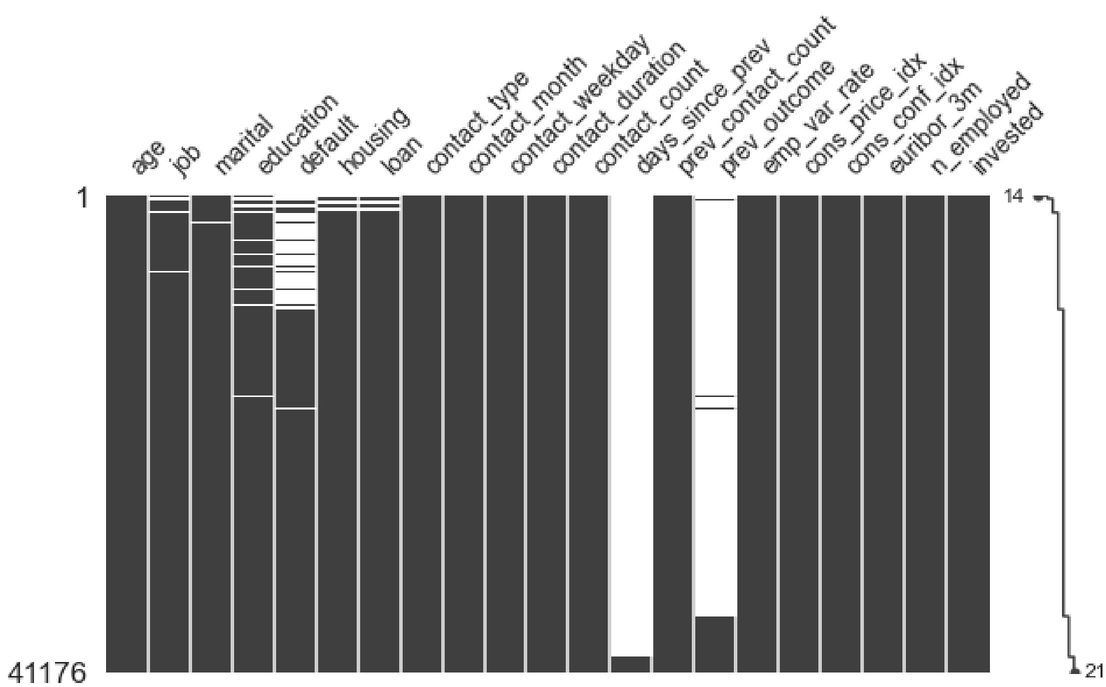
```
20216          93.444          -36.1      4.965      5228.1         no
20534          93.444          -36.1      4.966      5228.1         no
25217          93.200          -42.0      4.153      5195.8         no
28477          93.075          -47.1      1.423      5099.1         no
32516          92.893          -46.2      1.313      5099.1         no
36951          92.469          -33.6      1.072      5076.2        yes
38281          92.431          -26.9      0.742      5017.5         no

[12 rows x 21 columns]
```

[9]:
```python
cleaning.show_uniques(df, cut=20)
```

```
&lt;IPython.core.display.HTML object&gt;
```

[10]:
```python
msno.matrix(df, figsize=(10, 5), sort="ascending", fontsize=14)
```

[10]:
```
<AxesSubplot:>
```



I go ahead and encode "invested" as a boolean for convenience. It's the prediction target and I'll need to make calculations for EDA.

[11]:
```python
df["invested"] = df["invested"] == "yes"
df["invested"].value_counts(normalize=True)
```

[11]:
```
False    0.887337
True     0.112663
```

```
Name: invested, dtype: float64
```

I convert "prev_outcome" into two separate boolean features, eliminating ~35k NaNs and avoiding collinearity by implicitly dropping the NaN category. I wouldn't want to use imputation when NaNs make up the vast majority.

[12]:
```
df["prev_success"] = df["prev_outcome"] == "success"
df["prev_failure"] = df["prev_outcome"] == "failure"
df.drop(columns="prev_outcome", inplace=True)
df[["prev_success", "prev_failure"]].value_counts(normalize=True)
```

[12]:
```
prev_success  prev_failure
False         False           0.863391
              True            0.103264
True          False           0.033345
dtype: float64
```

[13]:
```
cleaning.token_info(df[["prev_success", "prev_failure"]], 1)
```

[13]:

	min_tokens	max_tokens	types
prev_success	0.033345	0.966655	2.0
prev_failure	0.103264	0.896736	2.0

Now to interpret the "days_since_prev" a.k.a. "pdays" feature. The guide says:

> number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

Passed by until what? Presumably, until they called back. But what if they never called back? Maybe those who never called back are also chalked up as null values, even though the guide doesn't say that.

Yep, as shown below, there are rows where "days_since_prev" is null and "prev_failure" is True. This must mean that those who never called back are chalked up as 999 days (null) and also as failures.

[14]:
```
df.loc[df["prev_failure"], ["days_since_prev", "prev_failure"]].head()
```

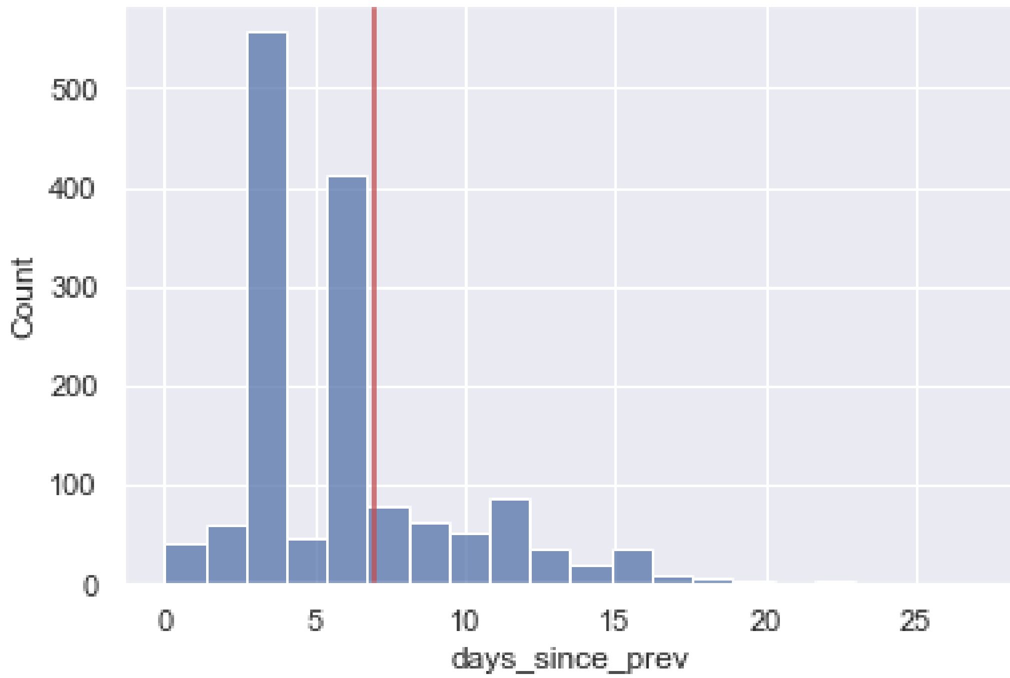
[14]:

	days_since_prev	prev_failure
24013	NaN	True
24019	NaN	True
24076	NaN	True
24102	NaN	True
24113	NaN	True

One week seems like a natural cutoff point for turning "days_since_prev" into categoricals.

[15]:
```
ax = sns.histplot(data=df, x="days_since_prev", bins=20)
ax.axvline(7, c="r")
```

[15]:
```
<matplotlib.lines.Line2D at 0x2067d046550>
```



I create three boolean variables for those who responded in under a week, over a week, or not at all. This effectively drops the NaN category.

I drop "days_since_prev" which is 96% NaNs and no longer necessary.

[16]:
```
df["prev_callback_under_1w"] = df["days_since_prev"] <= 7
df["prev_callback_over_1w"] = df["days_since_prev"] > 7
df["prev_callback_never"] = df["days_since_prev"].isna() & df["prev_failure"]
df.drop(columns="days_since_prev", inplace=True)
df.filter(like="prev_callback").tail()
```

[16]:

	prev_callback_under_1w	prev_callback_over_1w	prev_callback_never
41183	False	False	False
41184	False	False	False
41185	False	False	False
41186	False	False	False
41187	False	False	True

I convert the binary "contact_type" feature to a boolean feature "contact_cellular". Again, every variable with the "contact" prefix has to do with the last contact of the current campaign.

[17]:
```
df["contact_cellular"] = df["contact_type"] == "cellular"
df.drop(columns="contact_type", inplace=True)
df["contact_cellular"].value_counts()
```

[17]:

```
True     26135
False    15041
Name: contact_cellular, dtype: int64
```

I go ahead and numerically encode these binary string categoricals, preserving NaNs.

[18]:

```
string_cols = ["default", "housing", "loan"]
df[string_cols] = df[string_cols].applymap(strtobool, "ignore")
cleaning.show_uniques(df, columns=string_cols)
del string_cols
```

<IPython.core.display.HTML object>

For now I will hold off on converting these binary variables to categorical dtype. It will be easier to work with them as numeric variables, since they don't need to be one-hot encoded.

[19]:

```
binary_cats = utils.binary_cols(df)
df[binary_cats] = df[binary_cats].astype(np.float64)
cleaning.show_uniques(df, columns=binary_cats)
```

<IPython.core.display.HTML object>

Looks like most of the binary categoricals are imbalanced.

[20]:

```
cleaning.token_info(df[binary_cats], normalize=True)
```

[20]:

	min_tokens	max_tokens	types
default	0.000092	0.999908	2.0
prev_callback_over_1w	0.008209	0.991791	2.0
prev_callback_under_1w	0.028585	0.971415	2.0
prev_success	0.033345	0.966655	2.0
prev_callback_never	0.099815	0.900185	2.0
prev_failure	0.103264	0.896736	2.0
invested	0.112663	0.887337	2.0
loan	0.155477	0.844523	2.0
contact_cellular	0.365286	0.634714	2.0
housing	0.463221	0.536779	2.0

I drop the most extremely imbalanced binaries: "default" and "prev_callback_over_1w". These had a minority class of less than 1% of the sample.

[21]:

```
df.drop(columns=["default", "prev_callback_over_1w"], inplace=True)
del binary_cats
cleaning.token_info(df.loc[:, df.nunique() == 2], normalize=True)
```

[21]:

	min_tokens	max_tokens	types
prev_callback_under_1w	0.028585	0.971415	2.0
prev_success	0.033345	0.966655	2.0
prev_callback_never	0.099815	0.900185	2.0
prev_failure	0.103264	0.896736	2.0

```
invested                    0.112663    0.887337    2.0
loan                        0.155477    0.844523    2.0
contact_cellular            0.365286    0.634714    2.0
housing                     0.463221    0.536779    2.0
```

[22]:

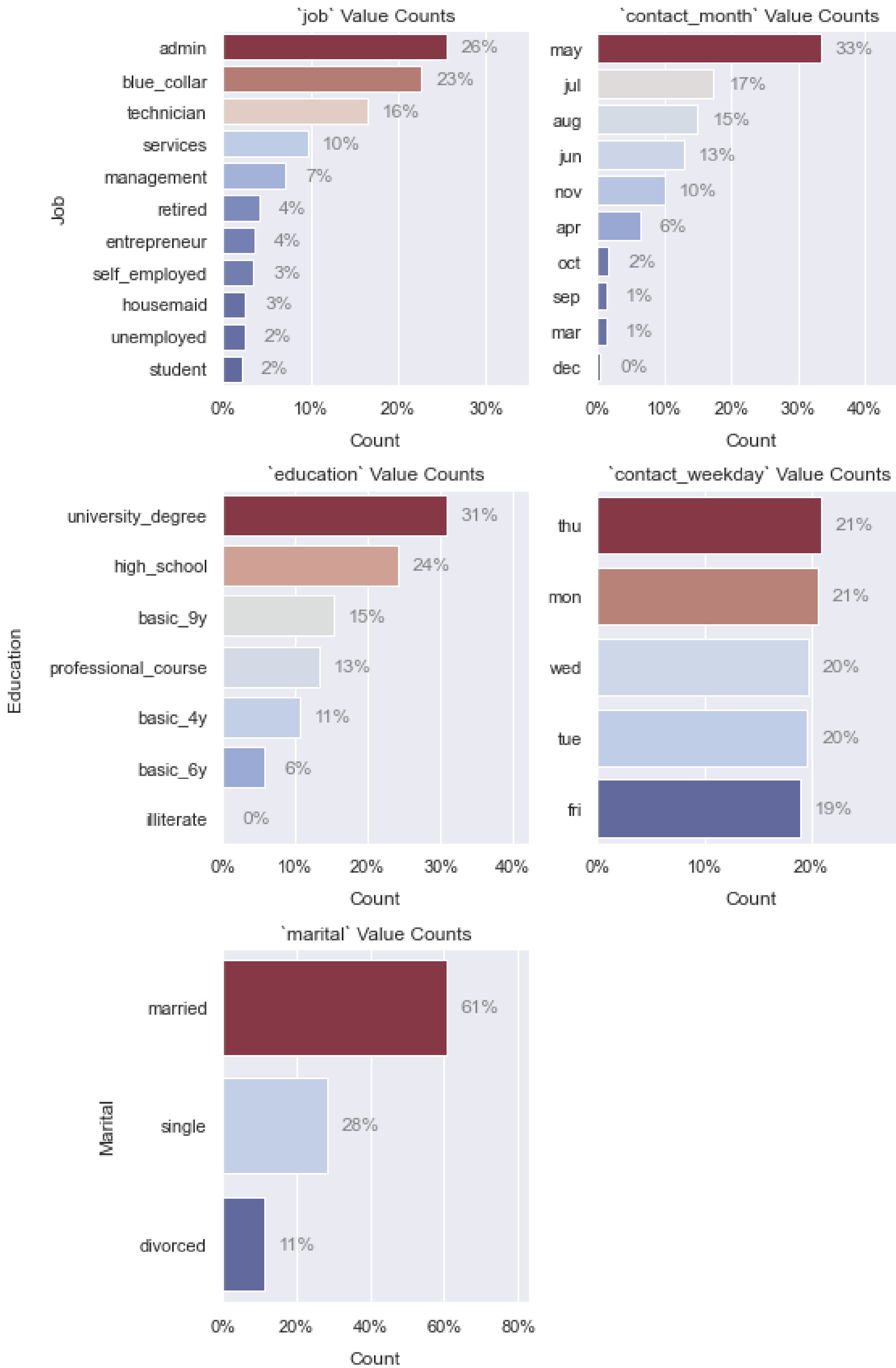
```
multi_cat = ["job", "marital", "education", "contact_month", "contact_weekday"]

# tweak some labels
df["job"] = df["job"].str.replace(".", "", regex=False)
df["job"] = df["job"].str.replace("-", "_", regex=False)
df["education"] = df["education"].str.replace(".", "_", regex=False)

# convert to unordered categoricals
df[multi_cat] = df[multi_cat].astype("category")



cleaning.show_uniques(df, columns=multi_cat)
```

```
<IPython.core.display.HTML object>
```

The class balance looks serviceable, but there are a couple extremely thin classes (under 1%): "dec" and "illiterate".

[23]:

```
plotting.multi_countplot(df[multi_cat], normalize=True, ncols=2, sp_height=4);
del multi_cat
```


`job` Value Counts
Job
admin
blue_collar
technician
services
management
retired
entrepreneur
self_employed
housemaid
unemployed
student
26%
23%
16%
10%
7%
4%
4%
3%
3%
2%
2%
0%
10%
20%
30%
Count
`contact_month` Value Counts
may
jul
aug
jun
nov
apr
oct
sep
mar
dec
33%
17%
15%
13%
10%
6%
2%
1%
1%
0%
0%
10%
20%
30%
40%
Count
`education` Value Counts
Education
university_degree
high_school
basic_9y
professional_course
basic_4y
basic_6y
illiterate
31%
24%
15%
13%
11%
6%
0%
0%
10%
20%
30%
40%
Count
`contact_weekday` Value Counts
thu
mon
wed
tue
fri
21%
21%
20%
20%
19%
0%
10%
20%
Count
`marital` Value Counts
Marital
married
single
divorced
61%
28%
11%
0%
20%
40%
60%
80%
Count

I drop the extremely thin "illiterate" and "dec" classes.

[24]:
```
# compute rows to keep
keep = (df.education != "illiterate") & (df.contact_month != "dec")

# overwrite `df` with keeper rows
df = df.loc[keep].copy()

# drop unused categories
df["education"] = df["education"].cat.remove_unused_categories()
df["contact_month"] = df["contact_month"].cat.remove_unused_categories()

# view results
print(f"Dropped {(~keep).sum()} observations.")
del keep
cleaning.token_info(df[["education", "contact_month"]], normalize=True)
```

```
Dropped 200 observations.
```

[24]:

	min_tokens	max_tokens	types
contact_month	0.013325	0.335904	9.0
education	0.058355	0.308049	6.0

I order the weekdays and months for plotting purposes.

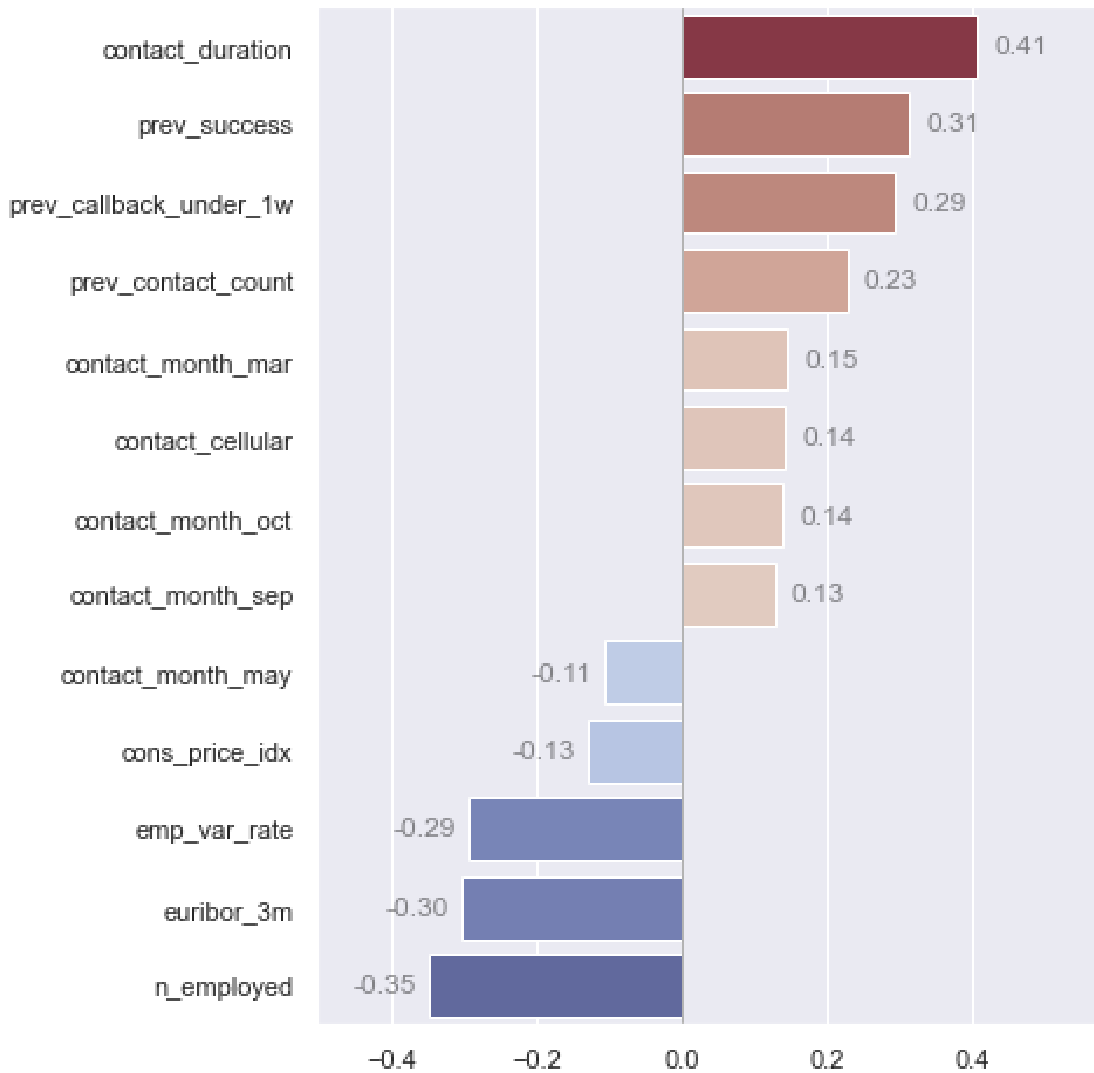
[25]:
```
# define order
days = ["mon", "tue", "wed", "thu", "fri"]
months = ["mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov"]

# convert to ordered categories
df["contact_weekday"].cat.reorder_categories(days, ordered=True, inplace=True)
df["contact_month"].cat.reorder_categories(months, ordered=True, inplace=True)

# mop up temp variables
del days, months

display(df["contact_weekday"].cat.categories)
display(df["contact_month"].cat.categories)
```

```
Index(['mon', 'tue', 'wed', 'thu', 'fri'], dtype='object')
```

```
Index(['mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov'],␣
↪dtype='object')
```

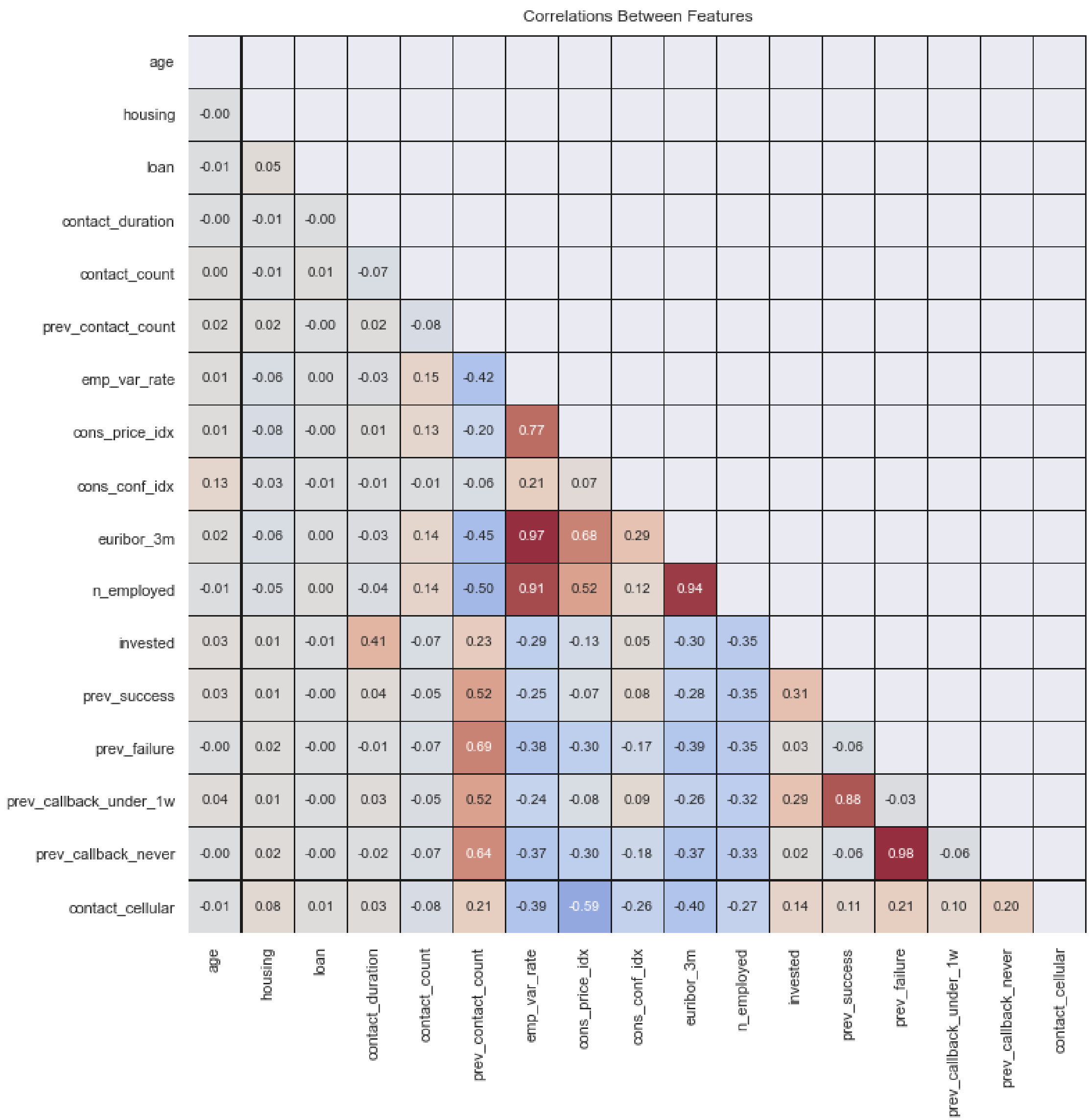
5 EXPLORE

[26]:
```
inv_corr = pd.get_dummies(df.drop(columns="invested"))
inv_corr = inv_corr.corrwith(df["invested"])
inv_corr = inv_corr.loc[inv_corr.abs() > .1]
ax = plotting.heated_barplot(inv_corr)
plotting.annot_bars(ax)
del inv_corr, ax
```



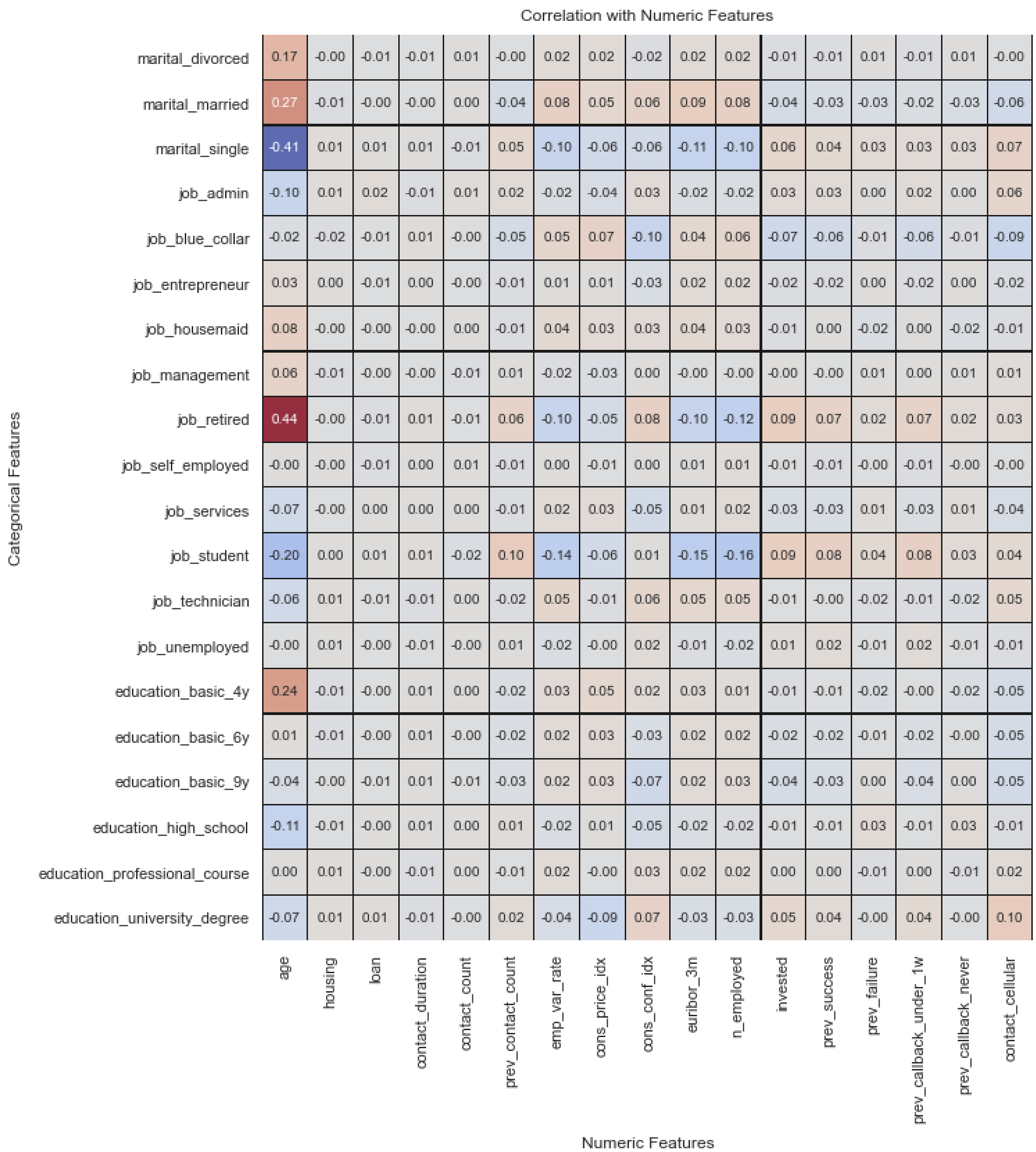
[27]:
```
plotting.pair_corr_heatmap(df, scale=.7)
```

[27]: <AxesSubplot:title={'center':'Correlations Between Features'}>

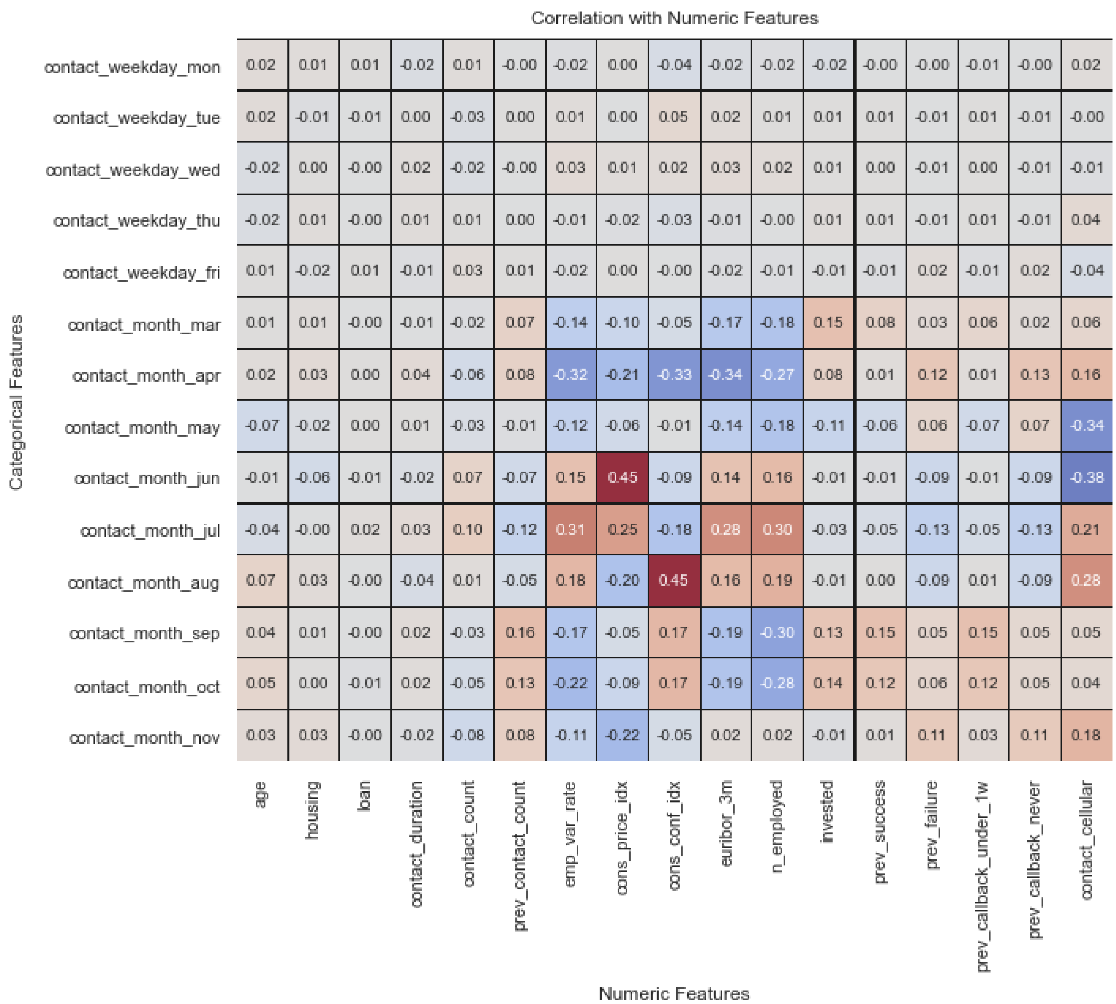


[28]:
```python
plotting.cat_corr_heatmap(
    df, ["marital", "job", "education"], scale=.6, fmt=".2f")
```

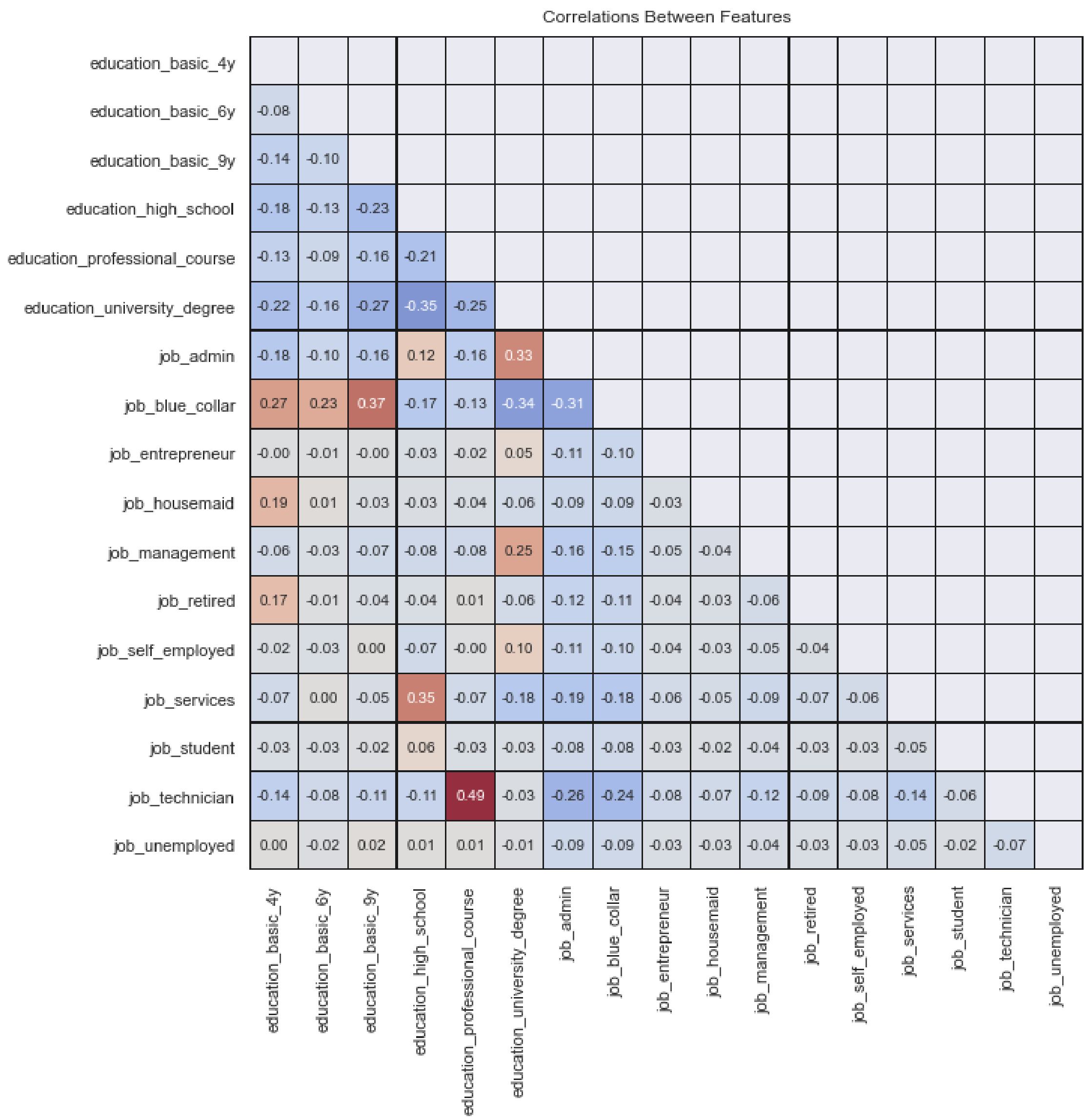
[28]:
```
<AxesSubplot:title={'center':'Correlation with Numeric Features'},
xlabel='Numeric Features', ylabel='Categorical Features'>
```



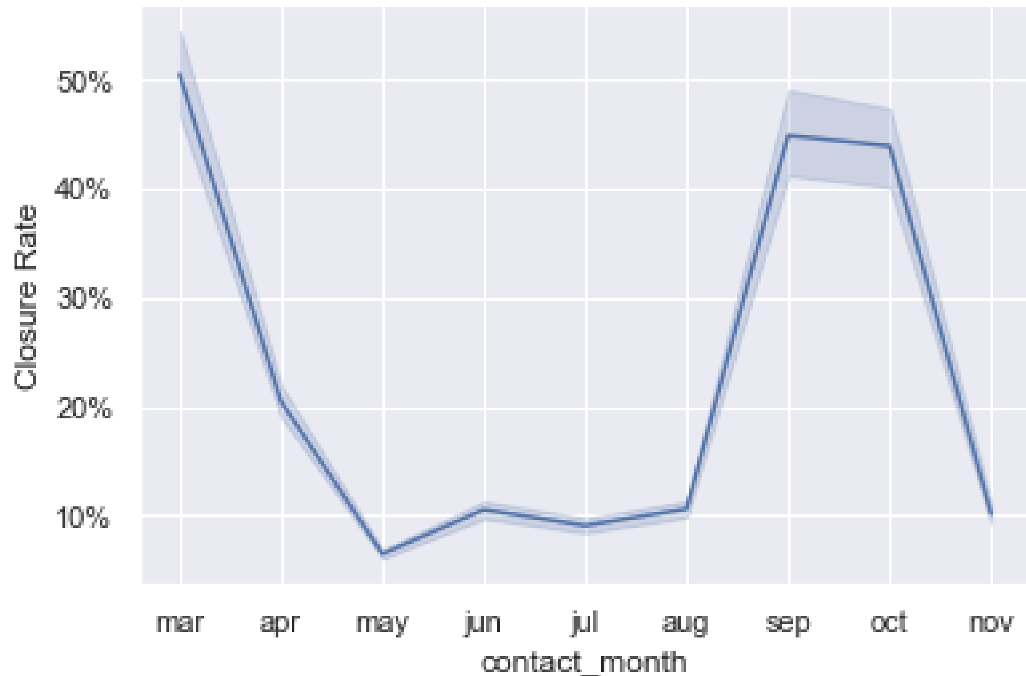
[29]:
```
temporal = ["contact_weekday", "contact_month"]
plotting.cat_corr_heatmap(df, temporal, scale=.6, fmt=".2f")
del temporal
```



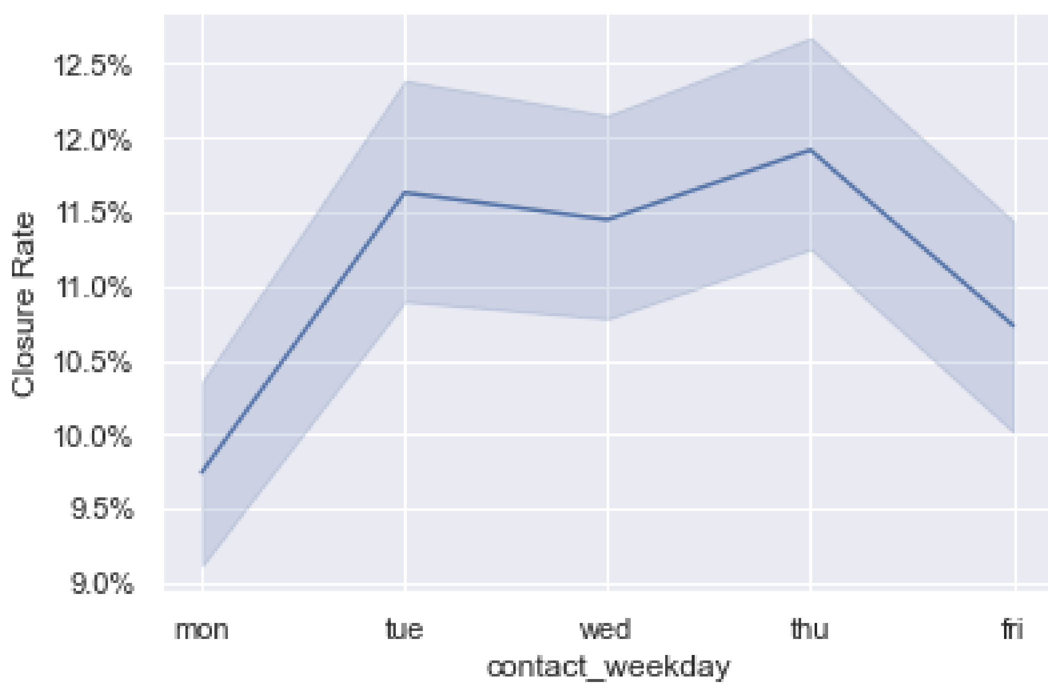
[30]:
```
dummies = pd.get_dummies(df[["education", "job"]])
plotting.pair_corr_heatmap(dummies, scale=.6, fmt=".2f")
del dummies
```



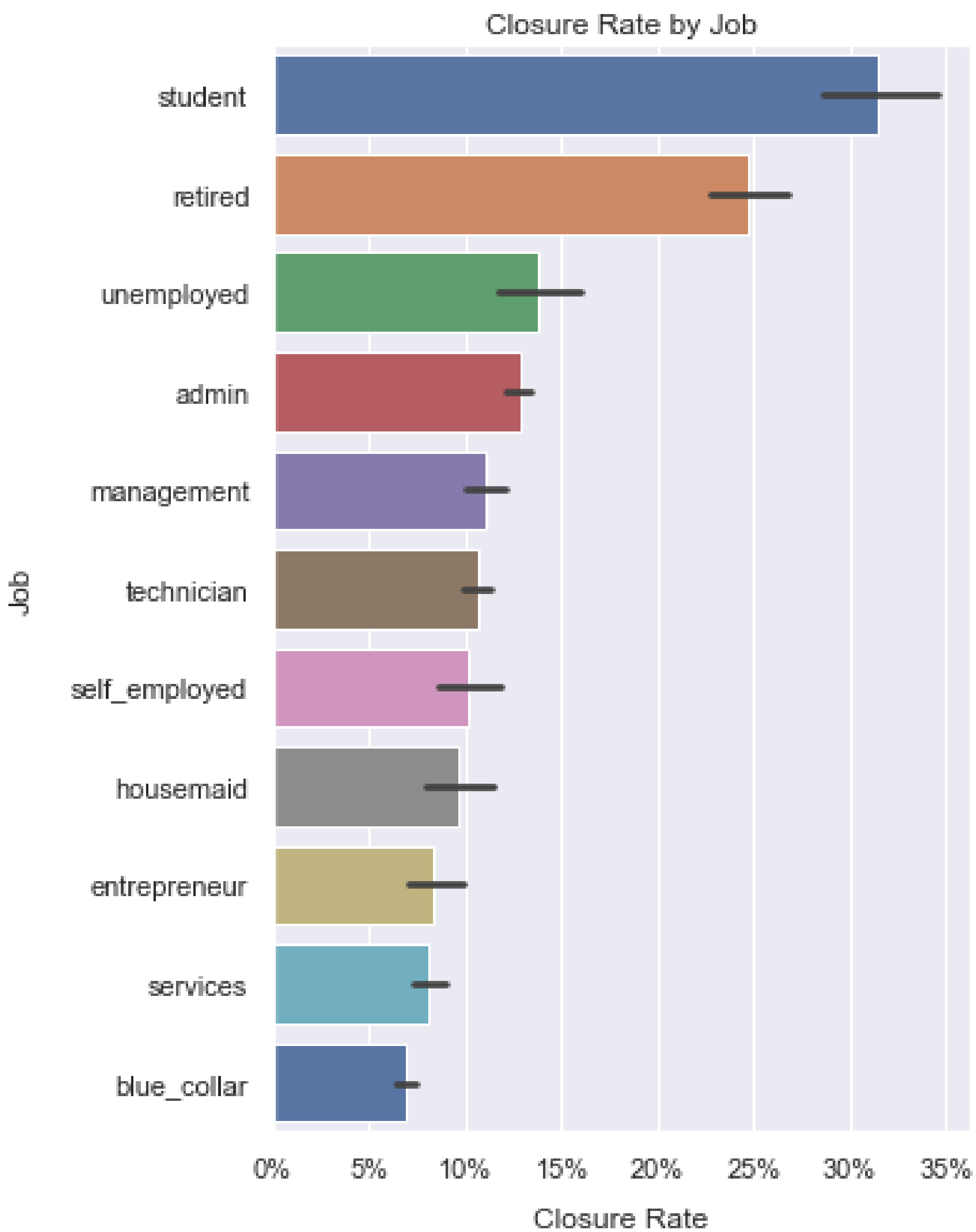

```
[31]: ax = sns.lineplot(data=df, x="contact_month", y="invested")
      ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1))
      ax.set_ylabel("Closure Rate")
      del ax
```



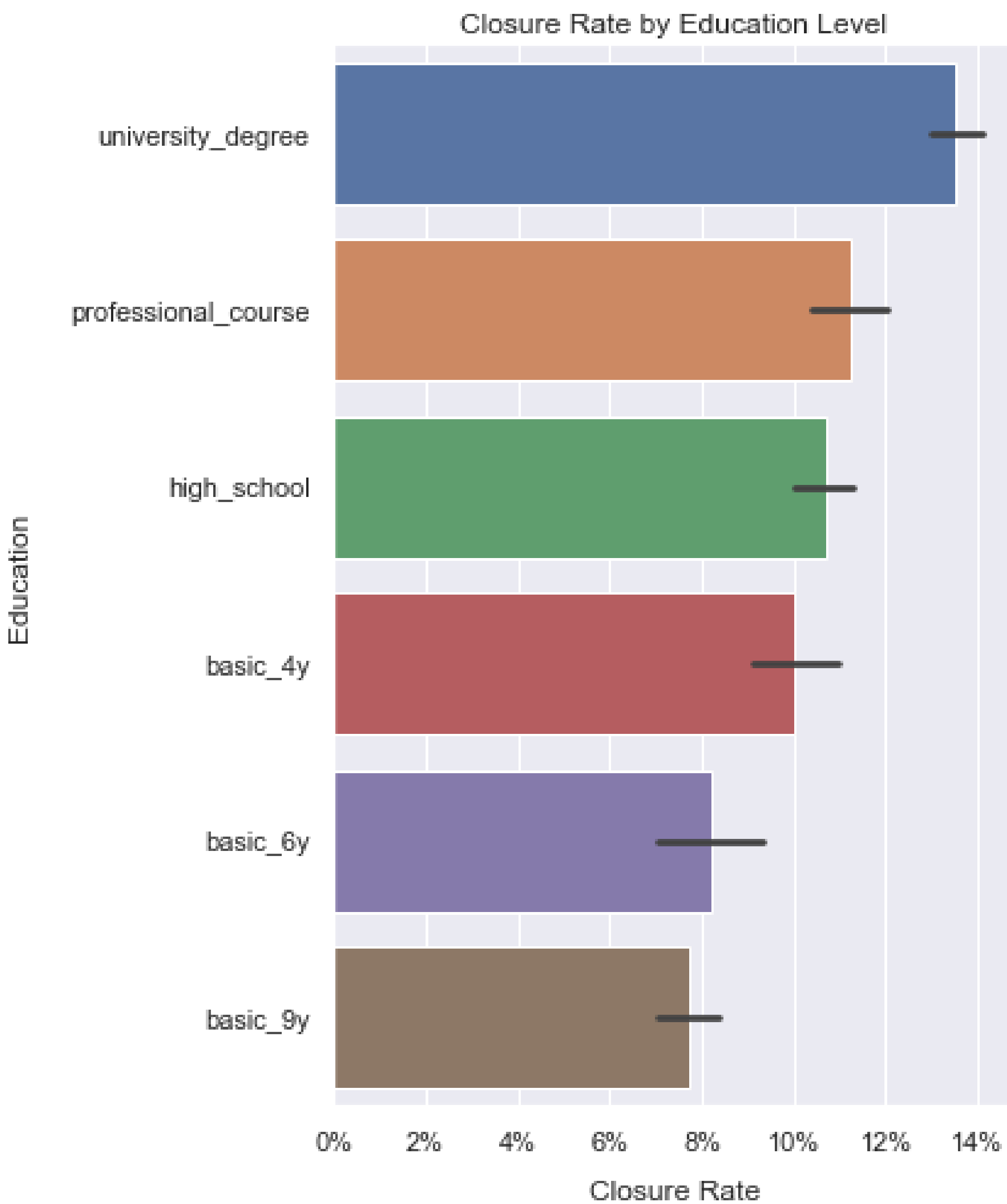
[32]:
```
ax = sns.lineplot(data=df, x="contact_weekday", y="invested")
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals=1))
ax.set_ylabel("Closure Rate")
del ax
```



[33]:
```
ax = plotting.simple_barplot(df, "job", "invested", sort="desc", orient="h",␣
↪palette="deep")
ax.xaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals=0))
ax.set_xlabel("Closure Rate", labelpad=10)
ax.set_title("Closure Rate by Job", pad=5)
del ax
```

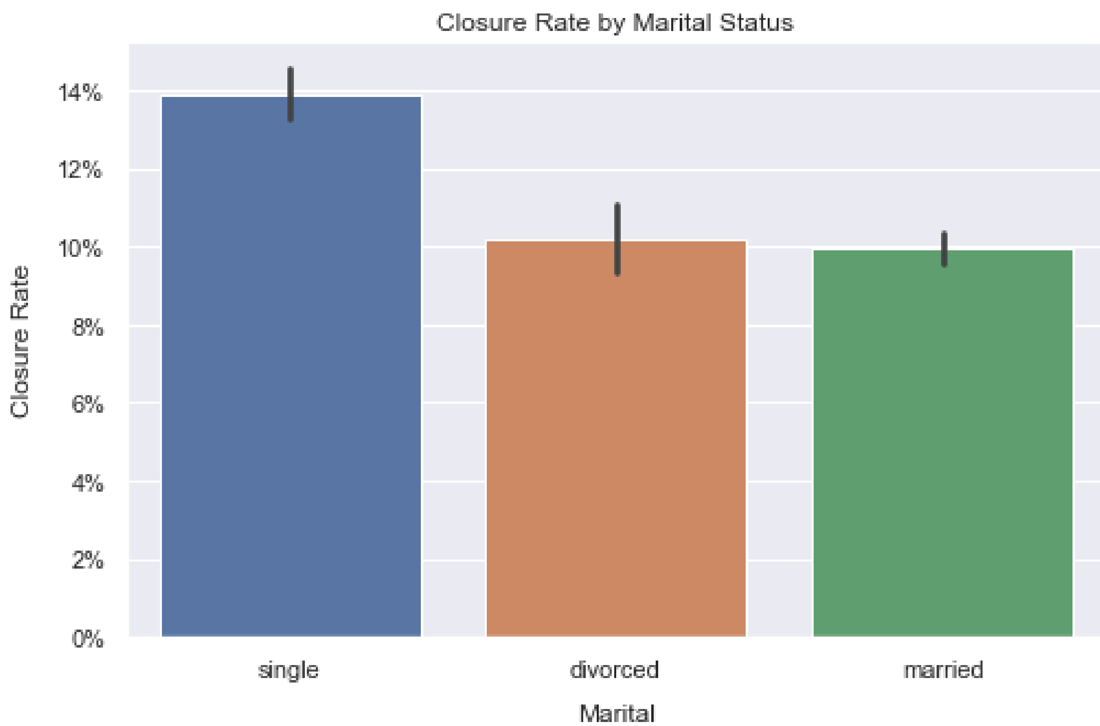


[34]:
```
ax = plotting.simple_barplot(df, "education", "invested", sort="desc",␣
↪orient="h", palette="deep")
ax.xaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals=0))
ax.set_xlabel("Closure Rate", labelpad=10)
ax.set_title("Closure Rate by Education Level", pad=5)
del ax
```



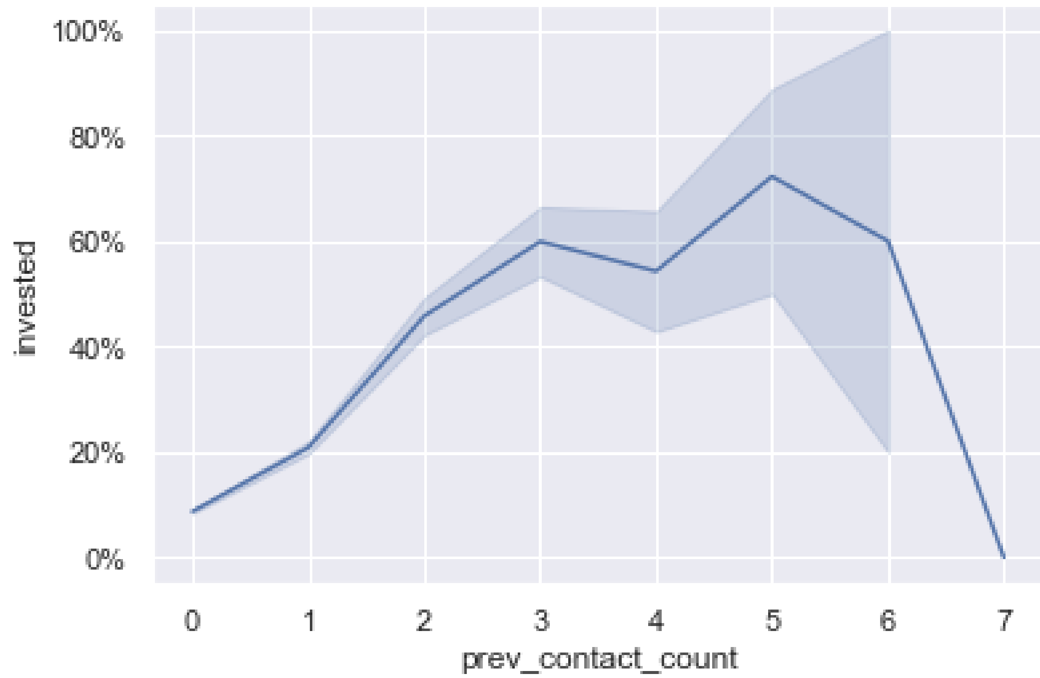
[35]:

```
ax = plotting.simple_barplot(df, "marital", "invested", sort="desc",␣
↪orient="v", palette="deep")
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals=0))
ax.set_ylabel("Closure Rate", labelpad=10)
ax.set_title("Closure Rate by Marital Status", pad=5)
del ax
```



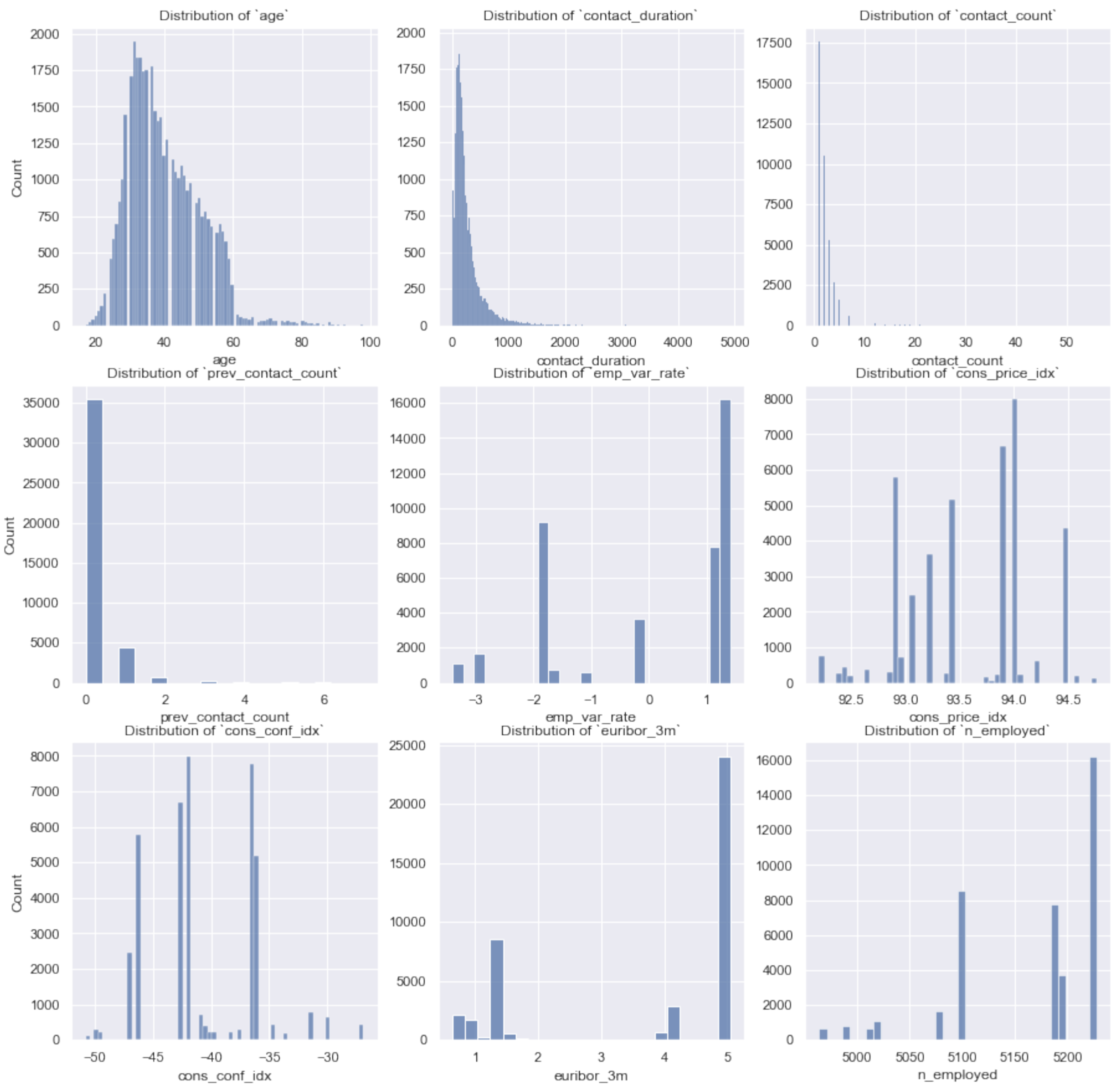
[36]:

```
ax = sns.lineplot(data=df, x="prev_contact_count", y="invested")
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals=0))
del ax
```



[37]:
```
plotting.multi_dist(df[utils.true_numeric_cols(df)])
```

[37]:
```
array([[<AxesSubplot:title={'center':'Distribution of `age`'}, xlabel='age',
ylabel='Count'>,
        <AxesSubplot:title={'center':'Distribution of `contact_duration`'},
xlabel='contact_duration'>,
        <AxesSubplot:title={'center':'Distribution of `contact_count`'},
xlabel='contact_count'>],
       [<AxesSubplot:title={'center':'Distribution of `prev_contact_count`'},
xlabel='prev_contact_count', ylabel='Count'>,
        <AxesSubplot:title={'center':'Distribution of `emp_var_rate`'},
xlabel='emp_var_rate'>,
        <AxesSubplot:title={'center':'Distribution of `cons_price_idx`'},
xlabel='cons_price_idx'>],
       [<AxesSubplot:title={'center':'Distribution of `cons_conf_idx`'},
xlabel='cons_conf_idx', ylabel='Count'>,
        <AxesSubplot:title={'center':'Distribution of `euribor_3m`'},
xlabel='euribor_3m'>,
        <AxesSubplot:title={'center':'Distribution of `n_employed`'},
xlabel='n_employed'>]],
      dtype=object)
```



6 MODEL

[410]:
```python
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.metrics import plot_roc_curve, plot_confusion_matrix,␣
 ↪classification_report
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, RobustScaler,␣
 ↪MinMaxScaler
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.compose import ColumnTransformer, make_column_selector
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.feature_selection import RFE
# from imblearn.over_sampling import SMOTENC
import functools
```

6.1 First Model

For my baseline model, I apply minimal preprocessing to the data. The first thing I do is ensure that all numeric features are in float format.

[39]:
```
df[utils.numeric_cols(df)] = df.select_dtypes("number").astype(np.float64)
```

My initial preprocessing "pipeline" is really just a `ColumnTransformer` which one-hot encodes categorical variables with three or more categories. Recall that binary categoricals are already numerically encoded.

This pipeline essentially does no preprocessing at all other than the bare minimum required for multi-class categorical variables.

[336]:
```
cat_xform = [
    ("onehot", OneHotEncoder(), utils.multicat_cols)
]
cat_xform = ColumnTransformer(cat_xform, remainder="passthrough",␣
 ↪sparse_threshold=0)
cat_xform
```

[336]:
```
ColumnTransformer(remainder='passthrough', sparse_threshold=0,
                  transformers=[('onehot', OneHotEncoder(),
                                 <function multicat_cols at
0x000002067B7BBB80>)])
```

Here I apply preprocessing and perform a train-test split.

Notice that I simply drop all observations with missing values. This is arguably the crudest way to deal with them, and I want to start off crudely.

I drop "contact_duration", because as noted in on the UCI Repo for this dataset, it reduces the practical value of the model. This was the duration of the last call, after which the broker knew whether or not the customer invested. The duration of the final call wouldn't be known prior to calling the customer. What should I tell the broker, "keep them on the line for longer than zero seconds?"

[286]:
```
# drop NaNs and irrelevant columns
X = df.dropna().drop(columns=["invested", "contact_duration"])

# drop NaNs and slice target column
y = df.dropna()["invested"].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[286]: ((28541, 20), (9514, 20), (28541,), (9514,))

6.1.1 Random Dummy Model

I create a dummy model which makes uniform random predictions. It's good to ensure that my models are better than an extremely dumb alternative. If the dummy model is good at all, it's due to pure luck.

I train the model on all features except the target.

The diagonal of the confusion matrix indicates that the dummy gets 51% of the true negatives and 48% of the true positives, which is pretty bad.

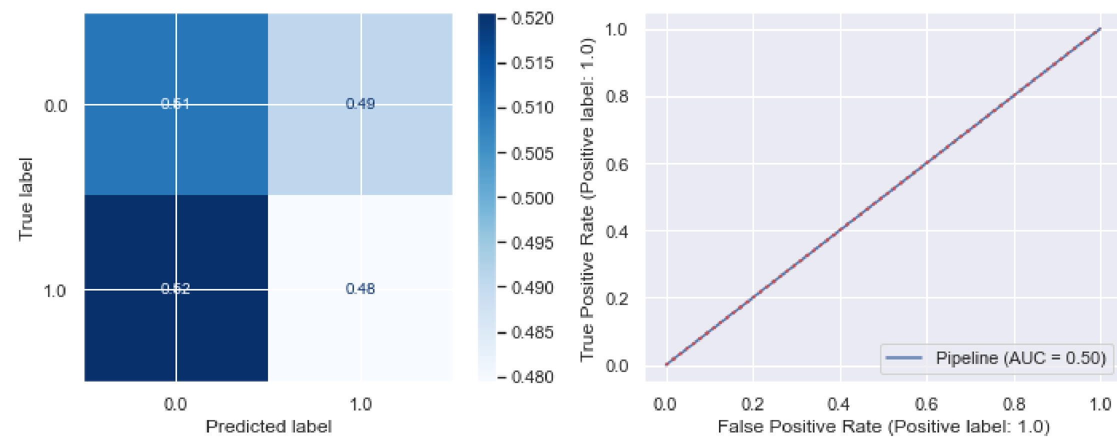
The ROC curve is not even a curve, because if falls directly on the 1:1 line, with 0.5 AUC. This is about the worst a model could do.

[287]:
```python
dummy_pipe = [
    ("cat_xform", cat_xform),
    ("dummy", DummyClassifier(strategy="uniform", random_state=63))
]
dummy_pipe = Pipeline(dummy_pipe)
dummy_pipe
```

[287]:
```
Pipeline(steps=[('cat_xform',
                 ColumnTransformer(remainder='passthrough', sparse_threshold=0,
                                   transformers=[('cat_pipe',
Pipeline(steps=[('mode_impute',
SimpleImputer(strategy='most_frequent')),
                                                                   ('onehot',
OneHotEncoder(sparse=False))]),
                                                 <function multicat_cols at
0x000002067B7BBB80>)])),
                ('dummy',
                 DummyClassifier(random_state=63, strategy='uniform'))])
```

[288]:
```python
dummy_pipe.fit(X_train, y_train)
diagnostics.class_report(dummy_pipe, X_test, y_test)
```

```
              precision    recall  f1-score   support

         0.0       0.89      0.51      0.65      8448
         1.0       0.11      0.48      0.18      1066

    accuracy                           0.51      9514
   macro avg       0.50      0.49      0.41      9514
weighted avg       0.80      0.51      0.59      9514
```



6.1.2 Baseline Logistic Regression

My baseline model is surprisingly good, despite the minimal preprocessing. It's trained on all features except the target, completely ignoring potential multicollinearity issues.

One powerful feature of the `LogisticRegression` estimator is the `class_weight="balanced"` setting, which adjusts the model to accommodate the very imbalanced (9:1) target classes.

The diagonal of the confusion matrix indicates that it gets ~88% of the labels correct. That's pretty good, and much better than `dummy_model`, which gets ~50% correct.

The ROC curve looks very good, and the Area Under Curve (AUC) is a solid 0.94.

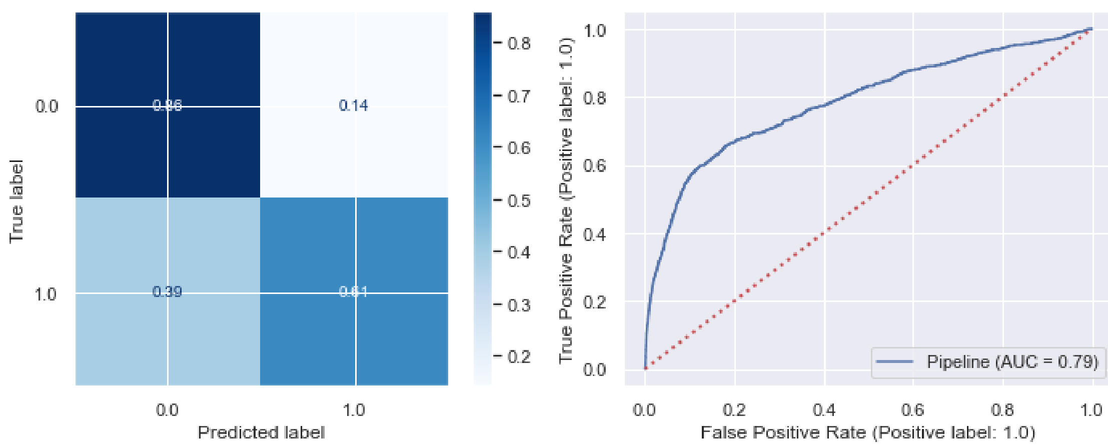
[289]:
```
logit_pipe = [
    ("cat_xform", cat_xform),
    ("logit", LogisticRegression(fit_intercept=False,
                                 C=1e12,
                                 multi_class="ovr",
                                 max_iter=1e4,
                                 class_weight="balanced",
                                 solver="liblinear"))
]
logit_pipe = Pipeline(logit_pipe)
logit_pipe
```

[289]:
```
Pipeline(steps=[('cat_xform',
                 ColumnTransformer(remainder='passthrough', sparse_threshold=0,
                                   transformers=[('cat_pipe',
Pipeline(steps=[('mode_impute',
SimpleImputer(strategy='most_frequent')),
                                                                  ('onehot',
OneHotEncoder(sparse=False))]),
                                                 <function multicat_cols at
```

```
0x000002067B7BBB80>)])),
                    ('logit',
                     LogisticRegression(C=1000000000000.0, class_weight='balanced',
                                        fit_intercept=False, max_iter=10000.0,
                                        multi_class='ovr', solver='liblinear'))])
```

[290]:
```
logit_pipe.fit(X_train, y_train)
diagnostics.class_report(logit_pipe, X_test, y_test)
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.86      0.90      8448
         1.0       0.35      0.61      0.45      1066

    accuracy                           0.83      9514
   macro avg       0.65      0.73      0.67      9514
weighted avg       0.88      0.83      0.85      9514
```



Recall that the target classes have about a 9:1 ratio.

[148]:
```
df["invested"].value_counts(1).round(2)
```

[148]:
```
0.0    0.89
1.0    0.11
Name: invested, dtype: float64
```

[149]:
```
logit_pipe["logit"].coef_
```

[149]:
```
array([[-3.40028468e-03, -1.43503957e-02, -7.93249214e-02,
        -2.10934691e-01,  2.33306711e-02,  3.34916466e-01,
        -9.09033977e-03, -1.36304591e-01,  3.53638904e-01,
        -3.09576218e-02, -3.15874541e-01, -3.23677393e-02,
```

```
       -6.12879723e-02,  5.30436473e-03, -1.46422390e-01,
        3.23639556e-02, -1.04823707e-01, -7.99444875e-03,
        2.42913770e-02,  1.14233867e-01, -1.77136425e-02,
        2.24547084e-01,  2.30997736e-01, -9.04266559e-02,
        1.22414196e+00, -5.91871197e-01, -7.47942444e-01,
       -5.59735720e-02, -2.64110614e-01,  4.24680436e-02,
       -1.76504546e-01, -3.24769259e-02, -1.62015523e-02,
        9.43636334e-02, -1.17945010e-03, -1.47271726e-02,
       -1.13827095e-01, -2.90591033e-02, -1.23787277e-01,
       -9.52269243e-01,  6.06392278e-01, -8.12502934e-03,
        7.69472995e-01, -1.16340398e-02,  1.61297461e+00,
        8.09652957e-01, -3.97344216e-02, -1.15377114e+00,
        5.40791969e-01]])
```

6.2 Second Model

6.2.1 Preprocessing Enhancements

[291]:
```
cat_pipe = Pipeline([
    ("mode_impute", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder()),
])
cat_pipe
```

[291]:
```
Pipeline(steps=[('mode_impute', SimpleImputer(strategy='most_frequent')),
                ('onehot', OneHotEncoder())])
```

[292]:
```
col_xform = [
    ("cat_pipe", cat_pipe, utils.multicat_cols),
    ("scale", StandardScaler(), utils.true_numeric_cols),
]
col_xform = ColumnTransformer(col_xform, remainder='passthrough')
col_xform
```

[292]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('cat_pipe',
                                 Pipeline(steps=[('mode_impute',
SimpleImputer(strategy='most_frequent')),
                                                 ('onehot', OneHotEncoder())]),
                                 <function multicat_cols at
0x000002067B7BBB80>),
                                ('scale', StandardScaler(),
                                 <function true_numeric_cols at
0x000002067B7A4CA0>)])
```

[293]:
```
pre_pipe = Pipeline([
    ("col_xform", col_xform),
    ("knn_impute",  KNNImputer()),
```

```
])
```

[435]:
```
logit_pipe = Pipeline([
    ("pre_pipe", pre_pipe),
    ("logit", LogisticRegressionCV(fit_intercept=False,
                                   Cs=[1.0, 1e3, 1e6, 1e9, 1e12],
                                   multi_class="ovr",
                                   class_weight="balanced",
                                   solver="liblinear"))
])
logit_pipe
```

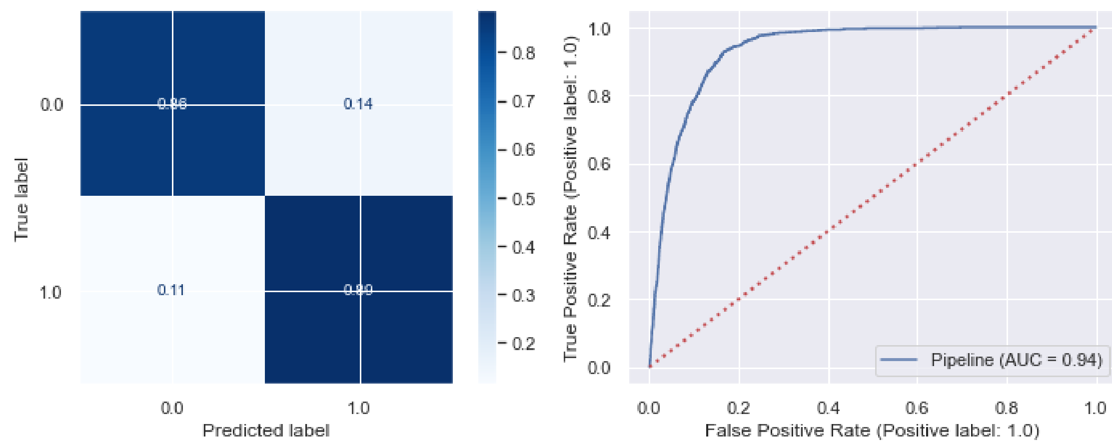
[435]:
```
Pipeline(steps=[('pre_pipe',
                 Pipeline(steps=[('col_xform',
                                  ColumnTransformer(remainder='passthrough',
                                                    transformers=[('cat_pipe',
Pipeline(steps=[('mode_impute',
    SimpleImputer(strategy='most_frequent')),
   ('onehot',
    OneHotEncoder())]),
                                                                   <function
multicat_cols at 0x000002067B7BBB80>),
                                                                  ('scale',
StandardScaler(),
                                                                   <function
true_numeric_cols at 0x000002067B7A4CA0>)])),
                                 ('knn_impute', KNNImputer())])),
                ('logit',
                 LogisticRegressionCV(Cs=[1.0, 1000.0, 1000000.0, 1000000000.0,
                                          1000000000000.0],
                                      class_weight='balanced',
                                      fit_intercept=False, multi_class='ovr',
                                      solver='liblinear'))])
```

[436]:
```
# drop irrelevant columns
X = df.drop(columns=["invested"])

# slice target column
y = df["invested"]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[436]:
```
((30732, 21), (10244, 21), (30732,), (10244,))
```

[437]:
```
logit_pipe.fit(X_train, y_train)
diagnostics.class_report(logit_pipe, X_test, y_test)
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.86      0.92      9129
         1.0       0.43      0.89      0.58      1115

    accuracy                           0.86     10244
   macro avg       0.71      0.87      0.75     10244
weighted avg       0.92      0.86      0.88     10244
```



7 iNTERPRET

[ ]:

8 CONCLUSIONS & RECOMMENDATIONS

Summarize your conclusions and bullet-point your list of recommendations, which are based on your modeling results.

9 TO DO/FUTURE WORK

-

[ ]: