CPSC 330 - Applied Machine Learning

Homework 5: Evaluation metrics

Associated lectures: Lectures 9, 10

Due date: Tuesday, June 07, 2022 at 18:00

Imports

In [1]:
```
import os
import re
import sys
from hashlib import sha1

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tests_hw5
from sklearn import datasets
from sklearn.compose import make_column_transformer
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
    make_scorer,
    precision_score,
    recall_score,
)
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
```

Instructions

---

rubric={points:3}

Follow the homework submission instructions.

You may work with a partner on this homework and submit your assignment as a group. Below are some instructions on working as a group.

- The maximum group size is 2.

- Use group work as an opportunity to collaborate and learn new things from each other.
- Be respectful to each other and make sure you understand all the concepts in the assignment well.
- It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline.
- You can find the instructions on how to do group submission on Gradescope here.

Exercise 1: Precision, recall, and f1 score by hand

Consider the problem of predicting whether a patient has a disease or not. Below are confusion matrices of two machine learning models: Model A and Model B.

- Model A confusion matrix

In [2]:
```
cm_A = pd.DataFrame(
    [[3, 10],
     [1, 106]],
    columns=["Predicted disease", "Predicted no disease"],
    index=["Actual disease", "Actual no disease"])

cm_A
```

Out[2]:

	Predicted disease	Predicted no disease
Actual disease	3	10
Actual no disease	1	106

- Model B confusion matrix

In [3]:
```
cm_B = pd.DataFrame(
    [[8, 5],
     [12, 95]],
    columns=["Predicted disease", "Predicted no disease"],
    index=["Actual disease", "Actual no disease"])

cm_B
```

Out[3]:

	Predicted disease	Predicted no disease
Actual disease	8	5
Actual no disease	12	95

1.1 Positive vs. negative class

rubric={points:2}

Your tasks:

Precision, recall, and f1 score depend upon which class is considered "positive", that is the thing you wish to find. In the example above, which class is likely to be the "positive" class? Why?

BEGIN SOLUTION

Having the disease is likely to be the "positive" class in the above example; we are likely to be interested in spotting the patients having the disease.

END SOLUTION

1.2 Accuracy

rubric={points:2}

Your tasks:

Calculate accuracies for Model A and Model B.

We'll store all metrics associated with Model A and Model B in the `results_dict` below.

In [4]:
```
results_dict = {"A": {}, "B": {}}
```

In [5]:
```
results_dict["A"]["accuracy"] = None
results_dict["B"]["accuracy"] = None

### BEGIN SOLUTION

results_dict["A"]["accuracy"] = np.diag(cm_A).sum() / cm_A.sum().sum()
results_dict["B"]["accuracy"] = np.diag(cm_B).sum() / cm_B.sum().sum()

### END SOLUTION
```

In [6]:
```
assert tests_hw5.ex1_2_1(
    results_dict["A"]["accuracy"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [7]:
```
assert tests_hw5.ex1_2_2(
    results_dict["B"]["accuracy"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [8]:
```
pd.DataFrame(results_dict)
```

Out[8]:

	A	B
accuracy	0.908333	0.858333

1.3 Which model would you pick?

rubric={points:1}

Your tasks:

Which model would you pick simply based on the accuracy metric?

BEGIN SOLUTION

Since Model A has higher accuracy than model B, I would pick model A simply based on this information.

END SOLUTION

1.4 Precision, recall, f1-score

rubric={points:6}

Your tasks:

1. Calculate precision, recall, f1-score for Model A and Model B manually, without calling `scikit-learn` functions.

In [9]:
```
results_dict["A"]["precision"] = None
results_dict["B"]["precision"] = None
results_dict["A"]["recall"] = None
results_dict["B"]["recall"] = None
results_dict["A"]["f1"] = None
results_dict["B"]["f1"] = None


### BEGIN SOLUTION


def calculate_f1(precision, recall):
    return (2 * precision * recall) / (precision + recall)


results_dict["A"]["precision"] = cm_A.iloc[0,0] / cm_A.iloc[:,0].sum()
results_dict["B"]["precision"] = cm_B.iloc[0,0] / cm_B.iloc[:,0].sum()
results_dict["A"]["recall"] = cm_A.iloc[0,0] / cm_A.iloc[0,:].sum()
results_dict["B"]["recall"] = cm_B.iloc[0,0] / cm_B.iloc[0,:].sum()
results_dict["A"]["f1"] = calculate_f1(
    results_dict["A"]["precision"], results_dict["A"]["recall"]
)
results_dict["B"]["f1"] = calculate_f1(
    results_dict["B"]["precision"], results_dict["B"]["recall"]
)
### END SOLUTION
```

In [10]:
```
assert tests_hw5.ex1_4_1(
```

```
    results_dict["A"]["precision"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [11]:
```
assert tests_hw5.ex1_4_2(
    results_dict["B"]["precision"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [12]:
```
assert tests_hw5.ex1_4_3(
    results_dict["A"]["recall"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [13]:
```
assert tests_hw5.ex1_4_4(
    results_dict["B"]["recall"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [14]:
```
assert tests_hw5.ex1_4_5(
    results_dict["A"]["f1"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

In [15]:
```
assert tests_hw5.ex1_4_6(
    results_dict["B"]["f1"]
), "Your answer is incorrect, see traceback above."
print("Success")
```

```
Success
```

Show the dataframe with all results.

In [16]:
```
pd.DataFrame(results_dict)
```

Out[16]:

	A	B
accuracy	0.908333	0.858333
precision	0.750000	0.400000
recall	0.230769	0.615385
f1	0.352941	0.484848

In [17]:
```
pd.DataFrame(results_dict)
```

Out[17]:

	A	B
accuracy	0.908333	0.858333
precision	0.750000	0.400000
recall	0.230769	0.615385
f1	0.352941	0.484848

1.5 Discussion

rubric={points:4}

Your tasks:

1. Which metric is more informative in this problem? Why?
2. Which model would you pick based on this information?

BEGIN SOLUTION

Recall would be more important than precision in this case because it's better to be a bit more aggressive in spotting the people with disease even with the risk of more false positives than incorrectly classifying them as non-diseased.

I would pick Model B because it has a better recall and `f1 score`.

END SOLUTION

(Optional) 1.6

rubric={points:1}

Your tasks:

Provide 4 to 5 example classification datasets (with links) where accuracy metric would be misleading. Discuss which evaluation metric would be more appropriate for each dataset. You may consider datasets we have used in this course so far. You could also look up datasets on Kaggle.

BEGIN SOLUTION

- Credit Card Fraud Detection Dataset Recall and f1 score would be more relevant.
- SMS Spam Collection Dataset Precision and f1 score would be more relevant.

(Full marks if there are 2 or more reasonable examples.)

END SOLUTION

Exercise 2: Classification evaluation metrics using `sklearn`

In general, when a dataset is imbalanced, accuracy does not provide the whole story. In class, we looked at credit card fraud dataset which is a classic example of an imbalanced dataset.

Another example is customer churn datasets. Customer churn refers to the notion of customers leaving a subscription service like Netflix. In this exercise, we will try to predict customer churn in a dataset where most of the customers stay with the service and a small minority cancel their subscription. To start, please download the Kaggle telecom customer churn dataset. Once you have the data, you should be able to run the following code:

The starter code below reads the data CSV as a pandas dataframe and splits it into 70% train and 30% test.

Note that `churn` column in the dataset is the target. "True" means the customer left the subscription (churned) and "False" means they stayed.

> Note that for this kind of problem a more appropriate technique is something called survival analysis and we'll be talking about it later in the course. For now, we'll just treat it as a binary classification problem.

In [18]:
```
df = pd.read_csv("bigml_59c28831336c6604c800002a.csv", encoding="utf-8")
train_df, test_df = train_test_split(df, test_size=0.3, random_state=123)
train_df
```

Out[18]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge
1402	NE	70	415	421-8535	no	no	0	213.4	86	36.28	...	77	17.40
1855	WI	67	510	417-2265	no	no	0	109.1	134	18.55	...	76	12.10
633	NJ	122	415	327-9341	no	yes	34	146.4	104	24.89	...	103	7.62
1483	NV	107	510	419-9688	yes	no	0	234.1	91	39.80	...	105	13.86
2638	HI	105	510	364-8128	no	no	0	125.4	116	21.32	...	95	22.23
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2154	WY	126	408	339-9798	yes	no	0	197.6	126	33.59	...	112	20.95
3089	WV	70	510	348-3777	no	yes	30	143.4	72	24.38	...	92	14.45
1766	NJ	125	415	406-6400	no	no	0	182.3	64	30.99	...	121	11.88
1122	NE	159	415	362-5111	no	no	0	189.1	105	32.15	...	147	20.92
1346	PA	106	408	403-9167	yes	no	0	133.7	45	22.73	...	107	15.96

2333 rows × 21 columns

2.1 Distribution of target values

rubric={points:4}

Your tasks:

Examine the distribution of target values in the train split. Do you see class imbalance? If yes, do we need to deal with it? Why or why not?

BEGIN SOLUTION

In [19]:
```
train_df["churn"].value_counts(normalize=True)
```

Out[19]:
```
False    0.850407
True     0.149593
Name: churn, dtype: float64
```

Yes, there is class imbalance. Only around 15% of the examples in the training set belong to the "True" class, which is of our interest. We are likely to be interested in finding churned customers so that the company can

offer them promotions etc and so predicting "True" examples correctly is more important to us.

END SOLUTION

(Optional) 2.2 EDA

rubric={points:1}

Your tasks:

Come up with **two** exploratory questions you would like to answer and explore those. Briefly discuss your results in 1-3 sentences.

You are welcome to use `pandas_profiling` (see Lecture 10) but you don't have to.

BEGIN SOLUTION

In [20]:
```
cor = train_df.corr()
cor
```

Out[20]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	to nig minu
account length	1.000000	-0.001182	-0.014392	0.004172	0.041893	0.004172	-0.007104	0.007933	-0.007079	-0.0223
area code	-0.001182	1.000000	-0.010420	-0.018004	-0.000605	-0.018000	0.008661	0.004977	0.008698	0.0070
number vmail messages	-0.014392	-0.010420	1.000000	-0.011443	-0.001229	-0.011445	-0.006903	0.001281	-0.006877	-0.0156
total day minutes	0.004172	-0.018004	-0.011443	1.000000	-0.005629	1.000000	0.023442	0.031951	0.023441	0.0078
total day calls	0.041893	-0.000605	-0.001229	-0.005629	1.000000	-0.005626	-0.015018	0.009994	-0.015008	0.0300
total day charge	0.004172	-0.018000	-0.011445	1.000000	-0.005626	1.000000	0.023457	0.031956	0.023456	0.0078
total eve minutes	-0.007104	0.008661	-0.006903	0.023442	-0.015018	0.023457	1.000000	-0.000852	1.000000	-0.0255
total eve calls	0.007933	0.004977	0.001281	0.031951	0.009994	0.031956	-0.000852	1.000000	-0.000838	0.0050
total eve charge	-0.007079	0.008698	-0.006877	0.023441	-0.015008	0.023456	1.000000	-0.000838	1.000000	-0.0255
total night minutes	-0.022323	0.007058	-0.015654	0.007811	0.030066	0.007816	-0.025507	0.005060	-0.025504	1.0000
total night calls	-0.018192	0.038585	0.005661	0.015868	-0.007456	0.015864	0.013250	0.002709	0.013255	0.0060
total night charge	-0.022311	0.007019	-0.015692	0.007758	0.030064	0.007762	-0.025524	0.005108	-0.025521	0.9999
total intl minutes	0.012221	-0.024145	0.007540	-0.005937	0.005074	-0.005940	-0.007664	0.013521	-0.007682	-0.0218
total intl calls	0.007324	-0.032345	0.005228	-0.002756	0.005216	-0.002758	-0.008652	0.027803	-0.008652	-0.0223
total intl charge	0.012280	-0.024228	0.007516	-0.005812	0.005156	-0.005814	-0.007760	0.013485	-0.007778	-0.0218
customer service calls	-0.006801	0.025106	0.000817	-0.021350	-0.020813	-0.021356	-0.026125	0.011812	-0.026142	-0.0094
churn	0.030816	-0.007699	-0.079242	0.215746	0.016546	0.215749	0.087715	0.024840	0.087705	0.0266

In [21]:

```
possibly_most_relevant = [
    "total day minutes",
    "total day charge",
    "total eve minutes",
    "total eve charge",
    "total intl minutes",
    "customer service calls",
    "churn",
```

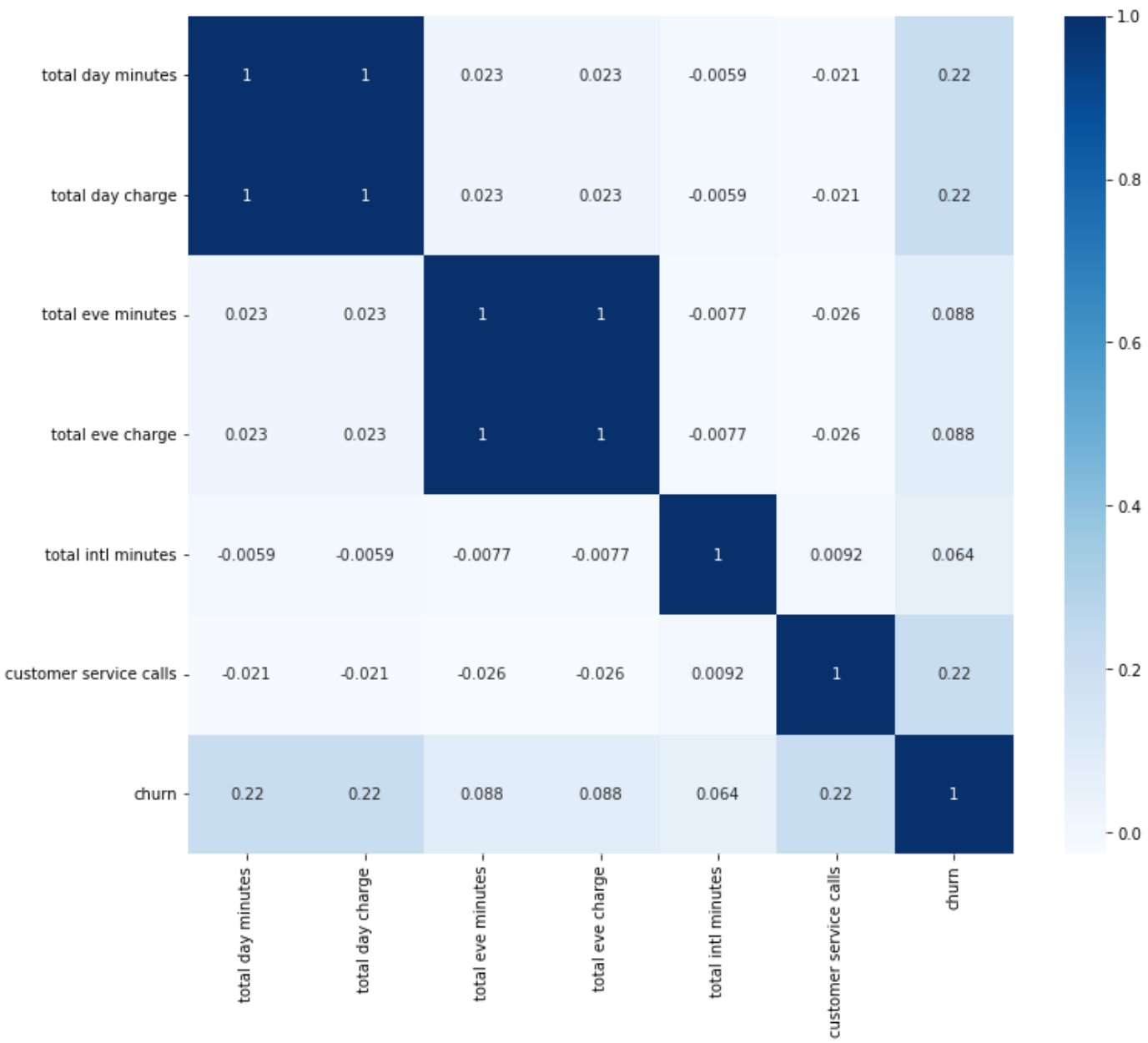
```
]
cor = train_df[possibly_most_relevant].corr()
cor
```

Out[21]:

	total day minutes	total day charge	total eve minutes	total eve charge	total intl minutes	customer service calls	churn
total day minutes	1.000000	1.000000	0.023442	0.023441	-0.005937	-0.021350	0.215746
total day charge	1.000000	1.000000	0.023457	0.023456	-0.005940	-0.021356	0.215749
total eve minutes	0.023442	0.023457	1.000000	1.000000	-0.007664	-0.026125	0.087715
total eve charge	0.023441	0.023456	1.000000	1.000000	-0.007682	-0.026142	0.087705
total intl minutes	-0.005937	-0.005940	-0.007664	-0.007682	1.000000	0.009233	0.064230
customer service calls	-0.021350	-0.021356	-0.026125	-0.026142	0.009233	1.000000	0.216641
churn	0.215746	0.215749	0.087715	0.087705	0.064230	0.216641	1.000000

In [22]:

```
import seaborn as sns

plt.figure(figsize=(12, 10))
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues)
plt.show()
```



We see that we have some colinear features in the data (examples: total day minutes and total day charge).

END SOLUTION

2.3 Column transformer

rubric={points:10}

The code below creates `X_train`, `y_train`, `X_test`, `y_test` for you. In preparation for building a classifier, set up a `ColumnTransformer` that performs whatever feature transformations you deem sensible. This can include dropping features if you think they are not helpful. Remember that by default `ColumnTransformer` will drop any columns that aren't accounted for when it's created.

In each case, briefly explain your rationale with 1-2 sentences. You do not need an explanation for every feature, but for every group of features that are being transformed the same way. For example, "I am doing transformation X to the following categorical features: `a`, `b`, `c` because of reason Y," etc.

In [23]:
```
X_train = train_df.drop(columns=["churn"])
X_test = test_df.drop(columns=["churn"])

y_train = train_df["churn"]
y_test = test_df["churn"]
```

BEGIN SOLUTION

In [24]:
```
train_df.head()
```

Out[24]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge
1402	NE	70	415	421-8535	no	no	0	213.4	86	36.28	...	77	17.40
1855	WI	67	510	417-2265	no	no	0	109.1	134	18.55	...	76	12.10
633	NJ	122	415	327-9341	no	yes	34	146.4	104	24.89	...	103	7.62
1483	NV	107	510	419-9688	yes	no	0	234.1	91	39.80	...	105	13.86
2638	HI	105	510	364-8128	no	no	0	125.4	116	21.32	...	95	22.23

5 rows × 21 columns

In [25]:
```
train_df["international plan"].unique()
```

Out[25]:
```
array(['no', 'yes'], dtype=object)
```

In [26]:
```
train_df["voice mail plan"].unique()
```

Out[26]:
```
array(['no', 'yes'], dtype=object)
```

In [27]:
```
train_df["area code"].unique()
```

Out[27]:
```
array([415, 510, 408])
```

In [28]:
```
train_df["state"].unique()
```

Out[28]:
```
array(['NE', 'WI', 'NJ', 'NV', 'HI', 'NM', 'MO', 'MI', 'SD', 'WY', 'GA',
       'WV', 'KS', 'DC', 'CT', 'TN', 'AK', 'WA', 'MS', 'CA', 'ND', 'MD',
       'AZ', 'RI', 'SC', 'AL', 'MA', 'ME', 'AR', 'VA', 'FL', 'NY', 'DE',
       'NH', 'MN', 'KY', 'NC', 'OK', 'OR', 'ID', 'IN', 'LA', 'UT', 'PA',
       'TX', 'OH', 'MT', 'IA', 'VT', 'IL', 'CO'], dtype=object)
```

In [29]:
```
train_df["state"].value_counts()
```

Out[29]:
```
WV    70
NY    64
OR    59
TX    59
OH    58
MN    58
MI    57
CT    54
MT    54
WY    54
WI    54
MD    53
AL    51
ID    51
VT    51
NJ    50
AZ    50
ME    49
UT    48
RI    48
FL    48
MA    47
ND    47
DE    46
NC    45
OK    45
AR    44
KS    44
MS    43
VA    43
MO    43
NM    43
IN    42
GA    41
KY    41
AK    40
NH    40
IL    39
CO    39
SC    39
SD    39
NV    39
TN    38
DC    38
NE    38
WA    34
PA    34
HI    33
LA    31
IA    30
CA    28
Name: state, dtype: int64
```

In [30]:
```
len(train_df["phone number"].unique())
```

Out[30]:
```
2333
```

The plan:

1. One-hot encode `area code`, `international plan` and `voice mail plan`, `state`. Note that `international plan` and `voice mail plan` and binary. But, if we use `drop='if_binary'` then a one-hot encoding is equivalent to a "binary encoding" because we just get one column that is either 0 or 1.

2. Drop `phone number`, as it is a unique identifier that is unlikely to be useful.
3. Treat the rest as numeric and standardize them.

In [31]:
```
categorical_features = ["area code", "state"]
binary_features = ["international plan", "voice mail plan"]
drop_features = ["phone number"]
numeric_features = list(
    set(train_df.columns)
    - set(["churn"])
    - set(binary_features)
    - set(categorical_features)
    - set(drop_features)
)
```

In [32]:
```
preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    (OneHotEncoder(drop="if_binary"), binary_features),
    (OneHotEncoder(handle_unknown="ignore", sparse=False), categorical_features),
    ("drop", drop_features),
)
```

END SOLUTION

2.4 Visualizing the transformed data

rubric={points:4}

Fit and transform your `ColumnTransformer` on your training set. Print the first 5 rows of the transformed data as a dataframe (not numpy array). See lecture 10 for code that can get you the new column names after transforming.

BEGIN SOLUTION

In [33]:
```
preprocessor.fit(X_train);
```

In [34]:
```
preprocessor.named_transformers_
```

Out[34]:
```
{'standardscaler': StandardScaler(),
 'onehotencoder-1': OneHotEncoder(drop='if_binary'),
 'onehotencoder-2': OneHotEncoder(handle_unknown='ignore', sparse=False),
 'drop': 'drop'}
```

In [35]:
```
new_columns = (
    numeric_features
    + binary_features
    + list(
        preprocessor.named_transformers_["onehotencoder-2"].get_feature_names_out(
            categorical_features
        )
    )
)
```

In [36]:
```
X_train_enc = pd.DataFrame(
    preprocessor.transform(X_train), index=X_train.index, columns=new_columns
```

```
)
```

In [37]:
```
X_train_enc.head()
```

Out[37]:

	customer service calls	number vmail messages	total night minutes	total eve calls	total intl calls	total day calls	total eve minutes	total night charge	total intl charge	total night calls
1402	-0.415269	-0.587624	1.088667	-1.156734	-0.200722	-0.721211	0.069871	1.089926	-1.644994	0.052115
1855	0.337507	-0.587624	-2.162302	-1.207278	0.198158	1.655252	-1.167277	-2.164029	0.222249	-0.720990
633	0.337507	1.900976	0.369287	0.157417	-0.200722	0.169963	-2.210130	0.369252	1.916105	-0.463288
1483	-0.415269	-0.587624	1.597736	0.258506	-0.599603	-0.473663	-0.754894	1.596582	-0.097850	0.000574
2638	0.337507	-0.587624	0.793839	-0.246937	1.793679	0.764078	1.195994	0.792921	0.408973	0.206736

5 rows × 71 columns

END SOLUTION

2.5 area code feature

rubric={points:4}

The original dataset had a feature called `area code`. Let's assume we encoded this feature with one-hot encoding.

1. The area codes were numbers to begin with. Why do we want to use one-hot encoding on this feature?
2. What were the possible values of `area code`?
3. What new feature(s) were created to replace `area code`?

BEGIN SOLUTION

(1) We use OHE because the actual numeric values of the area codes are meaningless - they are categories.

(2) Here are the possible values:

In [38]:
```
X_train["area code"].unique()
```

Out[38]:
```
array([415, 510, 408])
```

(3) Here are the new features:

In [39]:
```
area_code_cols = X_train_enc.columns[X_train_enc.columns.str.startswith("area code")]
area_code_cols
```

Out[39]:
```
Index(['area code_408', 'area code_415', 'area code_510'], dtype='object')
```

(4) Here are the new representations (since we didn't use `drop='first'` here:

- '408' is represented as [1,0,0]
- '415' is represented as [0,1,0]
- '510' is represented as [1,0,0]

In [40]:
```
X_train_enc.loc[X_train["area code"] == 408][area_code_cols].iloc[0].values
```

Out[40]:
```
array([1., 0., 0.])
```

In [41]:
```
X_train_enc.loc[X_train["area code"] == 415][area_code_cols].iloc[0].values
```

Out[41]:
```
array([0., 1., 0.])
```

In [42]:
```
X_train_enc.loc[X_train["area code"] == 510][area_code_cols].iloc[0].values
```

Out[42]:
```
array([0., 0., 1.])
```

END SOLUTION

2.6 Dummy classifier

rubric={points:4}

Your tasks:

Create a `DummyClassifier`. Report the following scoring metrics via cross-validation: accuracy, precision, recall, f1-score. Briefly comment on your results, including any *warnings* the code produces (2 sentences max).

BEGIN SOLUTION

In [43]:
```
dc = DummyClassifier()
```

In [44]:
```
score_types = ["accuracy", "precision", "recall", "f1"]
```

In [45]:
```
for score_type in score_types:
    print(
        "%-9s: %.2f"
        % (score_type, cross_val_score(dc, X_train, y_train, scoring=score_type).mean())
    )
```

```
accuracy : 0.85
precision: 0.00
recall   : 0.00
f1       : 0.00
```

```
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/home/moveisi/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

We get a warning because `DummyClassifier` always predicts False (no churn), so the precision is undefined.

END SOLUTION

2.7 Logistic regression

rubric={points:8}

Your tasks:

1. Train and score a logistic regression classifier on the dataset.
2. Report the same metrics as in the previous part.
3. Are you satisfied with the results? Use your `DummyClassifier` results as a reference point. Discuss in a few sentences.

BEGIN SOLUTION

In [46]:
```python
pipe = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
```

In [47]:
```python
pd.DataFrame(cross_validate(pipe, X_train, y_train, return_train_score=True)).mean()
```

Out[47]:
```
fit_time       0.088277
score_time     0.013618
test_score     0.855978
train_score    0.866481
dtype: float64
```

In [48]:
```python
for score_type in score_types:
    print(
        "%-9s: %.2f"
        % (
            score_type,
```

```
            cross_val_score(pipe, X_train, y_train, scoring=score_type).mean(),
        )
    )
```

```
accuracy : 0.86
precision: 0.55
recall   : 0.21
f1       : 0.30
```

These results above are surprisingly similar to what we got with `DummyClassifier`. The accuracy is roughly the same. While the precision is now better (and at least defined!), the recall and F1-scores are still very poor. The low recall indicates that we're hardly ever predicting True. This is likely due to the class imbalance.

END SOLUTION

2.8 Logistic regression with `class_weight`

rubric={points:6}

Your tasks:

1. Set the `class_weight` parameter of your logistic regression model to `'balanced'` and report the same metrics as in the previous part.
2. Do you prefer this model to the one in the previous part? Discuss your results in a few sentences.

BEGIN SOLUTION

In [49]:
```
pipe_balanced = make_pipeline(
    preprocessor, LogisticRegression(max_iter=1000, class_weight="balanced")
)
```

In [50]:
```
pd.DataFrame(
    cross_validate(pipe_balanced, X_train, y_train, return_train_score=True)
).mean()
```

Out[50]:
```
fit_time       0.061446
score_time     0.009211
test_score     0.760807
train_score    0.776252
dtype: float64
```

In [51]:
```
for score_type in score_types:
    print(
        "%-9s: %.2f"
        % (
            score_type,
            cross_val_score(pipe_balanced, X_train, y_train, scoring=score_type).mean(),
        )
    )
```

```
accuracy : 0.76
precision: 0.35
recall   : 0.70
f1       : 0.47
```

Our accuracy is now lower, but the recall is now much better. The F1-score is better now as well.

END SOLUTION

2.9 Hyperparameter optimization

rubric={points:10}

Now let's tune the hyperparameters of our `LogisticRegression` using `GridSearchCV` to maximize cross-validation f1 score.

Your tasks:

1. Jointly optimize `C` (choose some reasonable values) and `class_weight` (`None` vs. `'balanced'`) with `GridSearchCV` and `scoring="f1"`.
2. What values of `C` and `class_weight` are chosen and what is the best cross-validation f1 score?

BEGIN SOLUTION

In [52]:
```
pipe = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
```

In [53]:
```
param_grid = {
    "logisticregression__C": 2.0 ** np.arange(-4, 4),
    "logisticregression__class_weight": [None, "balanced"],
}
```

In [54]:
```
search = GridSearchCV(
    pipe,
    param_grid,
    verbose=1,
    n_jobs=-1,
    return_train_score=True,
    scoring="f1",
)
```

In [55]:
```
search.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

In [56]:
```
grid_results_df = pd.DataFrame(search.cv_results_)[
    [
        "mean_test_score",
        "mean_train_score",
        "param_logisticregression__C",
        "param_logisticregression__class_weight",
        "rank_test_score",
    ]
]
```

```
grid_results_df = grid_results_df.sort_values(by="mean_test_score", ascending=False)
grid_results_df
```

Out[56]:

	mean_test_score	mean_train_score	param_logisticregression__C	param_logisticregression__class_weight	rank_tes
1	0.481077	0.496341	0.0625	balanced	
3	0.477458	0.503381	0.125	balanced	
5	0.475169	0.507852	0.25	balanced	
7	0.469676	0.513824	0.5	balanced	
9	0.467027	0.512418	1.0	balanced	
11	0.465481	0.512885	2.0	balanced	
15	0.461991	0.513988	8.0	balanced	
13	0.461813	0.514124	4.0	balanced	
14	0.312894	0.371034	8.0	None	
12	0.309597	0.372006	4.0	None	
8	0.302739	0.355603	1.0	None	
10	0.302670	0.366568	2.0	None	
6	0.280559	0.343734	0.5	None	
4	0.275942	0.330673	0.25	None	
2	0.245554	0.295233	0.125	None	
0	0.220725	0.244659	0.0625	None	

END SOLUTION

2.10 Test results

rubric={points:10}

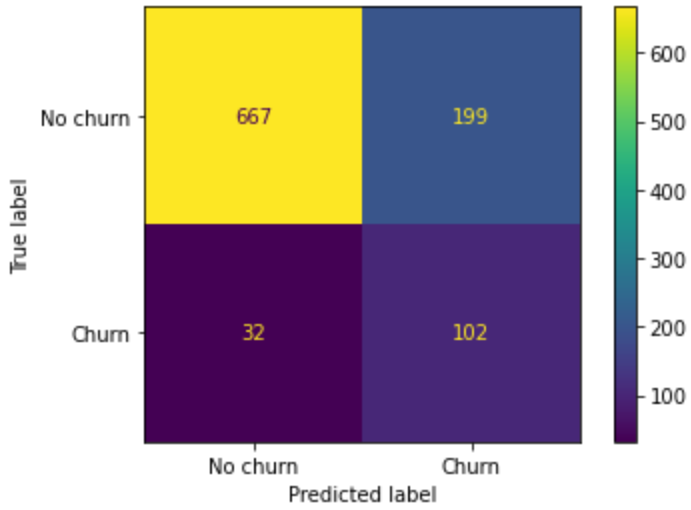
Your tasks

1. Evaluate the best model on the test set. In particular show each of the following on the test set:
   - Confusion matrix.
   - Classification report.
   - Precision-recall curve with average precision score.
   - ROC curve with AUC.
2. Comment on the results.

> Note that we are not doing it here but in real life, you would also plot confusion matrix, precision-recall curve, and ROC curve on validation data to examine errors and to choose a threshold which works for your operating point.

BEGIN SOLUTION

In [57]:
```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

disp = ConfusionMatrixDisplay.from_estimator(
    search, X_test, y_test, display_labels=["No churn", "Churn"]
)  # recommended approach in scikit-learn 1.0
```

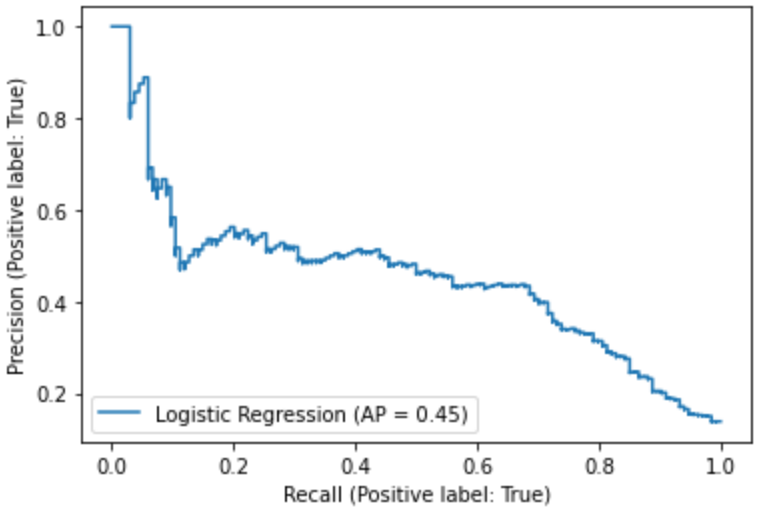


In [58]:
```
print(
    classification_report(
        y_test, search.predict(X_test), target_names=["No churn", "Churn"]
    )
)
```

```
              precision    recall  f1-score   support

    No churn       0.95      0.77      0.85       866
       Churn       0.34      0.76      0.47       134

    accuracy                           0.77      1000
   macro avg       0.65      0.77      0.66      1000
weighted avg       0.87      0.77      0.80      1000
```

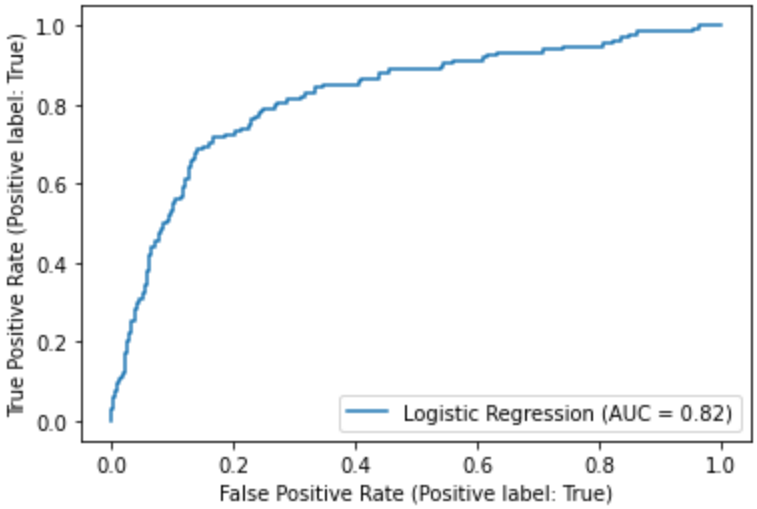
In [59]:
```
from sklearn.metrics import PrecisionRecallDisplay

PrecisionRecallDisplay.from_estimator(
    search, X_test, y_test, name="Logistic Regression"
)
# recommended approach in scikit-learn 1.0
```

Out[59]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at 0x7f35c1928f10>
```



In [60]:
```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(search, X_test, y_test, name="Logistic Regression")
# recommended approach in scikit-learn 1.0
```

Out[60]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f35c16f1870>
```



In [61]:
```
from sklearn.metrics import roc_auc_score

print("AUC score: %0.3f" % (roc_auc_score(y_test, search.predict_proba(X_test)[:, 1])))
```

```
AUC score: 0.821
```

Some observations:

1. The best hyperparameters found by our grid search are: C = 0.0625 and class_weight="balanced" with a validation f1-score of 0.48. The test f1 score is 0.47 which is comparable to the validation score. So seems like we did not end up overfitting the validation set.
2. The average precision score is 0.45 and seems like we have a reasonable compromise around recall value of ~0.70.
3. AUC is 0.82. So on average about 82% of the time positive examples have a bigger score compared to the negative examples.

END SOLUTION

Exercise 3: Regression metrics

For this exercise, we'll use California housing dataset from `sklearn datasets`. The code below loads the dataset.

In [62]:
```
from sklearn.datasets import fetch_california_housing

housing_df = fetch_california_housing(as_frame=True).frame
```

3.1: Data spitting and exploration

rubric={points:4}

Your tasks:

1. Split the data into train (80%) and test (20%) splits.
2. Explore the train split. Do you need to apply any transformations on the data? If yes, create a preprocessor with the appropriate transformations.
3. Separate `X` and `y` in train and test splits.

BEGIN SOLUTION

In [63]:
```
train_df, test_df = train_test_split(housing_df, test_size=0.2, random_state=123)
train_df.head()
```

Out[63]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
9950	4.5694	28.0	6.219512	1.030488	504.0	3.073171	38.38	-122.33	2.875
3547	5.6392	18.0	5.951644	1.034816	3010.0	2.911025	34.26	-118.60	2.715
4448	1.7292	47.0	3.628032	1.032345	1452.0	3.913747	34.07	-118.21	1.917
6984	4.6226	36.0	5.126238	0.985149	988.0	2.445545	33.96	-118.02	2.197
4432	2.4375	49.0	4.024390	0.942073	1405.0	4.283537	34.08	-118.20	1.140

In [64]:
```
train_df.describe()
```

Out[64]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Long
count	16512.000000	16512.000000	16512.000000	16512.000000	16512.000000	16512.000000	16512.000000	16512.0
mean	3.870716	28.570918	5.422508	1.096515	1434.115734	3.064722	35.627757	-119.5
std	1.900132	12.582281	2.520931	0.491311	1130.309986	10.624706	2.134543	2.0
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.750000	32.540000	-124.3
25%	2.562500	18.000000	4.430476	1.005825	789.000000	2.429013	33.930000	-121.8
50%	3.529400	29.000000	5.227170	1.048765	1170.000000	2.816122	34.250000	-118.4
75%	4.750375	37.000000	6.051005	1.099493	1735.000000	3.278954	37.710000	-118.0
max	15.000100	52.000000	141.909091	34.066667	28566.000000	1243.333333	41.950000	-114.3

All our features are numeric. The only transformation which seems relevant here is scaling.

In [65]:
```
preprocessor = StandardScaler()
```

In [66]:
```
X_train, y_train = train_df.drop(columns=["MedHouseVal"]), train_df["MedHouseVal"]
X_test, y_test = test_df.drop(columns=["MedHouseVal"]), test_df["MedHouseVal"]
```

END SOLUTION

3.2 Baseline: DummyRegressor

rubric={points:2}

Your tasks:

1. Carry out cross-validation using `DummyRegressor` with default scoring.
2. What metric is used for scoring by default?

BEGIN SOLUTION

In [67]:
```
dr = DummyRegressor()
pd.DataFrame(cross_validate(dr, X_train, y_train, return_train_score=True)).mean()
```

Out[67]:
```
fit_time       0.001610
score_time     0.000676
test_score    -0.000147
train_score    0.000000
dtype: float64
```

By default we get R^2 score.

END SOLUTION

3.3 Different regressors

rubric={points:8}

In this exercise, we are going to use `RandomForestRegressor` model which we haven't looked into yet. At this point you should feel comfortable using models with our usual ML workflow even if you don't know the details. We'll talk about `RandomForestRegressor` later in the course.

The code below defines a custom scorer called `mape_scorer` and creates dictionaries for different regressors (`models`) and different scoring metrics (`score_types_reg`).

Your tasks:

1. Using the `models` and the evaluation metrics `score_types_reg` in the code below, carry out cross-validation with each model, by passing the evaluation metrics to `scoring` argument of `cross_validate` . Use a pipeline with the model as an estimator if you are applying any transformations.
2. Show results as a dataframe.
3. Interpret the results. How do the models compare to the baseline? Which model seems to be performing well with different metrics?

In [68]:
```
def mape(true, pred):
    return 100.0 * np.mean(np.abs((pred - true) / true))


# make a scorer function that we can pass into cross-validation
mape_scorer = make_scorer(mape, greater_is_better=False)

models = {
    "Ridge": Ridge(),
    "Random Forest": RandomForestRegressor(),
}

score_types_reg = {
    "neg_mean_squared_error": "neg_mean_squared_error",
    "neg_root_mean_squared_error": "neg_root_mean_squared_error",
    "neg_mean_absolute_error": "neg_mean_absolute_error",
    "r2": "r2",
    "mape_scorer": mape_scorer,
}
```

BEGIN SOLUTION

In [69]:
```
def score_with_metrics(models, preprocessor, scoring="f1"):
    """
    Return cross-validation scores for given models as a dataframe.

    Parameters
    ----------
    models : dict
        a dictionary with names and scikit-learn models
    scoring : list/dict/string
        scoring parameter values for cross-validation

    Returns
    ----------
```

```
        None

    """
    results_df = {}
    for (name, model) in models.items():
        pipeline = make_pipeline(preprocessor, model)
        scores = cross_validate(
            pipeline, X_train, y_train, return_train_score=True, scoring=scoring
        )
        df = pd.DataFrame(scores)
        results_df[name] = df.mean()
    return pd.DataFrame(results_df)
```

In [70]:
```
score_with_metrics(models, preprocessor, scoring=score_types_reg)
```

Out[70]:

	Ridge	Random Forest
fit_time	0.011845	7.217220
score_time	0.005637	0.091246
test_neg_mean_squared_error	-0.683433	-0.263069
train_neg_mean_squared_error	-0.524369	-0.036870
test_neg_root_mean_squared_error	-0.810381	-0.512796
train_neg_root_mean_squared_error	-0.724132	-0.192011
test_neg_mean_absolute_error	-0.535444	-0.334654
train_neg_mean_absolute_error	-0.531388	-0.125217
test_r2	0.481744	0.802273
train_r2	0.606300	0.972316
test_mape_scorer	-31.979551	-18.878607
train_mape_scorer	-31.762979	-7.074229

1. Both Ridge and RandomForestRegressor are performing better than the DummyRegressor.
2. Random forest model has the highest R^2 score, neg MAPE, neg RMSE and neg MSE. With this model on average we have 19% error on prediction. That said, it seems to be much slower compared to Ridge.

END SOLUTION

(Optional) 3.4 Hyperparameter optimization

rubric={points:1}

Your tasks:

1. Carry out hyperparameter optimization using `RandomizedSearchCV` and `Ridge` with the following `param_dist`. The `alpha` hyperparameter of `Ridge` controls the fundamental tradeoff. Choose the metric of your choice for hyperparameter optimization.
2. Are you getting better scores compared to the default values?

In [71]:
```
from scipy.stats import loguniform

param_dist = {"ridge__alpha": loguniform(1e-3, 1e3)}
```

BEGIN SOLUTION

In [72]:
```
pipe = make_pipeline(StandardScaler(), Ridge())

random_search_ridge = RandomizedSearchCV(
    pipe,
    param_distributions=param_dist,
    n_iter=50,
    cv=3,
    verbose=1,
    n_jobs=-1,
    scoring=mape_scorer,
    random_state=123,
)
random_search_ridge.fit(X_train, y_train)
```

Out[72]:
```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
RandomizedSearchCV(cv=3,
                   estimator=Pipeline(steps=[('standardscaler',
                                              StandardScaler()),
                                             ('ridge', Ridge())]),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'ridge__alpha': <scipy.stats._distn_infrastructure.rv_fr
ozen object at 0x7f35c1789c90>},
                   random_state=123,
                   scoring=make_scorer(mape, greater_is_better=False),
                   verbose=1)
```

We are not getting much different results with optimized `alpha` compared to the default `alpha` used in `sklearn`. This might be because Ridge is probably not the best model for this dataset. The train score is also quite high suggesting that the model is underfit.

END SOLUTION

3.5 Test results

rubric={points:4}

Your tasks:

1. Try the best model on the test set.
2. Briefly comment on the results. (1 to 2 sentences)

BEGIN SOLUTION

In [73]:
```
print("Random Search best hyperparameters: %s" % (random_search_ridge.best_params_))
print("Random Search best model score: %0.3f" % (random_search_ridge.best_score_))
print(
    "Train score on the full train set: %0.3f"
```

```
    % (random_search_ridge.score(X_train, y_train))
)
print(
    "Test score on the full test set: %0.3f"
    % (random_search_ridge.score(X_test, y_test))
)
```

```
Random Search best hyperparameters: {'ridge__alpha': 26.789982776216593}
Random Search best model score: -31.887
Train score on the full train set: -31.835
Test score on the full test set: -31.084
```

END SOLUTION

3.6 Model interpretation

rubric={points:4}

Ridge is a linear model and it learns coefficients associated with each feature during `fit()`.

Your tasks:

1. Visualize coefficients learned by the `Ridge` model above as a pandas dataframe with two columns: features and coefficients. If you attempted 3.4, use the `Ridge` model with best hyperparameters. Otherwise use the `Ridge` model with default hyperparameters.
2. Increasing which feature values would result in higher housing price?

BEGIN SOLUTION

In [74]:
```
df = pd.DataFrame(
    data={
        "features": X_train.columns,
        "coefficients": random_search_ridge.best_estimator_.named_steps["ridge"].coef_,
    }
)
df.sort_values("coefficients")
```

Out[74]:

	features	coefficients
6	Latitude	-0.870521
7	Longitude	-0.835697
2	AveRooms	-0.276697
5	AveOccup	-0.041684
4	Population	-0.006641
1	HouseAge	0.117318
3	AveBedrms	0.311874
0	MedInc	0.834711

Given this model and the dataset, increasing latitude and longitude would result in lower housing prices,

whereas increasing median income, and average bedrooms would result in higher housing prices.

END SOLUTION

Submission instructions

PLEASE READ: When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.