19_survival-analysis

June 25, 2022

CPSC 330
Applied Machine Learning

1 Lecture 19: Survival analysis

UBC 2022 Summer

Instructor: Mehrdad Oveisi

1.1 Imports

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn.model_selection import (
    cross_val_predict,
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import (
    FunctionTransformer,
    OneHotEncoder,
    OrdinalEncoder,
    StandardScaler,
)
```

```
plt.rcParams["font.size"] = 16

# does lifelines try to mess with this?
pd.options.display.max_rows = 10
```

[2]:
```
import lifelines
```

1.2 Learning objectives

- Explain the problem with treating right-censored data the same as “regular” data.
- Determine whether survival analysis is an appropriate tool for a given problem.
- Apply survival analysis in Python using the `lifelines` package.
- Interpret a survival curve, such as the Kaplan-Meier curve.
- Interpret the coefficients of a fitted Cox proportional hazards model.
- Make predictions for existing individuals and interpret these predictions.

1.3 Customer churn: our standard approach

- In hw5 you looked at a dataset about customer churn.
- In hw5, the dataset was interesting because it’s unbalanced (most customers stay). We used typical binary classification approach on the dataset.

- Today we’ll look at a different customer churn dataset, because it has a feature we need - time!
- We’ll **explore the time aspect of the dataset today**.

[3]:
```
df = pd.read_csv("data/WA_Fn-UseC_-Telco-Customer-Churn.csv")
train_df, test_df = train_test_split(df, random_state=123)
train_df.head()
```

[3]:
```
      customerID  gender  SeniorCitizen Partner Dependents  tenure  \
6464  4726-DLWQN    Male              1      No         No      50
5707  4537-DKTAL  Female              0      No         No       2
3442  0468-YRPXN    Male              0      No         No      29
3932  1304-NECVQ  Female              1      No         No       2
6124  7153-CHRBV  Female              0     Yes        Yes      57

     PhoneService MultipleLines InternetService OnlineSecurity  …  \
6464          Yes           Yes             DSL            Yes  …
5707          Yes            No             DSL             No  …
3442          Yes            No     Fiber optic             No  …
3932          Yes           Yes     Fiber optic             No  …
6124          Yes            No             DSL            Yes  …

     DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
6464               No          No         Yes              No  Month-to-month
5707               No          No          No              No  Month-to-month
3442              Yes         Yes         Yes             Yes  Month-to-month
```

```
3932              Yes         No          No              No  Month-to-month
6124              Yes        Yes          No              No        One year

     PaperlessBilling              PaymentMethod MonthlyCharges  TotalCharges  \
6464              Yes  Bank transfer (automatic)          70.35        3454.6
5707               No           Electronic check          45.55          84.4
3442              Yes    Credit card (automatic)          98.80        2807.1
3932              Yes           Electronic check          78.55        149.55
6124              Yes               Mailed check          59.30       3274.35

     Churn
6464    No
5707    No
3442    No
3932   Yes
6124    No

[5 rows x 21 columns]
```

We can treat this as a **binary classification** problem where we want to predict `Churn` (yes/no) from these other columns.

[4]:
```
train_df.shape
```

[4]:
```
(5282, 21)
```

[5]:
```
train_df["Churn"].value_counts()
```

[5]:
```
No     3912
Yes    1370
Name: Churn, dtype: int64
```

[6]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5282 entries, 6464 to 3582
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        5282 non-null   object
 1   gender            5282 non-null   object
 2   SeniorCitizen     5282 non-null   int64
 3   Partner           5282 non-null   object
 4   Dependents        5282 non-null   object
 5   tenure            5282 non-null   int64
 6   PhoneService      5282 non-null   object
 7   MultipleLines     5282 non-null   object
 8   InternetService   5282 non-null   object
 9   OnlineSecurity    5282 non-null   object
```

```
 10  OnlineBackup      5282 non-null   object
 11  DeviceProtection  5282 non-null   object
 12  TechSupport       5282 non-null   object
 13  StreamingTV       5282 non-null   object
 14  StreamingMovies   5282 non-null   object
 15  Contract          5282 non-null   object
 16  PaperlessBilling  5282 non-null   object
 17  PaymentMethod     5282 non-null   object
 18  MonthlyCharges    5282 non-null   float64
 19  TotalCharges      5282 non-null   object
 20  Churn             5282 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 907.8+ KB
```

Question: Does this mean there is **no missing data?**

Ok, let's try our usual approach:

[7]:
```
train_df["SeniorCitizen"].value_counts()
```

[7]:
```
0    4430
1     852
Name: SeniorCitizen, dtype: int64
```

[8]:
```
numeric_features = ["tenure", "MonthlyCharges", "TotalCharges"]
drop_features = ["customerID"]
passthrough_features = ["SeniorCitizen"]
target_column = ["Churn"]
# the rest are categorical
categorical_features = list(
    set(train_df.columns)
    - set(numeric_features)
    - set(passthrough_features)
    - set(drop_features)
    - set(target_column)
)
```

[9]:
```
preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    (OneHotEncoder(), categorical_features),
    ("passthrough", passthrough_features),
    ("drop", drop_features),
)
```

[10]:
```
preprocessor.fit(train_df);
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [10], in <cell line: 1>()
```

```
----> 1 preprocessor.fit(train_df)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/compose/
 ↪_column_transformer.py:642, in ColumnTransformer.fit(self, X, y)
    624 """Fit all transformers using X.
    625 
    626 Parameters
   (…)
    638     This estimator.
    639 """
    640 # we use fit_transform to make sure to set sparse_output_ (for which we
    641 # need the transformed data) to have consistent output type in predict
--> 642 self.fit_transform(X, y=y)
    643 return self

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/compose/
 ↪_column_transformer.py:675, in ColumnTransformer.fit_transform(self, X, y)
    672 self._validate_column_callables(X)
    673 self._validate_remainder(X)
--> 675 result = self._fit_transform(X, y, _fit_transform_one)
    677 if not result:
    678     self._update_fitted_transformers([])

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/compose/
 ↪_column_transformer.py:606, in ColumnTransformer._fit_transform(self, X, y,␣
 ↪func, fitted, column_as_strings)
    600 transformers = list(
    601     self._iter(
    602         fitted=fitted, replace_strings=True,␣
 ↪column_as_strings=column_as_strings
    603     )
    604 )
    605 try:
--> 606     return Parallel(n_jobs=self.n_jobs)(
    607         delayed(func)(
    608             transformer=clone(trans) if not fitted else trans,
    609             X=_safe_indexing(X, column, axis=1),
    610             y=y,
    611             weight=weight,
    612             message_clsname="ColumnTransformer",
    613             message=self._log_message(name, idx, len(transformers)),
    614         )
    615␣
 ↪        for idx, (name, trans, column, weight) in enumerate(transformers, 1)
    616     )
    617 except ValueError as e:
    618     if "Expected 2D array, got 1D array instead" in str(e):
```

```
File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py:
↪1041, in Parallel.__call__(self, iterable)
   1032 try:
   1033     # Only set self._iterating to True if at least a batch
   1034     # was dispatched. In particular this covers the edge
   (…)
   1038     # was very quick and its callback already dispatched all the
   1039     # remaining jobs.
   1040     self._iterating = False
-> 1041     if self.dispatch_one_batch(iterator):
   1042         self._iterating = self._original_iterator is not None
   1044     while self.dispatch_one_batch(iterator):

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py:
↪859, in Parallel.dispatch_one_batch(self, iterator)
    857     return False
    858 else:
--> 859     self._dispatch(tasks)
    860     return True

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py:
↪777, in Parallel._dispatch(self, batch)
    775 with self._lock:
    776     job_idx = len(self._jobs)
--> 777     job = self._backend.apply_async(batch, callback=cb)
    778     # A job can complete so quickly than its callback is
    779     # called before we get here, causing self._jobs to
    780     # grow. To ensure correct results ordering, .insert is
    781     # used (rather than .append) in the following line
    782     self._jobs.insert(job_idx, job)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/
↪_parallel_backends.py:208, in SequentialBackend.apply_async(self, func,␣
↪callback)
    206 def apply_async(self, func, callback=None):
    207     """Schedule a func to be run"""
--> 208     result = ImmediateResult(func)
    209     if callback:
    210         callback(result)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/
↪_parallel_backends.py:572, in ImmediateResult.__init__(self, batch)
    569 def __init__(self, batch):
    570     # Don't delay the application, to avoid keeping the input
    571     # arguments in memory
--> 572     self.results = batch()
```

```
File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py:
↪262, in BatchedCalls.__call__(self)
    258 def __call__(self):
    259     # Set the default nested backend to self._backend but do not set th
    260     # change the default number of processes to -1
    261     with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262         return [func(*args, **kwargs)
    263                 for func, args, kwargs in self.items]

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py:
↪262, in <listcomp>(.0)
    258 def __call__(self):
    259     # Set the default nested backend to self._backend but do not set th
    260     # change the default number of processes to -1
    261     with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262         return [func(*args, **kwargs)
    263                 for func, args, kwargs in self.items]

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/utils/fixes
↪py:216, in _FuncWrapper.__call__(self, *args, **kwargs)
    214 def __call__(self, *args, **kwargs):
    215     with config_context(**self.config):
--> 216         return self.function(*args, **kwargs)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/pipeline.py
↪893, in _fit_transform_one(transformer, X, y, weight, message_clsname,␣
↪message, **fit_params)
    891 with _print_elapsed_time(message_clsname, message):
    892     if hasattr(transformer, "fit_transform"):
--> 893         res = transformer.fit_transform(X, y, **fit_params)
    894     else:
    895         res = transformer.fit(X, y, **fit_params).transform(X)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:852 ␣
↪in TransformerMixin.fit_transform(self, X, y, **fit_params)
    848 # non-optimized default implementation; override when a better
    849 # method is possible for a given clustering algorithm
    850 if y is None:
    851     # fit method of arity 1 (unsupervised transformation)
--> 852     return self.fit(X, **fit_params).transform(X)
    853 else:
    854     # fit method of arity 2 (supervised transformation)
    855     return self.fit(X, y, **fit_params).transform(X)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/
↪preprocessing/_data.py:806, in StandardScaler.fit(self, X, y, sample_weight)
    804 # Reset internal state before fitting
    805 self._reset()
```

```
--> 806 return self.partial_fit(X, y, sample_weight)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/
 ↪preprocessing/_data.py:841, in StandardScaler.partial_fit(self, X, y,␣
 ↪sample_weight)
    809 """Online computation of mean and std on X for later scaling.
    810
    811 All of X is processed as a single batch. This is intended for cases
   (…)
    838     Fitted scaler.
    839 """
    840 first_call = not hasattr(self, "n_samples_seen_")
--> 841 X = self._validate_data(
    842     X,
    843     accept_sparse=("csr", "csc"),
    844     estimator=self,
    845     dtype=FLOAT_DTYPES,
    846     force_all_finite="allow-nan",
    847     reset=first_call,
    848 )
    849 n_features = X.shape[1]
    851 if sample_weight is not None:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:566 ␣
 ↪in BaseEstimator._validate_data(self, X, y, reset, validate_separately,␣
 ↪**check_params)
    564     raise ValueError("Validation should be done on X, y or both.")
    565 elif not no_val_X and no_val_y:
--> 566     X = check_array(X, **check_params)
    567     out = X
    568 elif no_val_X and not no_val_y:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/utils/
 ↪validation.py:746, in check_array(array, accept_sparse, accept_large_sparse,␣
 ↪dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples ␣
 ↪ensure_min_features, estimator)
    744         array = array.astype(dtype, casting="unsafe", copy=False)
    745     else:
--> 746         array = np.asarray(array, order=order, dtype=dtype)
    747 except ComplexWarning as complex_warning:
    748     raise ValueError(
    749         "Complex data not supported\n{}\n".format(array)
    750     ) from complex_warning

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/generic
 ↪py:2064, in NDFrame.__array__(self, dtype)
   2063 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064     return np.asarray(self._values, dtype=dtype)
```

```
ValueError: could not convert string to float: ''
```

Hmmm, one of the numeric features is causing problems?

[11]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Oh, looks like `TotalCharges` is not a numeric type. What if we change the type of this column to float?

[12]:
```python
train_df["TotalCharges"] = train_df["TotalCharges"].astype(float)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [12], in <cell line: 1>()
----> 1 train_df["TotalCharges"] = train_df["TotalCharges"].astype(float)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/generic
↪py:5912, in NDFrame.astype(self, dtype, copy, errors)
   5905     results = [
```

```
   5906         self.iloc[:, i].astype(dtype, copy=copy)
   5907         for i in range(len(self.columns))
   5908     ]
   5910 else:
   5911     # else, only a single dtype is given
-> 5912     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5913     return self._constructor(new_data).__finalize__(self, method="astype")
   5915 # GH 33113: handle empty frame or series

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/internals/managers.py:419, in BaseBlockManager.astype(self, dtype, copy, errors)
    418 def astype(self: T, dtype, copy: bool = False, errors: str = "raise") -> T:
--> 419     return self.apply("astype", dtype=dtype, copy=copy, errors=errors)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/internals/managers.py:304, in BaseBlockManager.apply(self, f, align_keys, ignore_failures, **kwargs)
    302         applied = b.apply(f, **kwargs)
    303     else:
--> 304         applied = getattr(b, f)(**kwargs)
    305 except (TypeError, NotImplementedError):
    306     if not ignore_failures:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/internals/blocks.py:580, in Block.astype(self, dtype, copy, errors)
    562 """
    563 Coerce to the new dtype.
    564
   (…)
    576 Block
    577 """
    578 values = self.values
--> 580 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
    582 new_values = maybe_coerce_values(new_values)
    583 newb = self.make_block(new_values)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/dtypes/cast.py:1292, in astype_array_safe(values, dtype, copy, errors)
   1289     dtype = dtype.numpy_dtype
   1291 try:
-> 1292     new_values = astype_array(values, dtype, copy=copy)
   1293 except (ValueError, TypeError):
   1294     # e.g. astype_nansafe can fail on object-dtype of strings
   1295     #  trying to convert to float
   1296     if errors == "ignore":
```

```
File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/dtypes/
  ↪cast.py:1237, in astype_array(values, dtype, copy)
   1234     values = values.astype(dtype, copy=copy)
   1236 else:
-> 1237     values = astype_nansafe(values, dtype, copy=copy)
   1239 # in pandas we don't store numpy str dtypes, so convert to object
   1240 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/dtypes/
  ↪cast.py:1181, in astype_nansafe(arr, dtype, copy, skipna)
   1177     raise ValueError(msg)
   1179 if copy or is_object_dtype(arr.dtype) or is_object_dtype(dtype):
   1180     # Explicit copy, or required since NumPy can't view from / to object.
-> 1181     return arr.astype(dtype, copy=True)
   1183 return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: ''
```

Argh!!

[13]:
```python
for val in train_df["TotalCharges"]:
    try:
        float(val)
    except ValueError:
        print(val)
```

Any ideas?

Well, it turns out we can't see those problematic values because they are whitespace!

[14]:
```python
for val in train_df["TotalCharges"]:
    try:
        float(val)
    except ValueError:
        print('"%s"' % val)
```

```
" "
" "
" "
```

```
" "
" "
" "
" "
" "
```

Let's replace the whitespaces with NaNs.

[15]:
```python
train_df = train_df.assign(
    TotalCharges=train_df["TotalCharges"].replace(" ", np.nan).astype(float)
)
test_df = test_df.assign(
    TotalCharges=test_df["TotalCharges"].replace(" ", np.nan).astype(float)
)
```

[16]:
```python
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5282 entries, 6464 to 3582
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        5282 non-null   object
 1   gender            5282 non-null   object
 2   SeniorCitizen     5282 non-null   int64
 3   Partner           5282 non-null   object
 4   Dependents        5282 non-null   object
 5   tenure            5282 non-null   int64
 6   PhoneService      5282 non-null   object
 7   MultipleLines     5282 non-null   object
 8   InternetService   5282 non-null   object
 9   OnlineSecurity    5282 non-null   object
 10  OnlineBackup      5282 non-null   object
 11  DeviceProtection  5282 non-null   object
 12  TechSupport       5282 non-null   object
 13  StreamingTV       5282 non-null   object
 14  StreamingMovies   5282 non-null   object
 15  Contract          5282 non-null   object
 16  PaperlessBilling  5282 non-null   object
 17  PaymentMethod     5282 non-null   object
 18  MonthlyCharges    5282 non-null   float64
 19  TotalCharges      5274 non-null   float64
 20  Churn             5282 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 907.8+ KB
```

But now we are going to have missing values and we need to include imputation for numeric features in our preprocessor.

[17]:
```
preprocessor = make_column_transformer(
    (
        make_pipeline(SimpleImputer(strategy="median"), StandardScaler()),
        numeric_features,
    ),
    (OneHotEncoder(handle_unknown="ignore"), categorical_features),
    ("passthrough", passthrough_features),
    ("drop", drop_features),
)
```

Now let's try that again...

[18]:
```
preprocessor.fit(train_df);
```

It worked! Let's get the column names of the transformed data from the column transformer.

[19]:
```
new_columns = (
    numeric_features
    + preprocessor.named_transformers_["onehotencoder"]
    .get_feature_names_out(categorical_features)
    .tolist()
    + passthrough_features
)
```

[20]:
```
X_train_enc = pd.DataFrame(
    preprocessor.transform(train_df), index=train_df.index, columns=new_columns
)
X_test_enc = pd.DataFrame(
    preprocessor.transform(train_df), index=train_df.index, columns=new_columns
)
```

[21]:
```
X_train_enc.head()
```

[21]:
```
        tenure  MonthlyCharges  TotalCharges  OnlineBackup_No  \
6464  0.707712        0.185175      0.513678              0.0
5707 -1.248999       -0.641538     -0.979562              1.0
3442 -0.148349        1.133562      0.226789              1.0
3932 -1.248999        0.458524     -0.950696              1.0
6124  0.993065       -0.183179      0.433814              1.0

      OnlineBackup_No internet service  OnlineBackup_Yes  StreamingMovies_No  \
6464                               0.0               1.0                 1.0
5707                               0.0               0.0                 1.0
3442                               0.0               0.0                 0.0
3932                               0.0               0.0                 1.0
6124                               0.0               0.0                 1.0

      StreamingMovies_No internet service  StreamingMovies_Yes  Dependents_No  \
```

```
6464                                    0.0                        0.0                 1.0
5707                                    0.0                        0.0                 1.0
3442                                    0.0                        1.0                 1.0
3932                                    0.0                        0.0                 1.0
6124                                    0.0                        0.0                 0.0

      …  PaymentMethod_Bank transfer (automatic)  \
6464  …                                      1.0
5707  …                                      0.0
3442  …                                      0.0
3932  …                                      0.0
6124  …                                      0.0

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
6464                                    0.0                             0.0
5707                                    0.0                             1.0
3442                                    1.0                             0.0
3932                                    0.0                             1.0
6124                                    0.0                             0.0

      PaymentMethod_Mailed check  PhoneService_No  PhoneService_Yes  \
6464                         0.0              0.0               1.0
5707                         0.0              0.0               1.0
3442                         0.0              0.0               1.0
3932                         0.0              0.0               1.0
6124                         1.0              0.0               1.0

      StreamingTV_No  StreamingTV_No internet service  StreamingTV_Yes  \
6464             0.0                              0.0              1.0
5707             1.0                              0.0              0.0
3442             0.0                              0.0              1.0
3932             1.0                              0.0              0.0
6124             1.0                              0.0              0.0

      SeniorCitizen
6464            1.0
5707            0.0
3442            0.0
3932            1.0
6124            0.0

[5 rows x 45 columns]
```

Before we look into survival analysis, let's just **treat it as a binary classification** model where we want to predict whether a customer churned or not.

[22]:
```
results = {}
```

[23]:
```python
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))

    return pd.Series(data=out_col, index=mean_scores.index)
```

[24]:
```python
X_train = train_df.drop(columns=["Churn"])
X_test = test_df.drop(columns=["Churn"])

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

1.3.1 DummyClassifier

[25]:
```python
dc = DummyClassifier()
```

[26]:
```python
results["dummy"] = mean_std_cross_val_scores(
    dc, X_train, y_train, return_train_score=True
)
pd.DataFrame(results)
```

[26]:

	dummy
fit_time	0.003 (+/- 0.001)
score_time	0.001 (+/- 0.000)

```
test_score   0.741 (+/- 0.000)
train_score  0.741 (+/- 0.000)
```

1.3.2 LogisticRegression

[27]:
```
lr = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
```

[28]:
```
results["logistic regression"] = mean_std_cross_val_scores(
    lr, X_train, y_train, return_train_score=True
)
pd.DataFrame(results)
```

[28]:
```
                         dummy logistic regression
fit_time     0.003 (+/- 0.001)   0.079 (+/- 0.012)
score_time   0.001 (+/- 0.000)   0.011 (+/- 0.003)
test_score   0.741 (+/- 0.000)   0.804 (+/- 0.013)
train_score  0.741 (+/- 0.000)   0.809 (+/- 0.002)
```

[29]:
```
confusion_matrix(y_train, cross_val_predict(lr, X_train, y_train))
```

[29]:
```
array([[3516,  396],
       [ 637,  733]])
```

1.3.3 RandomForestClassifier

[30]:
```
rf = make_pipeline(preprocessor, RandomForestClassifier())
```

[31]:
```
results["random forest"] = mean_std_cross_val_scores(
    rf, X_train, y_train, return_train_score=True
)
pd.DataFrame(results)
```

[31]:
```
                         dummy logistic regression      random forest
fit_time     0.003 (+/- 0.001)   0.079 (+/- 0.012)  0.311 (+/- 0.035)
score_time   0.001 (+/- 0.000)   0.011 (+/- 0.003)  0.026 (+/- 0.000)
test_score   0.741 (+/- 0.000)   0.804 (+/- 0.013)  0.788 (+/- 0.011)
train_score  0.741 (+/- 0.000)   0.809 (+/- 0.002)  0.998 (+/- 0.000)
```

[32]:
```
confusion_matrix(y_train, cross_val_predict(rf, X_train, y_train))
```

[32]:
```
array([[3536,  376],
       [ 723,  647]])
```

- This is was we did in hw5.
- What's wrong with this approach?

And now the rest of the class is about what is wrong with what we just did!

1.4 Censoring and survival analysis

1.4.1 Time to event and censoring

Imagine that you want to analyze *the time until an event occurs*. For example,

- the time until a disease kills its host.
- the time until a piece of equipment breaks.
- the time that someone unemployed will take to land a new job.
- the time until a customer leaves a subscription service (this dataset).

In our example, instead of predicting the binary label churn or no churn, it will be more useful to **when the customer is likely to churn** (the time until churn happens) so that we can take some action.

[33]:
```
train_df[["tenure"]].head()
```

[33]:

	tenure
6464	50
5707	2
3442	29
3932	2
6124	57

The tenure column is the number of months the customer has stayed with the company.

Although this branch of statistics is usually referred to as **Survival Analysis**, the event in question does not need to be related to actual "survival". The important thing is to understand that we are interested in **the time until something happens**, or whether or not something will happen in a certain time frame.

Question: But why is this different? Can't you just use the techniques you learned so far (e.g., regression models) to predict the time? Take a minute to think about this.

The answer would be yes if you could observe the actual time in all occurrences, but you usually cannot. Frequently, there will be some kind of **censoring** which will not allow you to observe the exact time that the event happened for all units/individuals that are being studied.

[34]:
```
train_df[["tenure", "Churn"]].head()
```

[34]:

	tenure	Churn
6464	50	No
5707	2	No
3442	29	No
3932	2	Yes
6124	57	No

- What this means is that we **don't have correct target values** to train or test our model.
- This is a problem!

Let's consider some approaches to deal with this censoring issue.

1.4.2 Approach 1: Only consider the examples where "Churn"=Yes

Let's just consider the cases *for which we have the time*, to obtain the average subscription length.

[35]:
```python
train_df_churn = train_df.query(
    "Churn == 'Yes'"
)  # Consider only examples where the customers churned.
test_df_churn = test_df.query(
    "Churn == 'Yes'"
)  # Consider only examples where the customers churned.
train_df_churn.head()
```

[35]:
```
      customerID  gender  SeniorCitizen Partner Dependents  tenure  \
3932  1304-NECVQ  Female              1      No         No       2
301   8098-LLAZX  Female              1      No         No       4
5540  3803-KMQFW  Female              0     Yes        Yes       1
4084  2777-PHDEI  Female              0      No         No       1
3272  6772-KSATR    Male              0      No         No       1

     PhoneService MultipleLines InternetService       OnlineSecurity  …  \
3932          Yes           Yes     Fiber optic                   No  …
301           Yes           Yes     Fiber optic                   No  …
5540          Yes            No              No  No internet service  …
4084          Yes            No     Fiber optic                   No  …
3272          Yes           Yes     Fiber optic                  Yes  …

         DeviceProtection          TechSupport          StreamingTV  \
3932                  Yes                   No                   No
301                    No                   No                  Yes
5540  No internet service  No internet service  No internet service
4084                   No                   No                  Yes
3272                   No                   No                   No

          StreamingMovies        Contract PaperlessBilling     PaymentMethod  \
3932                   No  Month-to-month              Yes  Electronic check
301                   Yes  Month-to-month              Yes  Electronic check
5540  No internet service  Month-to-month               No      Mailed check
4084                   No  Month-to-month               No  Electronic check
3272                   No  Month-to-month              Yes  Electronic check

     MonthlyCharges  TotalCharges  Churn
3932          78.55        149.55    Yes
301           95.45        396.10    Yes
5540          20.55         20.55    Yes
4084          78.05         78.05    Yes
3272          81.70         81.70    Yes

[5 rows x 21 columns]
```

[36]:
```python
train_df.shape
```

[36]: (5282, 21)

[37]:
```python
train_df_churn.shape
```

[37]: (1370, 21)

[38]:
```python
numeric_features
```

[38]: ['tenure', 'MonthlyCharges', 'TotalCharges']

[39]:
```python
preprocessing_notenure = make_column_transformer(
    (
        make_pipeline(SimpleImputer(strategy="median"), StandardScaler()),
        numeric_features[1:],  # Getting rid of the tenure column
    ),
    (OneHotEncoder(handle_unknown="ignore"), categorical_features),
    ("passthrough", passthrough_features),
)
```

[40]:
```python
tenure_lm = make_pipeline(preprocessing_notenure, Ridge())

tenure_lm.fit(train_df_churn.drop(columns=["tenure"]),␣
 ↪train_df_churn["tenure"]);
```

[41]:
```python
pd.DataFrame(
    tenure_lm.predict(test_df_churn.drop(columns=["tenure"]))[:10],
    columns=["tenure_predictions"],
)
```

[41]:

	tenure_predictions
0	5.062449
1	13.198645
2	11.859455
3	5.865562
4	58.154842
5	3.757932
6	18.932070
7	7.720893
8	36.818041
9	7.263541

What will be wrong with our estimated survival times? Will they be too low or too high?

On average they will be **underestimates** (too small), because we are ignoring the currently subscribed (un-churned) customers. Our dataset is a biased sample of those who churned within the time window of the data collection. **Long-time subscribers were more likely to be removed** from the dataset! This is a common mistake - see the Calling Bullshit video I posted on the README!

1.4.3 Approach 2: Assume everyone churns right now

Assume everyone churns right now - in other words, use the original dataset.

[42]:
```
train_df[["tenure", "Churn"]].head()
```

[42]:
```
      tenure Churn
6464      50    No
5707       2    No
3442      29    No
3932       2   Yes
6124      57    No
```

[43]:
```
tenure_lm.fit(train_df.drop(columns=["tenure"]), train_df["tenure"]);
```

[44]:
```
pd.DataFrame(
    tenure_lm.predict(test_df_churn.drop(columns=["tenure"]))[:10],
    columns=["tenure_predictions"],
)
```

[44]:
```
   tenure_predictions
0            6.400047
1           20.220392
2           22.332746
3           12.825470
4           59.885968
5            7.075453
6           17.731498
7           10.407862
8           38.425365
9           10.854500
```

What will be wrong with our estimated survival time?

[45]:
```
train_df[["tenure", "Churn"]].head()
```

[45]:
```
      tenure Churn
6464      50    No
5707       2    No
3442      29    No
3932       2   Yes
6124      57    No
```

It will be an **underestimate** again. For those still subscribed, while we did not remove them, we recorded a total tenure shorter than in reality, because they will keep going for some amount of time.

1.4.4 Approach 3: Survival analysis

Deal with this properly using survival analysis.

- You may learn about this in a statistics course.
- We will use the `lifelines` package in Python and will not go into the math/stats of how it works.

[46]:
```
train_df[["tenure", "Churn"]].head()
```

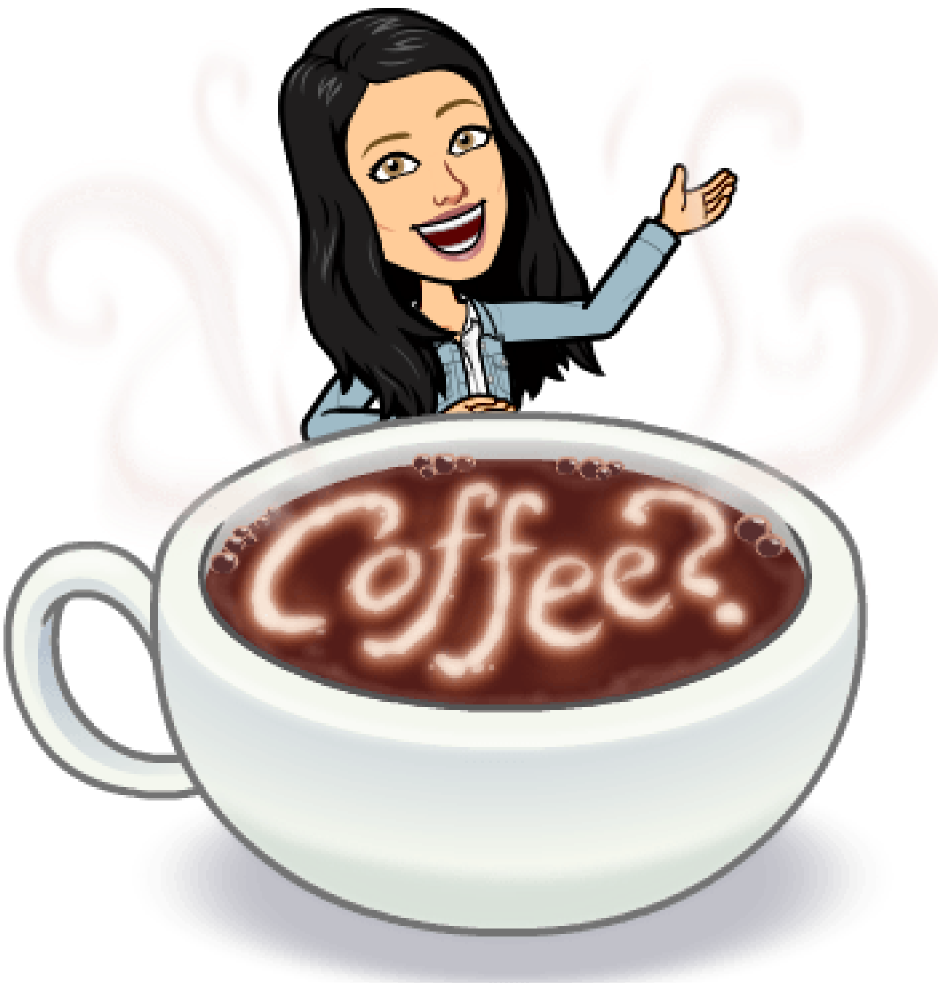
[46]:

	tenure	Churn
6464	50	No
5707	2	No
3442	29	No
3932	2	Yes
6124	57	No

Types of questions we might want to answer:

1. How long do customers stay with the service?
2. For a particular customer, can we predict how long they might stay with the service?
3. What factors influence a customer's churn time?

1.5 Break (5 min)



1.6 Kaplan-Meier survival curve

Before we do anything further, I want to modify our dataset slightly:

1. I'm going to **drop** the `TotalCharges` (yes, after all that work fixing it) because it's a bit of a strange feature.
    - Its value actually **changes over time**, but we only have the value at the end.
    - We still have `MonthlyCharges`.
2. I'm going to **not scale** the `tenure` column, since it will be convenient to keep it in its original units of months.

Just for our sanity, I'm redefining the features.

[47]:
```
numeric_features = ["MonthlyCharges"]
drop_features = ["customerID", "TotalCharges"]
passthrough_features = ["tenure", "SeniorCitizen"]  # don't want to scale tenure
target_column = ["Churn"]
```

```
# the rest are categorical
categorical_features = list(
    set(train_df.columns)
    - set(numeric_features)
    - set(passthrough_features)
    - set(drop_features)
    - set(target_column)
)
```

[48]:
```
preprocessing_final = make_column_transformer(
    (
        FunctionTransformer(lambda x: x == "Yes"),
        target_column,
    ),  # because we need it in this format for lifelines package
    ("passthrough", passthrough_features),
    (StandardScaler(), numeric_features),
    (OneHotEncoder(handle_unknown="ignore", sparse=False),␣
 ↪categorical_features),
    ("drop", drop_features),
)
```

[49]:
```
preprocessing_final.fit(train_df);
```

Let's get the column names of the columns created by our column transformer.

[50]:
```
new_columns = (
    target_column
    + passthrough_features
    + numeric_features
    + preprocessing_final.named_transformers_["onehotencoder"]
    .get_feature_names_out(categorical_features)
    .tolist()
)
```

[51]:
```
train_df_surv = pd.DataFrame(
    preprocessing_final.transform(train_df), index=train_df.index,␣
 ↪columns=new_columns
)
test_df_surv = pd.DataFrame(
    preprocessing_final.transform(test_df), index=test_df.index,␣
 ↪columns=new_columns
)
```

[52]:
```
train_df_surv.head()
```

[52]:
```
      Churn  tenure  SeniorCitizen  MonthlyCharges  OnlineBackup_No  \
6464    0.0    50.0            1.0        0.185175              0.0
5707    0.0     2.0            0.0       -0.641538              1.0
```

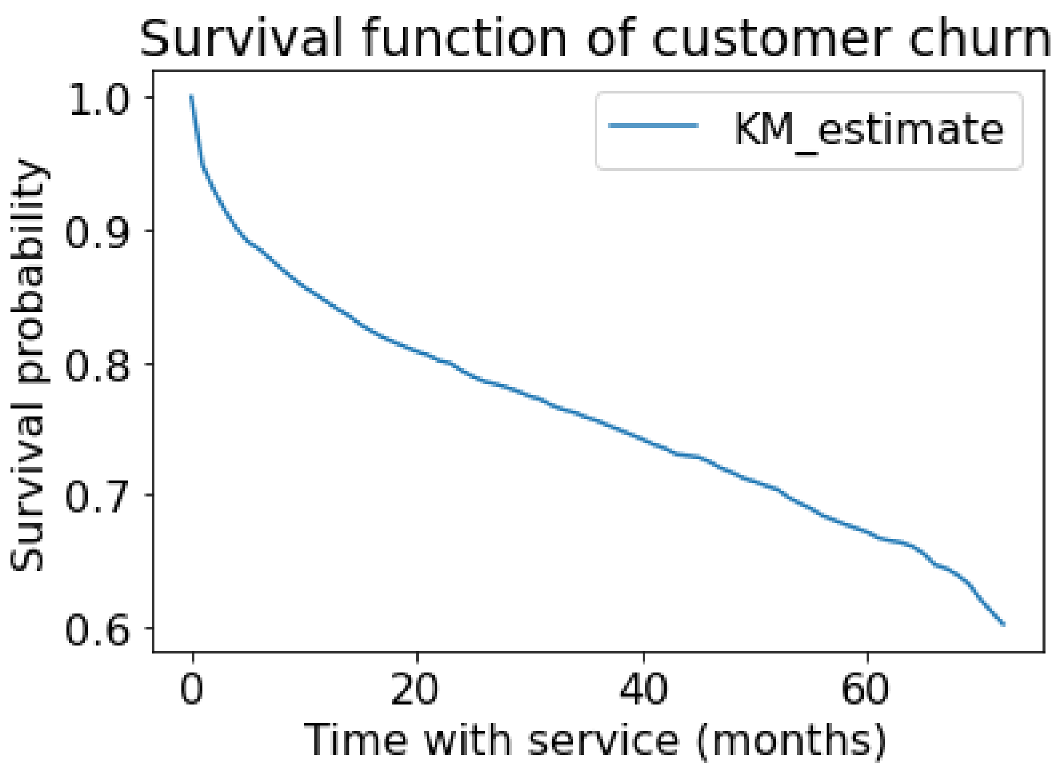
```
3442   0.0   29.0           0.0       1.133562           1.0
3932   1.0    2.0           1.0       0.458524           1.0
6124   0.0   57.0           0.0      -0.183179           1.0

     OnlineBackup_No internet service  OnlineBackup_Yes  StreamingMovies_No  \
6464                              0.0               1.0                 1.0
5707                              0.0               0.0                 1.0
3442                              0.0               0.0                 0.0
3932                              0.0               0.0                 1.0
6124                              0.0               0.0                 1.0

     StreamingMovies_No internet service  StreamingMovies_Yes  …  \
6464                                 0.0                  0.0  …
5707                                 0.0                  0.0  …
3442                                 0.0                  1.0  …
3932                                 0.0                  0.0  …
6124                                 0.0                  0.0  …

     gender_Male  PaymentMethod_Bank transfer (automatic)  \
6464         1.0                                      1.0
5707         0.0                                      0.0
3442         1.0                                      0.0
3932         0.0                                      0.0
6124         0.0                                      0.0

     PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
6464                                   0.0                             0.0
5707                                   0.0                             1.0
3442                                   1.0                             0.0
3932                                   0.0                             1.0
6124                                   0.0                             0.0

     PaymentMethod_Mailed check  PhoneService_No  PhoneService_Yes  \
6464                        0.0              0.0               1.0
5707                        0.0              0.0               1.0
3442                        0.0              0.0               1.0
3932                        0.0              0.0               1.0
6124                        1.0              0.0               1.0

     StreamingTV_No  StreamingTV_No internet service  StreamingTV_Yes
6464            0.0                              0.0              1.0
5707            1.0                              0.0              0.0
3442            0.0                              0.0              1.0
3932            1.0                              0.0              0.0
6124            1.0                              0.0              0.0

[5 rows x 45 columns]
```

- We'll start with a model called `KaplanMeierFitter` from `lifelines` package to get a Kaplan Meier curve.

- For this model we only use two columns: tenure and churn.
- We do not use any other features.

[53]:
```
kmf = lifelines.KaplanMeierFitter()
kmf.fit(train_df_surv["tenure"], train_df_surv["Churn"]);
```

[54]:
```
kmf.survival_function_.plot()
plt.title("Survival function of customer churn")
plt.xlabel("Time with service (months)")
plt.ylabel("Survival probability");
```



- What is this plot telling us?
- It shows the probability of survival over time.
- For example, after 20 months the probability of survival is ~0.8.
- Over time it's going down.

[55]:
```
np.mean(y_train == "No"), np.mean(y_train == "Yes")
```

[55]:
```
(0.7406285497917455, 0.2593714502082545)
```

What's the average tenure?

[56]:
```python
np.mean(train_df_surv["tenure"])
```

[56]: 32.6391518364256

What's the average tenure of the people who churned?

[57]:
```python
np.mean(train_df_surv.query("Churn == 1.0")["tenure"])
```

[57]: 17.854744525547446

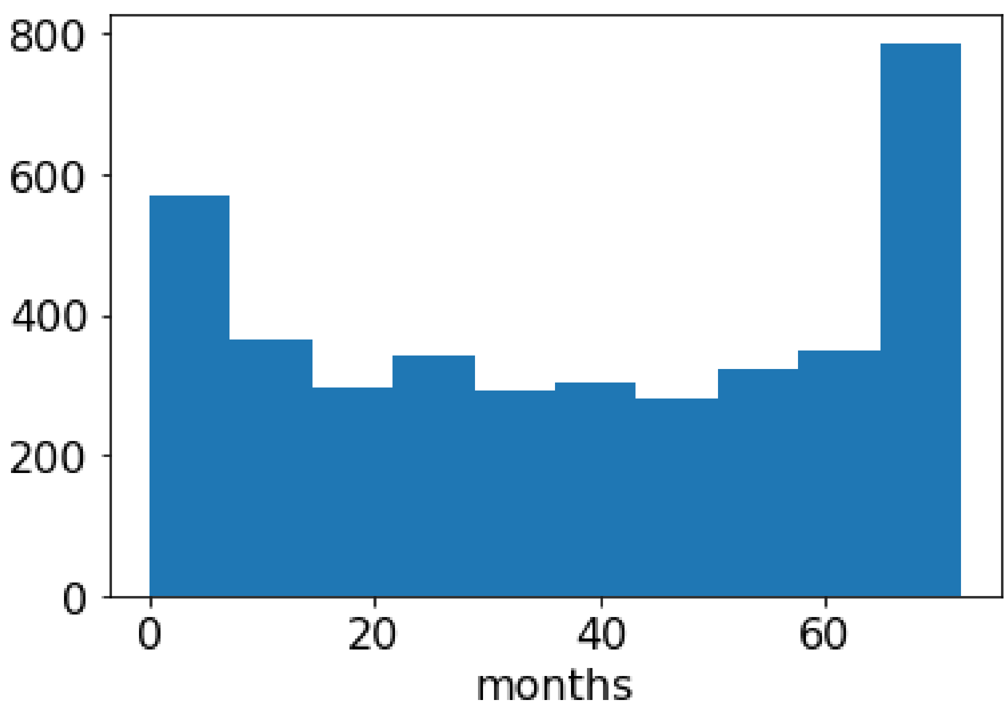
What's the average tenure of the people who did not churn?

[58]:
```python
np.mean(train_df_surv.query("Churn == 0.0")["tenure"])
```

[58]: 37.816717791411044

- Let's look at the histogram of number of people who have **not churned**.
- The key point here is that people ***joined at different times***.

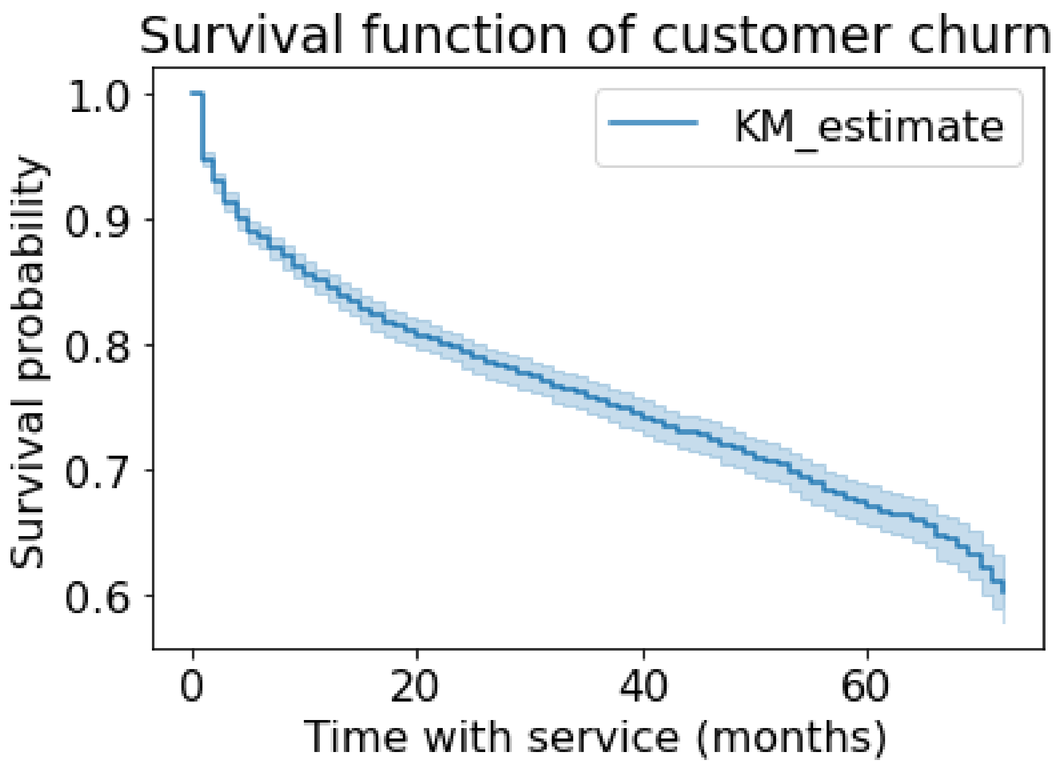
[59]:
```python
train_df[y_train == "No"]["tenure"].hist(grid=False)
plt.xlabel("months");
```



- Since the data was collected at a fixed time and these are the people who hadn't yet churned, those with **larger `tenure`** values here **must have joined earlier**.

Lifelines can also give us some "error bars":

[60]:
```
kmf.plot()
plt.title("Survival function of customer churn")
plt.xlabel("Time with service (months)")
plt.ylabel("Survival probability");
```



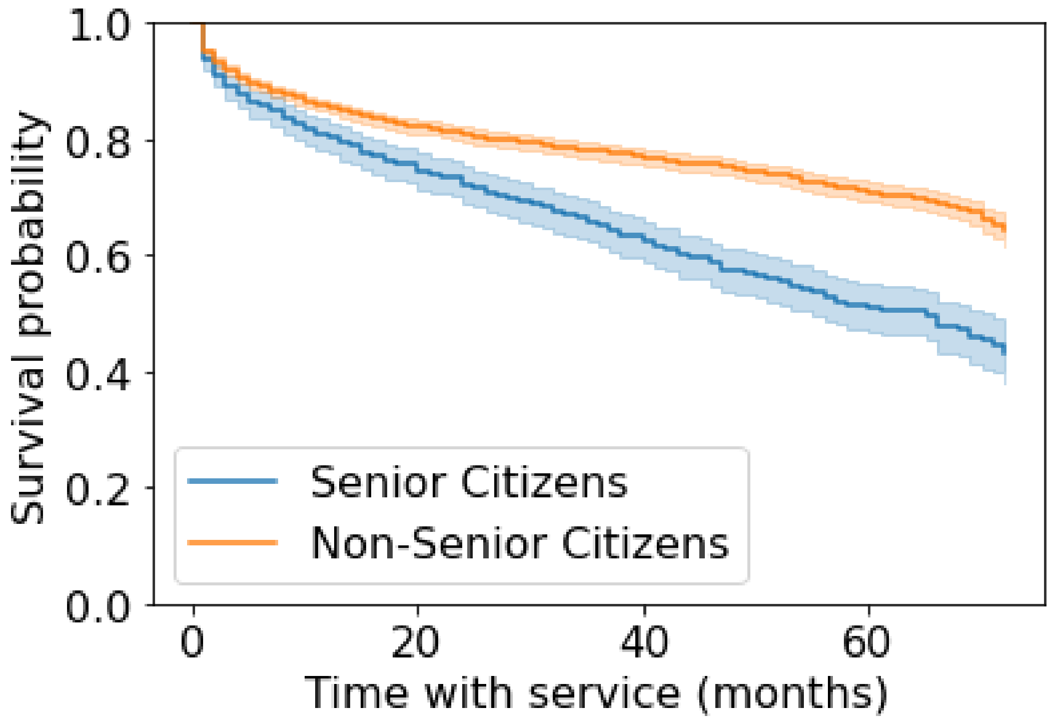
- We already have some **actionable information** here.
- The curve drops down fast at the beginning suggesting that people tend to leave early on.
- If there would have been a big drop in the curve, it means a bunch of people left at that time (e.g., after a 1-month free trial).
- BTW, the original paper by Kaplan and Meier has been cited over 57000 times!

We can also create the K-M curve for **different subgroups**:

[61]:
```
T = train_df_surv["tenure"]
E = train_df_surv["Churn"]
senior = train_df_surv["SeniorCitizen"] == 1
```

[62]:
```
ax = plt.subplot(111)

kmf.fit(T[senior], event_observed=E[senior], label="Senior Citizens")
kmf.plot(ax=ax)

kmf.fit(T[~senior], event_observed=E[~senior], label="Non-Senior Citizens")
```

```
kmf.plot(ax=ax)

plt.ylim(0, 1)
plt.xlabel("Time with service (months)")
plt.ylabel("Survival probability");
```



- It looks like senior citizens churn more quickly than others.
- This is quite useful!

1.7 Cox proportional hazards model

- We haven't been incorporating other features in the model so far.
- The Cox proportional hazards model is a commonly used model that allows us to interpret **how features influence a censored** tenure/duration.
- You can think of it **like linear regression** for **survival analysis**: we will get a coefficient for each feature that tells us how it influences survival.
- It makes some strong assumptions (the proportional hazards assumption) that may not be true, but we won't go into this here.
- The proportional hazard model works multiplicatively, like linear regression with log-transformed targets.

[63]:
```
cph = lifelines.CoxPHFitter()
cph.fit(train_df_surv, duration_col="tenure", event_col="Churn");
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/
↪coxph_fitter.py:1524, in SemiParametricPHFitter.
↪_newton_rhapson_for_efron_model(self, X, T, E, weights, entries,␣
↪initial_point, step_size, precision, show_progress, max_steps)
   1523 try:
-> 1524     inv_h_dot_g_T = spsolve(-h, g, assume_a="pos", check_finite=False)
   1525 except (ValueError, LinAlgError) as e:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/scipy/linalg/_basic
↪py:253, in solve(a, b, sym_pos, lower, overwrite_a, overwrite_b, debug,␣
↪check_finite, assume_a, transposed)
    250 lu, x, info = posv(a1, b1, lower=lower,
    251                    overwrite_a=overwrite_a,
    252                    overwrite_b=overwrite_b)
--> 253 _solve_check(n, info)
    254 rcond, info = pocon(lu, anorm)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/scipy/linalg/_basic
↪py:29, in _solve_check(n, info, lamch, rcond)
     28 elif 0 < info:
---> 29     raise LinAlgError('Matrix is singular.')
     31 if lamch is None:

LinAlgError: Matrix is singular.

During handling of the above exception, another exception occurred:

ConvergenceError                          Traceback (most recent call last)
Input In [63], in <cell line: 2>()
      1 cph = lifelines.CoxPHFitter()
----> 2 cph.fit(train_df_surv, duration_col="tenure", event_col="Churn")

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/utils/
↪__init__.py:56, in CensoringType.right_censoring.<locals>.f(model, *args,␣
↪**kwargs)
     53 @wraps(function)
     54 def f(model, *args, **kwargs):
     55     cls.set_censoring_type(model, cls.RIGHT)
---> 56     return function(model, *args, **kwargs)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/
↪coxph_fitter.py:290, in CoxPHFitter.fit(self, df, duration_col, event_col,␣
↪show_progress, initial_point, strata, step_size, weights_col, cluster_col,␣
↪robust, batch_mode, timeline, formula, entry_col)
    184 """
```

```
    185 Fit the Cox proportional hazard model to a right-censored dataset. Alias␣
↪of `fit_right_censoring`.
    186
   (…)
    287
    288 """
    289 self.strata = utils.coalesce(strata, self.strata)
--> 290 self._model = self._fit_model(
    291     df,
    292     duration_col,
    293     event_col=event_col,
    294     show_progress=show_progress,
    295     initial_point=initial_point,
    296     strata=self.strata,
    297     step_size=step_size,
    298     weights_col=weights_col,
    299     cluster_col=cluster_col,
    300     robust=robust,
    301     batch_mode=batch_mode,
    302     timeline=timeline,
    303     formula=formula,
    304     entry_col=entry_col,
    305 )
    306 return self

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/
↪coxph_fitter.py:616, in CoxPHFitter._fit_model(self, *args, **kwargs)
    614 def _fit_model(self, *args, **kwargs):
    615     if self.baseline_estimation_method == "breslow":
--> 616         return self._fit_model_breslow(*args, **kwargs)
    617     elif self.baseline_estimation_method == "spline":
    618         return self._fit_model_spline(*args, **kwargs)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/
↪coxph_fitter.py:629, in CoxPHFitter._fit_model_breslow(self, *args, **kwargs)
    625 model = SemiParametricPHFitter(
    626     penalizer=self.penalizer, l1_ratio=self.l1_ratio, strata=self.
↪strata, alpha=self.alpha, label=self._label
    627 )
    628 if utils.CensoringType.is_right_censoring(self):
--> 629     model.fit(*args, **kwargs)
    630     return model
    631 else:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/utils/
↪__init__.py:56, in CensoringType.right_censoring.<locals>.f(model, *args,␣
↪**kwargs)
     53 @wraps(function)
```

```
     54 def f(model, *args, **kwargs):
     55     cls.set_censoring_type(model, cls.RIGHT)
---> 56     return function(model, *args, **kwargs)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/coxph_fitter.py:1257, in SemiParametricPHFitter.fit(self, df, duration_col, event_col, show_progress, initial_point, strata, step_size, weights_col, cluster_col, robust, batch_mode, timeline, formula, entry_col)
   1252 # this is surprisingly faster to do…
   1253 X_norm = pd.DataFrame(
   1254     utils.normalize(X.values, self._norm_mean.values, self._norm_std.values), index=X.index, columns=X.columns
   1255 )
-> 1257 params_, ll_, variance_matrix_, baseline_hazard_, baseline_cumulative_hazard_, model = self._fit_model(
   1258     X_norm,
   1259     T,
   1260     E,
   1261     weights=weights,
   1262     entries=entries,
   1263     initial_point=initial_point,
   1264     show_progress=show_progress,
   1265     step_size=step_size,
   1266 )
   1268 self.log_likelihood_ = ll_
   1269 self.model = model

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/coxph_fitter.py:1385, in SemiParametricPHFitter._fit_model(self, X, T, E, weights, entries, initial_point, step_size, show_progress)
   1374 def _fit_model(
   1375     self,
   1376     X: DataFrame,
   (…)
   1383     show_progress: bool = True,
   1384 ):
-> 1385     beta_, ll_, hessian_ = self._newton_rhapson_for_efron_model(
   1386         X, T, E, weights, entries, initial_point=initial_point, step_size=step_size, show_p
   1387     )
   1389     # compute the baseline hazard here.
   1390     predicted_partial_hazards_ = (
   1391         pd.DataFrame(np.exp(dot(X, beta_)), columns=["P"]).assign(T=T.values, E=E.values, W=weights.values).set_index(X.index)
   1392     )
```

```
File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/
↪coxph_fitter.py:1533, in SemiParametricPHFitter.
↪_newton_rhapson_for_efron_model(self, X, T, E, weights, entries,␣
↪initial_point, step_size, precision, show_progress, max_steps)
  1528     raise exceptions.ConvergenceError(
  1529         """Hessian or gradient contains nan or inf value(s). Convergenc␣
↪halted. {0}""".format(CONVERGENCE_DOCS),
  1530         e,
  1531     )
  1532 elif isinstance(e, LinAlgError):
-> 1533     raise exceptions.ConvergenceError(
  1534         """Convergence halted due to matrix inversion problems.␣
↪Suspicion is high collinearity. {0}""".format(
  1535             CONVERGENCE_DOCS
  1536         ),
  1537         e,
  1538     )
  1539 else:
  1540     # something else?
  1541     raise e

ConvergenceError: Convergence halted due to matrix inversion problems. Suspicio␣
↪is high collinearity. Please see the following tips in the lifelines␣
↪documentation: https://lifelines.readthedocs.io/en/latest/Examples.
↪html#problems-with-convergence-in-the-cox-proportional-hazard-modelMatrix is␣
↪singular.
```

- Ok, going that that URL, it seems the easiest solution is to add a penalizer.
  - FYI this is related to switching from `LinearRegression` to `Ridge`.
  - Adding `drop='first'` on our OHE might have helped with this.
  - (For 340 folks: we're adding regularization; `lifelines` adds both L1 and L2 regularization, aka elastic net)

[64]:
```python
cph = lifelines.CoxPHFitter(penalizer=0.1)
cph.fit(train_df_surv, duration_col="tenure", event_col="Churn");
```

We can look at the coefficients learned by the model and start interpreting them!

[65]:
```python
cph_params = pd.DataFrame(cph.params_).sort_values(by="coef", ascending=False)
cph_params
```

[65]:

	coef
covariate	
Contract_Month-to-month	0.812874
OnlineSecurity_No	0.311151
OnlineBackup_No	0.298561
PaymentMethod_Electronic check	0.280801
Partner_No	0.244814
…	…

```
OnlineBackup_Yes                          -0.282600
PaymentMethod_Credit card (automatic) -0.302801
OnlineSecurity_Yes                        -0.330346
Contract_One year                         -0.351822
Contract_Two year                         -0.776425

[43 rows x 1 columns]
```

- Looks like month-to-month leads to more churn, two-year contract leads to less churn; this makes sense!!!

[66]:
```
# cph.baseline_hazard_ # baseline hazard
```

[67]:
```
# cph.summary
```

Could we have gotten this type of information out of sklearn?

[68]:
```
y_train.head()
```

[68]:
```
6464     No
5707     No
3442     No
3932    Yes
6124     No
Name: Churn, dtype: object
```

[69]:
```
X_train.drop(columns=["tenure"]).head()
```

[69]:
```
      customerID  gender  SeniorCitizen Partner Dependents PhoneService  \
6464  4726-DLWQN    Male              1      No         No          Yes
5707  4537-DKTAL  Female              0      No         No          Yes
3442  0468-YRPXN    Male              0      No         No          Yes
3932  1304-NECVQ  Female              1      No         No          Yes
6124  7153-CHRBV  Female              0     Yes        Yes          Yes

     MultipleLines InternetService OnlineSecurity OnlineBackup  \
6464           Yes             DSL            Yes          Yes
5707            No             DSL             No           No
3442            No     Fiber optic             No           No
3932           Yes     Fiber optic             No           No
6124            No             DSL            Yes           No

     DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
6464               No          No         Yes              No  Month-to-month
5707               No          No          No              No  Month-to-month
3442              Yes         Yes         Yes             Yes  Month-to-month
3932              Yes          No          No              No  Month-to-month
6124              Yes         Yes          No              No        One year
```

```
     PaperlessBilling             PaymentMethod  MonthlyCharges  TotalCharges
6464              Yes  Bank transfer (automatic)          70.35       3454.60
5707               No           Electronic check          45.55         84.40
3442              Yes    Credit card (automatic)          98.80       2807.10
3932              Yes           Electronic check          78.55        149.55
6124              Yes               Mailed check          59.30       3274.35
```

I'm redefining feature types and our preprocessor for our sanity.

[70]:
```
numeric_features = ["MonthlyCharges", "TotalCharges"]
drop_features = ["customerID", "tenure"]
passthrough_features = ["SeniorCitizen"]
target_column = ["Churn"]
# the rest are categorical
categorical_features = list(
    set(train_df.columns)
    - set(numeric_features)
    - set(passthrough_features)
    - set(drop_features)
    - set(target_column)
)
```

[71]:
```
preprocessor = make_column_transformer(
    (
        make_pipeline(SimpleImputer(strategy="median"), StandardScaler()),
        numeric_features,
    ),
    (OneHotEncoder(handle_unknown="ignore"), categorical_features),
    ("passthrough", passthrough_features),
    ("drop", drop_features),
)
```

[72]:
```
preprocessor.fit(X_train);
```

[73]:
```
new_columns = (
    numeric_features
    + preprocessor.named_transformers_["onehotencoder"]
    .get_feature_names_out(categorical_features)
    .tolist()
    + passthrough_features
)
```

[74]:
```
lr = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
lr.fit(X_train, y_train)
lr_coefs = pd.DataFrame(
    data=np.squeeze(lr[1].coef_), index=new_columns, columns=["Coefficient"]
)
```

[75]:
```
lr_coefs.sort_values(by="Coefficient", ascending=False)
```

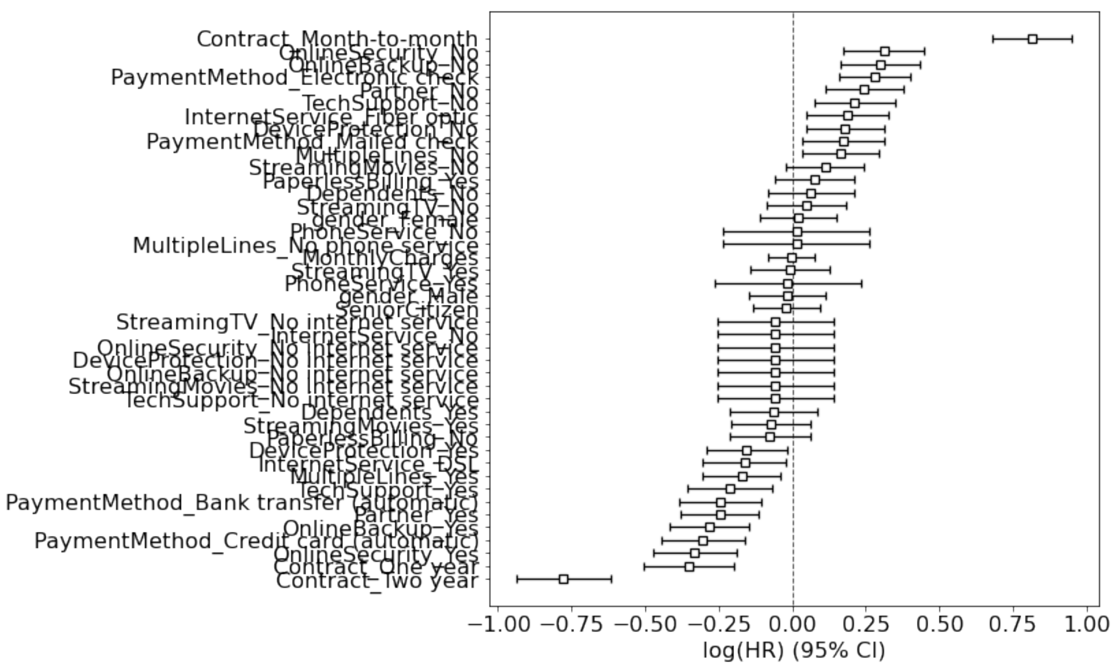
[75]:

	Coefficient
Contract_Month-to-month	0.787653
InternetService_Fiber optic	0.600509
OnlineSecurity_No	0.291008
StreamingTV_Yes	0.258659
PaymentMethod_Electronic check	0.251646
…	…
MultipleLines_No	-0.169654
PaymentMethod_Credit card (automatic)	-0.204406
InternetService_DSL	-0.461593
TotalCharges	-0.743315
Contract_Two year	-0.765519

[44 rows x 1 columns]

- There is some agreement, which is good.
- But our survival model is much more useful.
  - Not to mention more correct.
- One thing we get with `lifelines` is confidence intervals on the coefficients:

[76]:
```
plt.figure(figsize=(8, 8))
cph.plot();
```

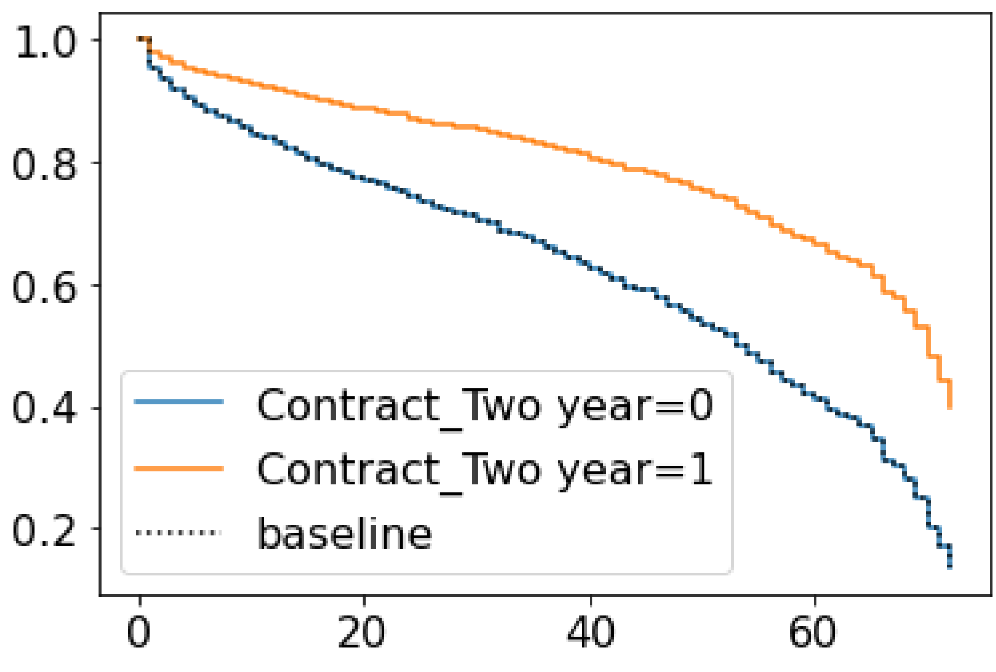


- (We could probably get the same for logistic regression if using `statsmodels` instead of

sklearn.)
- However, in general, I would be careful with all of this.
- Ideally we would have more statistical training when using `lifelines` - there is a lot that can go wrong.
  - It comes with various diagnostics as well.
- But I think it's very useful to know about survival analysis and the availability of software to deal with it.
- Oh, and there are lots of other nice plots.

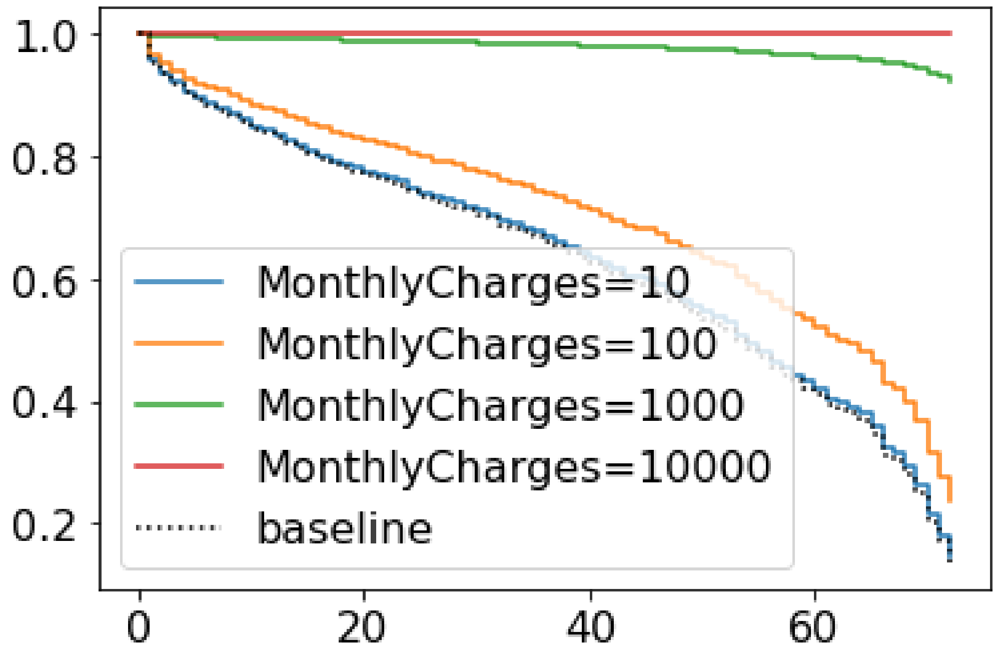
- Let's look at the survival plots for the people with
  - two-year contract (Contract_Two year = 1) and
  - people without two-year contract (Contract_Two year = 0)
- As expected, the former survive longer.

[77]:
```
cph.plot_partial_effects_on_outcome("Contract_Two year", [0, 1]);
```



Now let's look at the survival plots for the people with different MonthlyCharges.

[78]:
```
cph.plot_partial_effects_on_outcome("MonthlyCharges", [10, 100, 1000, 10_000]);
```



- That's the thing with linear models, they can't stop the growth.
- We have a negative coefficient associated with `MonthlyCharges`

[79]:
```
cph_params.loc["MonthlyCharges"]
```

[79]:
```
coef   -0.003185
Name: MonthlyCharges, dtype: float64
```

If your monthly charges are huge, it takes this to the extreme and thinks you'll basically never churn.

1.8 Prediction

- We can use survival analysis to make predictions as well.
- Here is the expected number of months to churn for the first 5 customers in the test set:

[80]:
```
test_df_surv.drop(columns=["tenure", "Churn"]).head()
```

[80]:
```
      SeniorCitizen  MonthlyCharges  OnlineBackup_No  \
941             0.0       -1.154900              0.0
1404            0.0       -1.383246              0.0
5515            0.0       -1.514920              0.0
3684            0.0        0.351852              1.0
7017            0.0       -1.471584              0.0

      OnlineBackup_No internet service  OnlineBackup_Yes  StreamingMovies_No  \
941                                0.0               1.0                 1.0
```

```
1404                             1.0              0.0                  0.0
5515                             1.0              0.0                  0.0
3684                             0.0              0.0                  1.0
7017                             1.0              0.0                  0.0

      StreamingMovies_No internet service  StreamingMovies_Yes  Dependents_No  \
941                                   0.0                  0.0            0.0
1404                                  1.0                  0.0            1.0
5515                                  1.0                  0.0            0.0
3684                                  0.0                  0.0            1.0
7017                                  1.0                  0.0            1.0

      Dependents_Yes  …  gender_Male  \
941              1.0  …          0.0
1404             0.0  …          0.0
5515             1.0  …          0.0
3684             0.0  …          1.0
7017             0.0  …          0.0

      PaymentMethod_Bank transfer (automatic)  \
941                                       0.0
1404                                      1.0
5515                                      0.0
3684                                      0.0
7017                                      1.0

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
941                                     0.0                             1.0
1404                                    0.0                             0.0
5515                                    0.0                             0.0
3684                                    1.0                             0.0
7017                                    0.0                             0.0

      PaymentMethod_Mailed check  PhoneService_No  PhoneService_Yes  \
941                          0.0              1.0               0.0
1404                         0.0              0.0               1.0
5515                         1.0              0.0               1.0
3684                         0.0              0.0               1.0
7017                         0.0              0.0               1.0

      StreamingTV_No  StreamingTV_No internet service  StreamingTV_Yes
941              1.0                              0.0              0.0
1404             0.0                              1.0              0.0
5515             0.0                              1.0              0.0
3684             1.0                              0.0              0.0
7017             0.0                              1.0              0.0
```

```
[5 rows x 43 columns]
```

[81]:
```python
test_df_surv.head()
```

[81]:
```
      Churn  tenure  SeniorCitizen  MonthlyCharges  OnlineBackup_No  \
941     0.0    13.0            0.0       -1.154900              0.0
1404    0.0    35.0            0.0       -1.383246              0.0
5515    0.0    18.0            0.0       -1.514920              0.0
3684    0.0    43.0            0.0        0.351852              1.0
7017    0.0    51.0            0.0       -1.471584              0.0

      OnlineBackup_No internet service  OnlineBackup_Yes  StreamingMovies_No  \
941                                0.0               1.0                 1.0
1404                               1.0               0.0                 0.0
5515                               1.0               0.0                 0.0
3684                               0.0               0.0                 1.0
7017                               1.0               0.0                 0.0

      StreamingMovies_No internet service  StreamingMovies_Yes  …  \
941                                   0.0                  0.0  …
1404                                  1.0                  0.0  …
5515                                  1.0                  0.0  …
3684                                  0.0                  0.0  …
7017                                  1.0                  0.0  …

      gender_Male  PaymentMethod_Bank transfer (automatic)  \
941           0.0                                      0.0
1404          0.0                                      1.0
5515          0.0                                      0.0
3684          1.0                                      0.0
7017          0.0                                      1.0

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
941                                     0.0                             1.0
1404                                    0.0                             0.0
5515                                    0.0                             0.0
3684                                    1.0                             0.0
7017                                    0.0                             0.0

      PaymentMethod_Mailed check  PhoneService_No  PhoneService_Yes  \
941                          0.0              1.0               0.0
1404                         0.0              0.0               1.0
5515                         1.0              0.0               1.0
3684                         0.0              0.0               1.0
7017                         0.0              0.0               1.0

      StreamingTV_No  StreamingTV_No internet service  StreamingTV_Yes
```

```
941     1.0                                0.0              0.0
1404    0.0                                1.0              0.0
5515    0.0                                1.0              0.0
3684    1.0                                0.0              0.0
7017    0.0                                1.0              0.0

[5 rows x 45 columns]
```

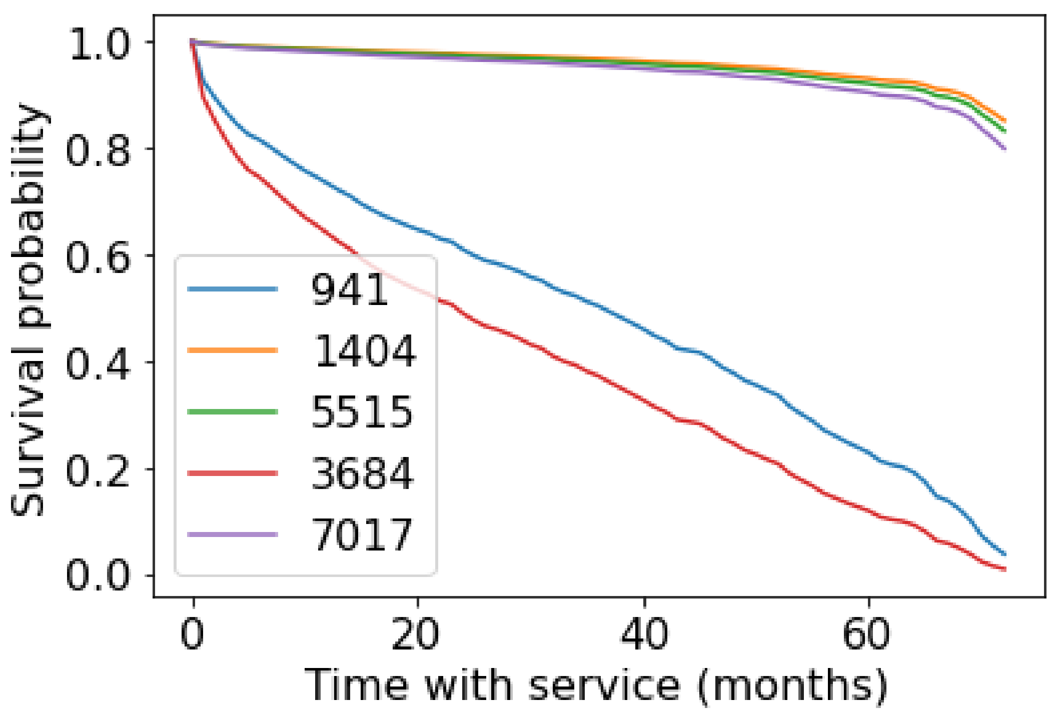
How long each non-churned customer is likely to stay according to the model assuming that they just joined right now?

[82]:
```
cph.predict_expectation(test_df_surv).head()  # assumes they just joined right␣
↪now
```

[82]:
```
941     35.206731
1404    69.023078
5515    68.608555
3684    27.565069
7017    67.890922
dtype: float64
```

Survival curves for first 5 customers in the test set:

[83]:
```
cph.predict_survival_function(test_df_surv[:5]).plot()
plt.xlabel("Time with service (months)")
plt.ylabel("Survival probability");
```



From `predict_survival_function` documentation:

> Predict the survival function for individuals, given their covariates. This assumes that the individual just entered the study (that is, we do not condition on how long they have already lived for.)

So these curves are "starting now".

- There's no probability prerequisite for this course, so this is optional material.
- But you can do some interesting stuff here with conditional probabilities.
- "Given that a customer has been here 5 months, what's the outlook?"
  - It will be different than for a new customer.
  - Thus, we might still want to predict for the non-churned customers in the training set!
  - Not something we really thought about with our traditional supervised learning.

Let's get the customers who have not churned yet.

[84]:
```
train_df_surv_not_churned = train_df_surv[train_df_surv["Churn"] == 0]
```

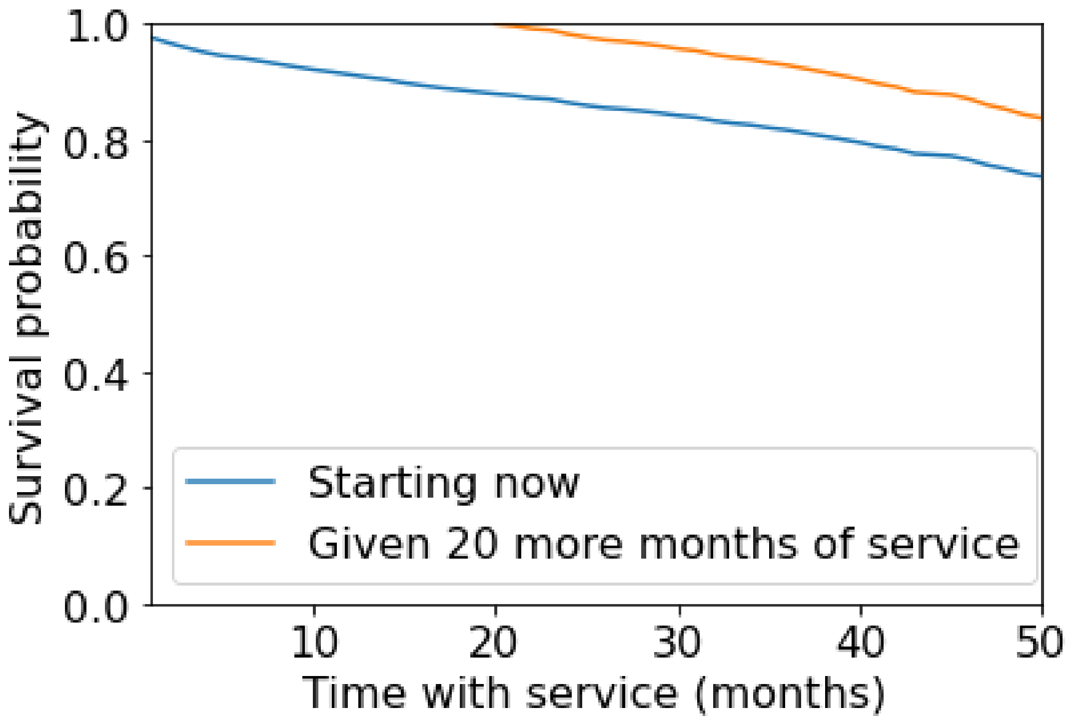
We can *condition* on the person having been around for 20 months.

[85]:
```
cph.predict_survival_function(train_df_surv_not_churned[:1],␣
 ↪conditional_after=20)
```

[85]:
```
          6464
0.0   1.000000
1.0   0.996788
2.0   0.991966
3.0   0.989443
4.0   0.982570
…          …
68.0  0.429635
69.0  0.429635
70.0  0.429635
71.0  0.429635
72.0  0.429635

[73 rows x 1 columns]
```

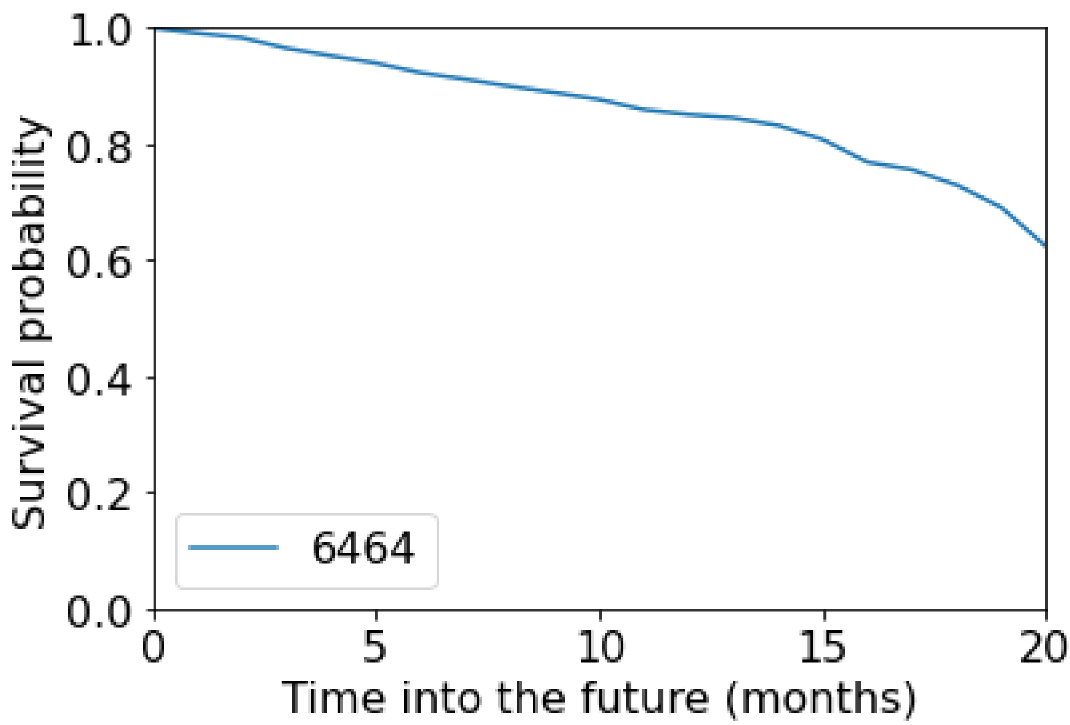
[86]:
```
plt.figure()
cph.predict_survival_function(train_df_surv_not_churned[:1]).plot(ax=plt.gca())
preds = cph.predict_survival_function(
    train_df_surv_not_churned[:1], conditional_after=20
)
plt.plot(preds.index[20:], preds.values[:-20])
plt.xlabel("Time with service (months)")
plt.ylabel("Survival probability")
plt.legend(["Starting now", "Given 20 more months of service"])
plt.ylim([0, 1])
```

```
plt.xlim([1, 50]);
```



- Look at how the survival function (and expected lifetime) is much longer *given* that the customer has already lasted 20 months.
- How long each non-churned customer is likely to stay according to the model assuming that they have been here for the tenure time?
- So, we can set this to their actual tenure so far to get a prediciton of what will happen going forward:

[87]:
```
cph.predict_survival_function(
    train_df_surv_not_churned[:1],
    conditional_after=train_df_surv_not_churned[:1]["tenure"],
).plot()
plt.xlabel("Time into the future (months)")
plt.ylabel("Survival probability")
plt.ylim([0, 1])
plt.xlim([0, 20]);
```



- Another useful application: you could ask what is the customer lifetime value.
  - Basically, how much money do you expect to make off this customer between now and when they churn?
- With regular supervised learning, tenure was a feature and we could only predict whether or not they had churned by then.

[ ]:

```

```

1.9 Evaluation

By default score returns "partial log likelihood":

[88]:
```
cph.score(train_df_surv)
```

[88]: -1.8641864810547808

[89]:
```
cph.score(test_df_surv)
```

[89]: -1.7277855153568578

We can look at the "concordance index" which is more interpretable:

[90]:
```
cph.concordance_index_
```

[90]: 0.8625886620148505

[91]:
```
cph.score(train_df_surv, scoring_method="concordance_index")
```

[91]: 0.8625886620148505

[92]:
```
cph.score(test_df_surv, scoring_method="concordance_index")
```

[92]: 0.8546143543902771

From the documentation here:

> Another censoring-sensitive measure is the concordance-index, also known as the c-index. This measure evaluates the accuracy of the ranking of predicted time. It is in fact a generalization of AUC, another common loss function, and is interpreted similarly:
>
> - 0.5 is the expected result from random predictions,
> - 1.0 is perfect concordance and,
> - 0.0 is perfect anti-concordance (multiply predictions with -1 to get 1.0)
>
> Here is an excellent introduction & description of the c-index for new users.

[93]:
```
# cph.log_likelihood_ratio_test()
```

[94]:
```
# cph.check_assumptions(df_train_surv)
```

1.10 Other approaches / what did we not cover?

There are many other approaches to modelling in survival analysis:

- Time-varying proportional hazards.
  - What if some of the features change over time, e.g. plan type, number of lines, etc.
- Approaches based on deep learning, e.g. the pysurvival package.
- Random survival forests.
- And more…

1.10.1 Types of censoring

There are also various types and sub-types of censoring we didn't cover:

- What we did today is called "right censoring"
- Sub-types within right censoring
  - Did everyone join at the same time?
  - Other there other reasons the data might be censored at random times, e.g. the person died?
- Left censoring
- Interval censoring

1.11 Summary

- Censoring and incorrect approaches to handling it
  - Throw away people who haven't churned
  - Assume everyone churns today
- Predicting tenure vs. churned

- Survival analysis encompasses both of these, and deals with censoring
- And it can make rich and interesting predictions!
- KM model -> doesn't look at features
- CPH model -> like linear regression, does look at the features

1.12 True/False questions

1. If all customers joined a service at the same time (hypothetically), then censoring would not be an issue. **FALSE**
2. The Cox proportional hazards model (`cph` above) assumes the effect of a feature is the same for all customers and over all time. **TRUE**
3. Survival analysis can be useful even without a "deployment" stage. **TRUE**

1.13 References

Some people working with this same dataset:

- https://medium.com/@zachary.james.angell/applying-survival-analysis-to-customer-churn-40b5a809b05a
- https://towardsdatascience.com/churn-prediction-and-prevention-in-python-2d454e5fd9a5 (Cox)
- https://towardsdatascience.com/survival-analysis-in-python-a-model-for-customer-churn-e737c5242822
- https://towardsdatascience.com/survival-analysis-intuition-implementation-in-python-504fde4fcf8e

lifelines documentation: - https://lifelines.readthedocs.io/en/latest/Survival%20analysis%20with%20lifelines.html - https://lifelines.readthedocs.io/en/latest/Survival%20Analysis%20intro.html#introduction-to-survival-analysis