

```
# Load libraries
import shutil # 提供文件复制、移动等操作功能
import os # 用于路径处理和目录操作
from pyspark.sql.functions import col, when # Spark SQL中常用的列操作函数
from pyspark.sql.types import StructType, StructField, DoubleType, StringType, IntegerType # 用于定义DataFrame的结构
import getpass # 用于获取当前操作系统用户名

# Set config for database name, file paths, and table names
user = getpass.getuser() # 获取当前系统用户名，替代Databricks中的用户标识
username_sql = user.replace(".", "_") # 将用户名中的点替换为下划线，避免数据库命名问题

# 修改路径为本地路径
database_name = f'dm_ibm_telco_churn_{username_sql}' # 设置数据库名称，带上用户名以便区分

# 使用本地路径而不是HDFS/DBFS路径
# 假设数据放在当前工作目录下的data文件夹中
data_path = os.path.join('data', 'ibm-telco-churn') # 定义原始数据的本地存放路径
os.makedirs(data_path, exist_ok=True) # 如果路径不存在则创建，避免后续出错

# 原始数据文件路径
driver_to_local_path = os.path.join(data_path, 'data.txt') # 指定原始数据文件的完整路径

# 处理后的数据路径
bronze_tbl_path = os.path.join(data_path, 'bronze') # bronze阶段数据输出路径
silver_tbl_path = os.path.join(data_path, 'silver') # silver阶段数据输出路径

# 创建必要的目录
os.makedirs(bronze_tbl_path, exist_ok=True) # 创建bronze目录（如不存在）
os.makedirs(silver_tbl_path, exist_ok=True) # 创建silver目录（如不存在）

# 表名保持不变
bronze_tbl_name = 'bronze_customers' # bronze阶段表名
silver_tbl_name = 'silver_monthly_customers' # silver阶段表名
'

getpass.getuser(): 获取当前系统登录用户名，避免硬编码，适配多用户环境。
os.path.join(): 构建跨平台安全路径，避免手动拼接出错。
os.makedirs(): 自动创建目录，不报错即使目录已存在。
f'dm_ibm_telco_churn_{username_sql}': 数据库名动态拼接用户名，确保唯一性。
```

```
# Define schema
schema = StructType([
    StructField('customerID', StringType()),
    StructField('gender', StringType()),
    StructField('seniorCitizen', DoubleType()),
    StructField('partner', StringType()),
    StructField('dependents', StringType()),
    StructField('tenure', DoubleType()),
    StructField('phoneService', StringType()),
    StructField('multipleLines', StringType()),
    StructField('internetService', StringType()),
    StructField('onlineSecurity', StringType()),
    StructField('onlineBackup', StringType()),
    StructField('deviceProtection', StringType()),
    StructField('techSupport', StringType()),
    StructField('streamingTV', StringType()),
    StructField('streamingMovies', StringType()),
    StructField('contract', StringType()),
    StructField('paperlessBilling', StringType()),
    StructField('paymentMethod', StringType()),
    StructField('monthlyCharges', DoubleType()),
    StructField('totalCharges', DoubleType()),
    StructField('Churn', StringType())
])
```

```
# 本地加载数据 - 两种方式可选

# 方式1：使用Spark（需要已配置好Spark环境）
from pyspark.sql import SparkSession

# 初始化Spark会话
spark = SparkSession.builder \
    .appName("TelcoChurnAnalysis") \
    .getOrCreate()

# 复制文件到工作目录（如果文件不在指定位置）
import shutil
shutil.copy2('Telco-Customer-Churn.csv', driver_to_local_path)

# 读取CSV文件
bronze_df = spark.read.format('csv') \
    .schema(schema) \
    .option('header', 'true') \
    .load(driver_to_local_path)

# 显示数据
bronze_df.show(5)
```

核心函数：

SparkSession.builder.getOrCreate(): 初始化或获取一个 Spark 会话，是 PySpark 操作的入口。

spark.read.format('csv')...load(): 加载 CSV 文件到 DataFrame，支持自定义 schema，确保数据格式正确。

```

# Construct silver table
silver_df = bronze_df.withColumn(
    'churn', # 使用新列名而不是覆盖原列
    when(col('Churn') == 'Yes', 1)
        .when(col('Churn') == 'No', 0)
        .otherwise(None) # 使用None而不是'Unknown'，因为这是数值列
).filter(col('contract') == 'Month-to-month') \
.filter(col('internetService') != 'No')

```

核心函数：

withColumn(): 用于添加或更新 DataFrame 的列，这里根据 Churn 列的值创建了新的 churn 列，将 'Yes' 和 'No' 转换为 1 和 0。

when(...).otherwise(None): 条件判断语句，类似 SQL 的 CASE WHEN，用于根据条件为新列赋值。

```

# 创建数据库和表的本地实现
try:
    # 删除旧数据库
    spark.sql(f'DROP DATABASE IF EXISTS {database_name} CASCADE')

    # 创建新数据库
    spark.sql(f'CREATE DATABASE {database_name}')
    spark.sql(f'USE {database_name}')

    # 将DataFrame保存为表
    bronze_df.write.mode('overwrite').saveAsTable(bronze_tbl_name)
    silver_df.write.mode('overwrite').saveAsTable(silver_tbl_name)

    print(f"成功创建数据库 {database_name} 和表 {bronze_tbl_name}, {silver_tbl_name}")
except Exception as e:
    print(f"使用Spark SQL创建数据库失败: {str(e)}")
    print("将使用本地Parquet文件替代")# 显示结果
print("\nSilver表预览:")
silver_df.show(5)

spark.sql(f'DROP DATABASE IF EXISTS {database_name} CASCADE'): 删除指定的数据库及其所有内容，CASCADE 表示连同数据库中的所有表都一起删除。
write.mode('overwrite').saveAsTable(): 将 DataFrame 保存为 Spark SQL 表，overwrite 模式会覆盖表中现有的数据。

```

```
import os
os.makedirs(bronze_tbl_path, exist_ok=True)
os.makedirs(silver_tbl_path, exist_ok=True)

# 使用Parquet格式写入数据
print("使用Parquet格式保存数据...")
bronze_df.write.parquet(
    bronze_tbl_path,
    mode="overwrite",
    compression="snappy" # 可选压缩方式
)
silver_df.write.parquet(
    silver_tbl_path,
    mode="overwrite",
    compression="snappy" # 可选压缩方式
)

print(f"数据已成功保存到:")
print(f"- Bronze层: {bronze_tbl_path}")
print(f"- Silver层: {silver_tbl_path}")

# 验证数据写入
print("\n验证写入结果:")
print("Bronze表预览:")
spark.read.parquet(bronze_tbl_path).show(5)
print("\nSilver表预览:")
spark.read.parquet(silver_tbl_path).show(5)
```

os.makedirs(..., exist_ok=True): 创建指定路径的目录，若目录已存在则不会报错，确保数据文件保存路径存在。

write.parquet(): 将 DataFrame 写入 Parquet 格式的文件，**overwrite** 模式会覆盖已存在的文件，**compression="snappy"** 指定使用 Snappy 压缩方式。
spark.read.parquet()：读取 Parquet 文件为 DataFrame，用于验证数据是否正确写入。

```

# 创建临时视图替代物理表
bronze_df.createOrReplaceTempView(bronze_tbl_name)
silver_df.createOrReplaceTempView(silver_tbl_name)

print(f"已创建临时视图: {bronze_tbl_name} 和 {silver_tbl_name}")
print("可以通过spark.sql()查询这些视图")

# 示例查询
print("\nSilver表示例查询结果:")
spark.sql(f"SELECT * FROM {silver_tbl_name} LIMIT 5").show()

# 使用Spark SQL查询
print("Bronze Customers表数据:")
spark.sql("SELECT * FROM bronze_customers LIMIT 5").show()

print("\nSilver Monthly Customers表数据:")
spark.sql("SELECT * FROM silver_monthly_customers LIMIT 5").show()

# Import Libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from lifelines import KaplanMeierFitter
from lifelines.utils import median_survival_times
from lifelines.statistics import pairwise_logrank_test
# Load the telco silver table
telco_pd = spark.table('silver_monthly_customers').toPandas()

print(telco_pd.columns.tolist())
['customerID', 'gender', 'seniorCitizen', 'partner', 'dependents', 'tenure', 'phoneService', 'multipleLines', 'internetService', 'onlineSecurity', 'onlineBackup', 'deviceProtection', 'techSupport', 'streamingTV', 'streamingMovies', 'contract', 'paperlessBilling', 'paymentMethod', 'monthlyCharges', 'totalCharges', 'churn']

kmf = KaplanMeierFitter()

T=telco_pd['tenure']
C=telco_pd['churn'].astype(float)

kmf.fit(T,C)

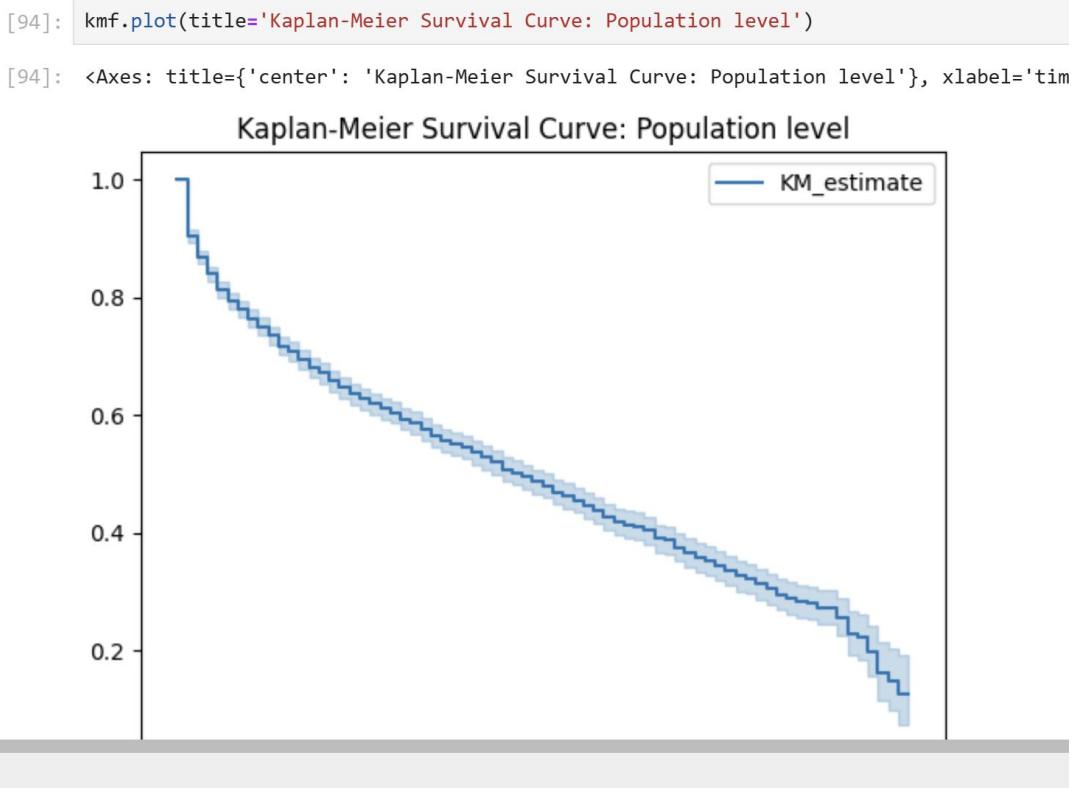
<lifelines.KaplanMeierFitter: "KM_estimate", fitted with 3351 total observations, 1795 right-censored observations>

```

spark.table('silver_monthly_customers').toPandas():
将 Spark SQL 表转换为 Pandas DataFrame，方便后续使用 lifelines 库进行生存分析。

KaplanMeierFitter(): 创建一个 Kaplan-Meier 生存分析模型实例，用于计算和拟合生存曲线。

`kmf.fit(T, C)`: 拟合 Kaplan-Meier 曲线，T 为时间（这里是 'tenure'），C 为是否发生事件 ('churn' 转换为 0 或 1)。



横轴 (X 轴) :

时间 (tenure): 表示从客户开始使用服务到当前时间的时长，通常以月数为单位。这个时间表示客户的使用期 (tenure)，即客户与公司签订合同的时间长度。

纵轴 (Y 轴) :

生存概率：表示客户在特定时间点之前“存活”的概率。生存概率指的是客户在该时间点之后仍然处于“未流失”（未取消服务）状态的概率，通常以百分比的形式（从 0 到 1，或者从 0% 到 100%）表示。

图的含义：生存曲线描绘的是一个群体在某个时间点上的流失概率，通常我们会看到曲线随着时间逐渐下降，表示随着时间推移，客户流失的概率逐渐增大。

曲线下降的速度较快时，说明大部分客户在该时间段内流失较多。相反，曲线较平缓时，表明客户的流失较少。

```
# Helper function for plotting Kaplan-Meier curves at the covariate level
def plot_km(col):
    ax = plt.subplot(111)
    for r in telco_pd[col].unique():
        ix = telco_pd[col] == r
        kmf.fit(T[ix], C[ix], label=r)
        kmf.plot(ax=ax)

# Helper function for printing out Log-rank test results
def print_logrank(col):
    log_rank = pairwise_logrank_test(telco_pd['tenure'], telco_pd[col], telco_pd['churn'])
    return log_rank.summary
```

plot_km(col)绘制不同协变量水平下的 Kaplan-Meier 生存曲线。**col:** 需要分组的协变量列（例如，性别、是否有合作伙伴等）。通过遍历协变量的每个唯一值（例如，不同性别），对每个组的数据进行生存曲线拟合并绘制在同一图上。

kmf.fit(T[ix], C[ix], label=r): 根据协变量的每个值过滤数据，拟合 Kaplan-Meier 曲线。

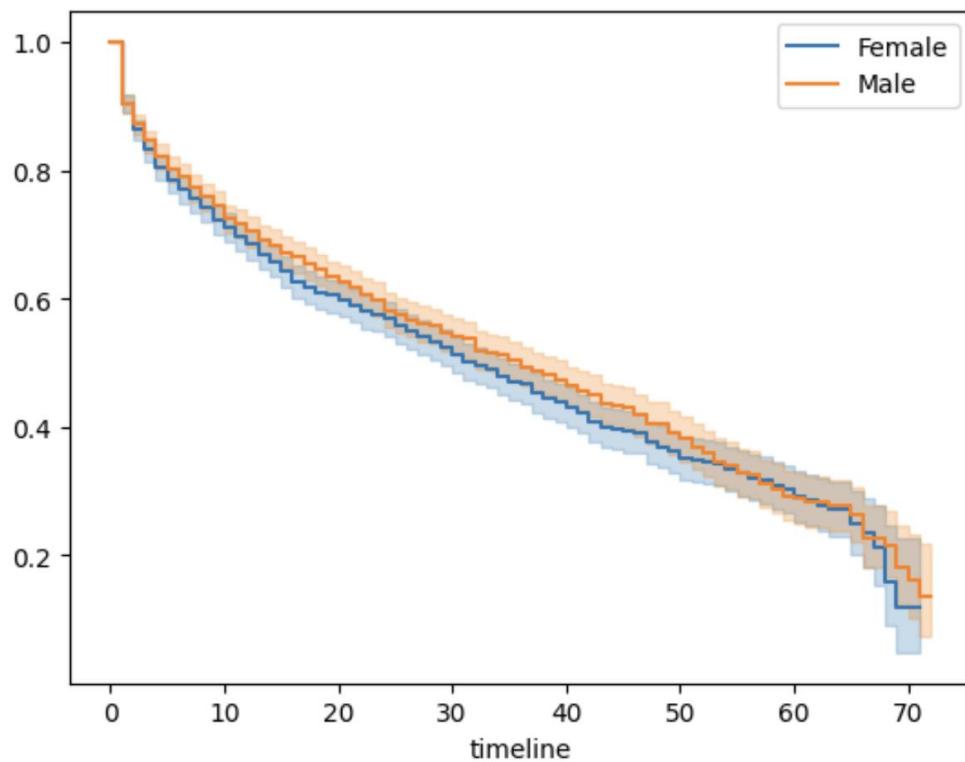
kmf.plot(ax=ax): 将每组数据的生存曲线绘制在同一个坐标轴上，便于对比不同组之间的生存情况。

print_logrank(col): 进行 Log-rank 检验，以比较不

同协变量水平下的生存曲线差异。`col`: 需要进行比较的协变量列。使用 `pairwise_logrank_test()` 进行检验，返回 **Log-rank** 检验的结果。

`log_rank.summary`: 显示检验结果的总结，包括 `p` 值、是否显著等统计信息，帮助判断不同协变量水平下的生存曲线是否存在显著差异。

```
: plot_km('gender')
```



```
[181]: print_logrank('gender')
print_logrank('techSupport')
print_logrank('streamingTV')
print_logrank('paperlessBilling')
print_logrank('paymentMethod')
```

		test_statistic	p	-log2(p)
Bank transfer (automatic)	Credit card (automatic)	0.061543	8.040732e-01	0.314601
	Electronic check	91.191889	1.303937e-21	69.377616
	Mailed check	43.536998	4.160192e-11	34.484559
Credit card (automatic)	Electronic check	79.991082	3.761035e-19	61.205504
	Mailed check	39.684613	2.984678e-10	31.641706
	Electronic check	0.898320	3.432326e-01	1.542741

```
[180]: print_logrank('deviceProtection')
```

		test_statistic	p	-log2(p)
No	Yes	71.496825	2.777047e-17	54.999226

```
[179]: print_logrank('onlineBackup')
```

		test_statistic	p	-log2(p)
No	Yes	189.482865	4.122979e-43	140.799221

```
[178]: print_logrank('onlineSecurity')
```

		test_statistic	p	-log2(p)
No	Yes	141.60316	1.187554e-32	106.053706

```
[177]: print_logrank('streamingMovies')
```

		test_statistic	p	-log2(p)
No	Yes	17.941685	0.000023	15.422016

```
[168]: print_logrank('onlineSecurity')
```

		test_statistic	p	-log2(p)
No	Yes	141.60316	1.187554e-32	106.053706

```
[169]: print_logrank('gender')
```

```
[170]: print_logrank('seniorCitizen')
[170]: test_statistic      p   -log2(p)
 0.0  1.0    0.125471  0.723174  0.467584

[171]: print_logrank('partner')
[171]: test_statistic      p   -log2(p)
 No  Yes   135.758896  2.252911e-31  101.807981

[172]: print_logrank('dependents')
[172]: test_statistic      p   -log2(p)
 No  Yes   35.031241  3.244576e-09  28.199323

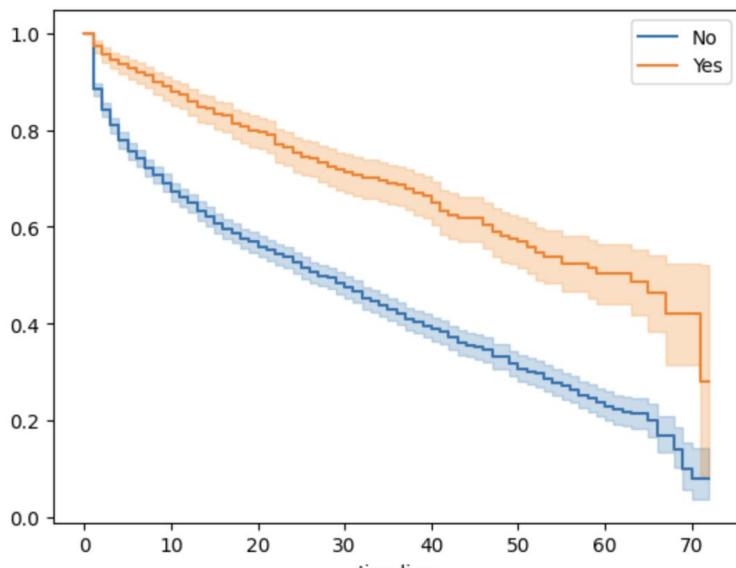
[173]: print_logrank('phoneService')
[173]: test_statistic      p   -log2(p)
 No  Yes   1.683709  0.194432  2.36266
```

```
[173]: print_logrank('phoneService')
[173]: test_statistic      p   -log2(p)
 No  Yes   1.683709  0.194432  2.36266

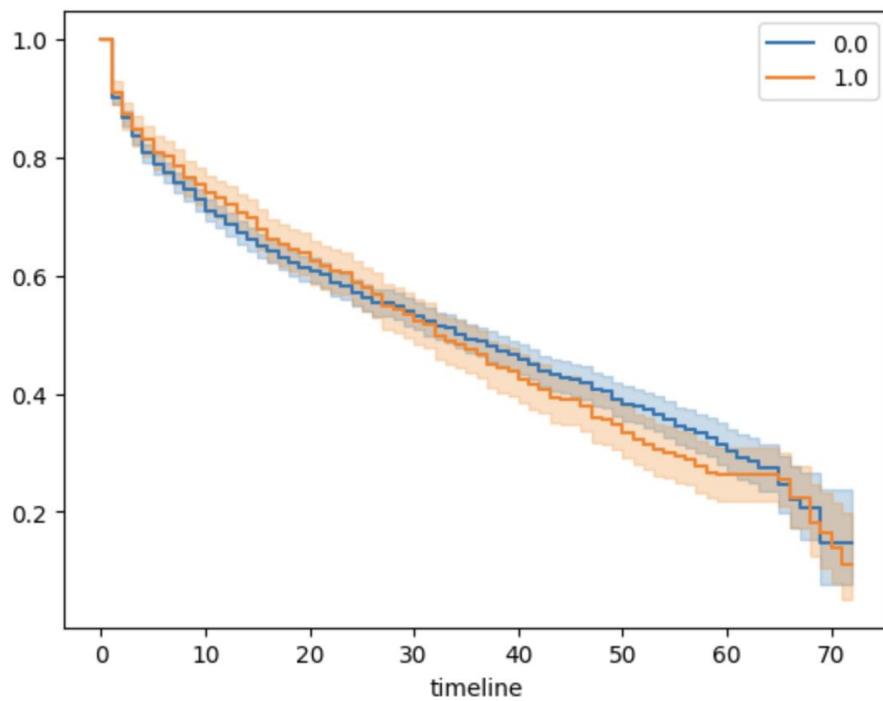
[174]: print_logrank('multipleLines')
[174]: test_statistic      p   -log2(p)
 No  No phone service   12.382712  4.333273e-04  11.172255
 Yes                         72.358368  1.794602e-17  55.629114
 No phone service          Yes   1.500291  2.206266e-01  2.180322
```

```
[175]: print_logrank('internetService')
[175]: test_statistic      p   -log2(p)
 DSL  Fiber optic   25.172866  5.241449e-07  20.863531
```

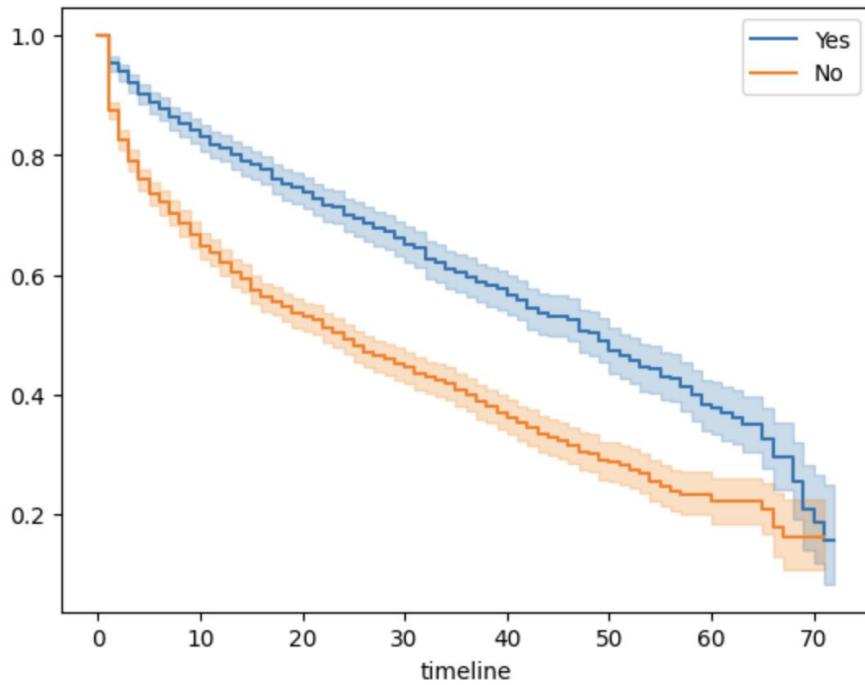
```
[164]: plot_km('onlineSecurity')
```



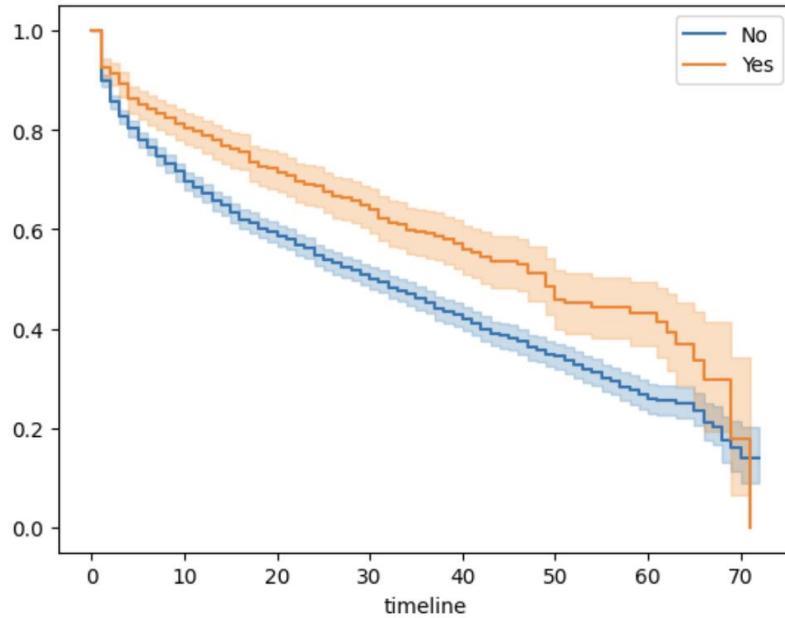
```
plot_km('seniorCitizen')
```



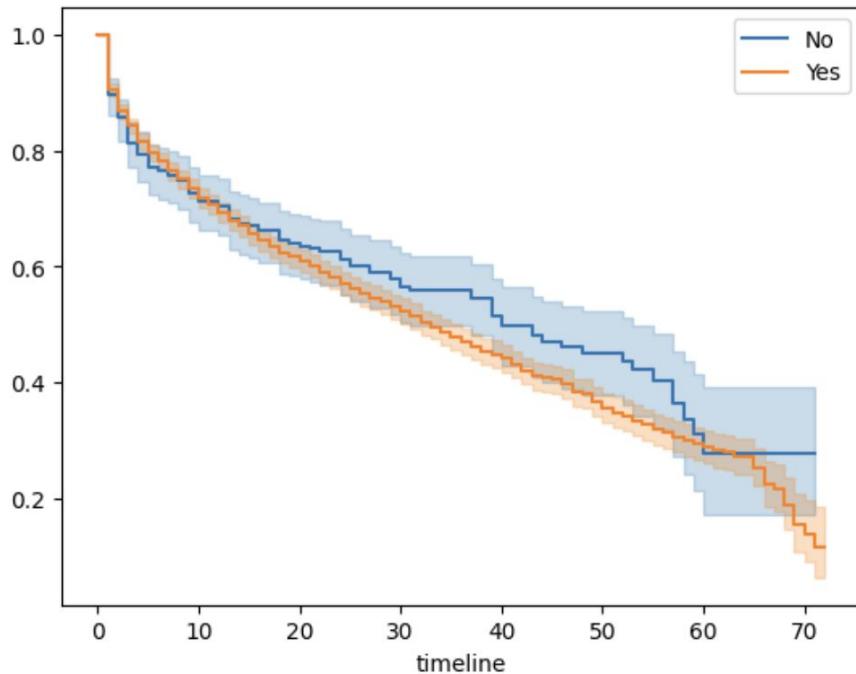
```
[]: plot_km('partner')
```



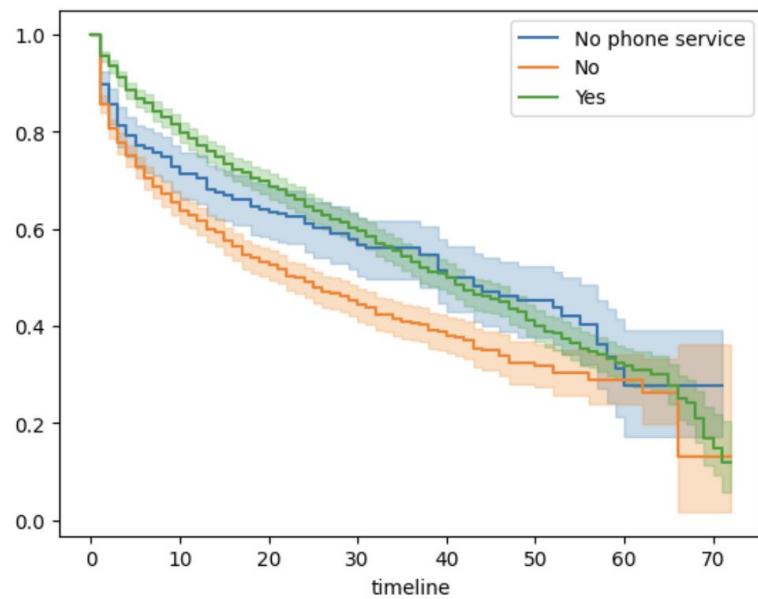
```
[184]: plot_km('dependents')
```



```
[185]: plot_km('phoneService')
```

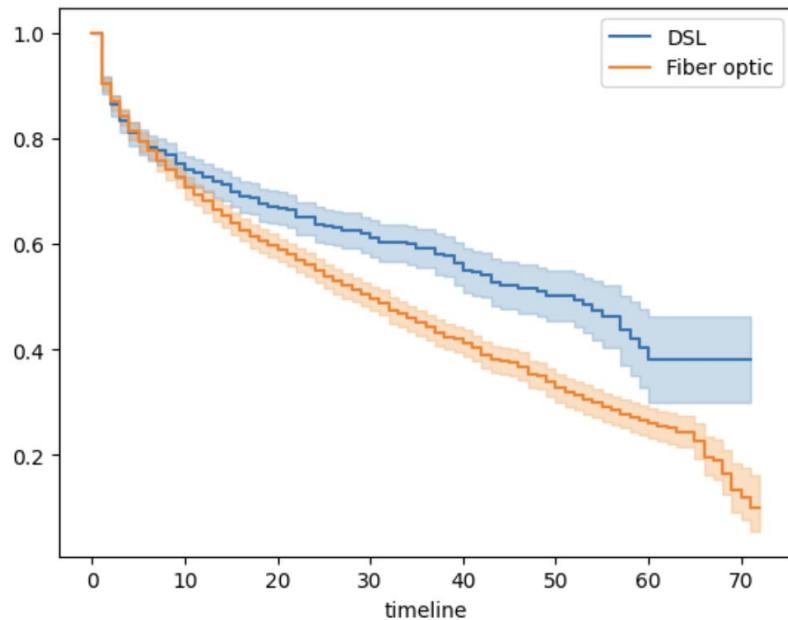


```
[186]: plot_km('multipleLines')
```

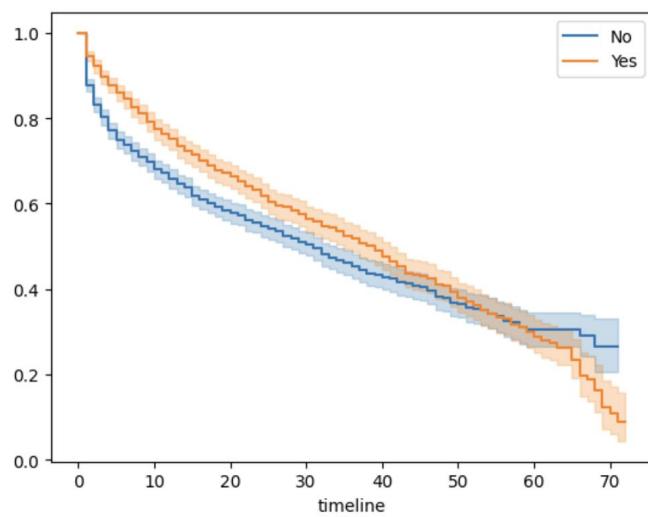


timeline

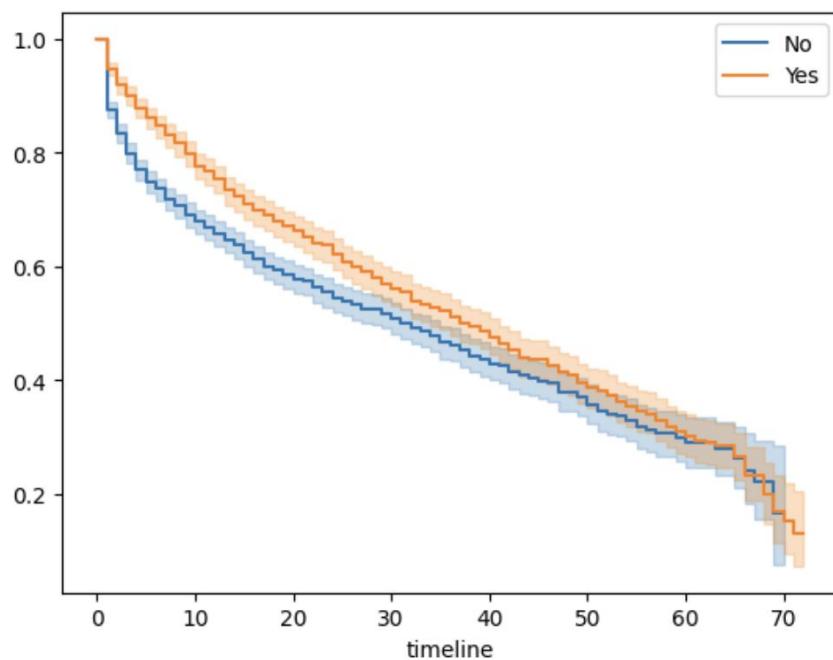
```
[187]: plot_km('internetService')
```



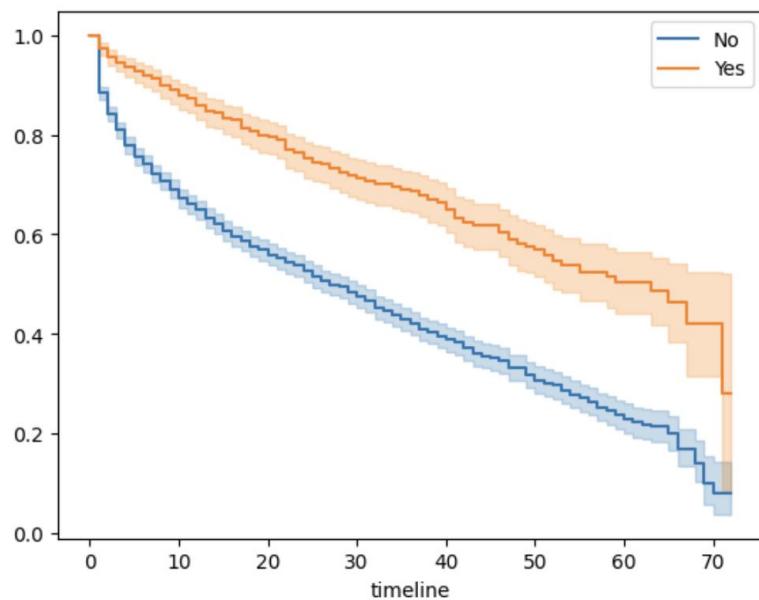
```
.88]: plot_km('streamingTV')
```



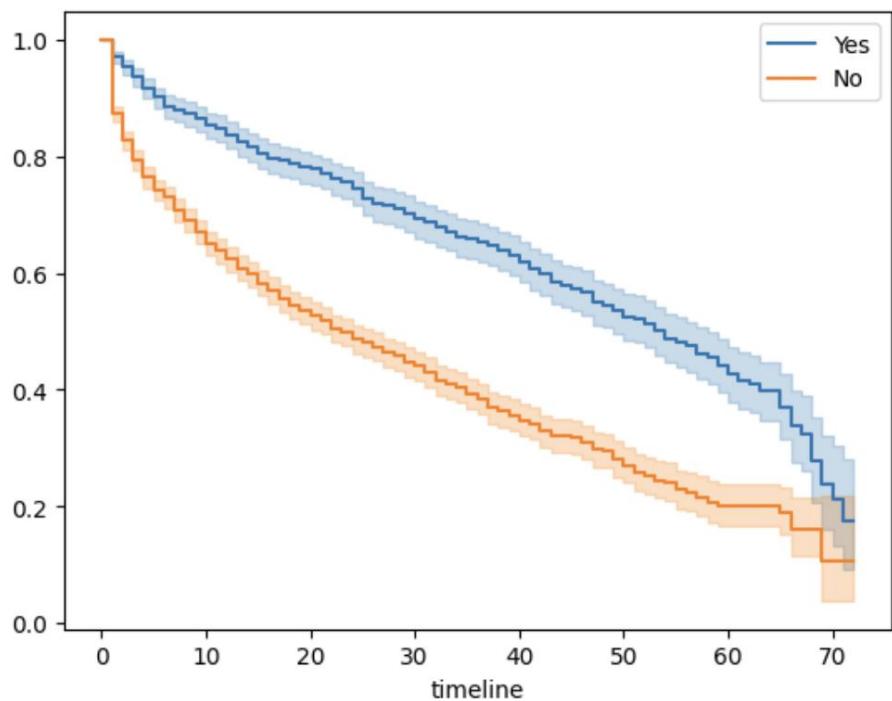
```
[89]: plot_km('streamingMovies')
```



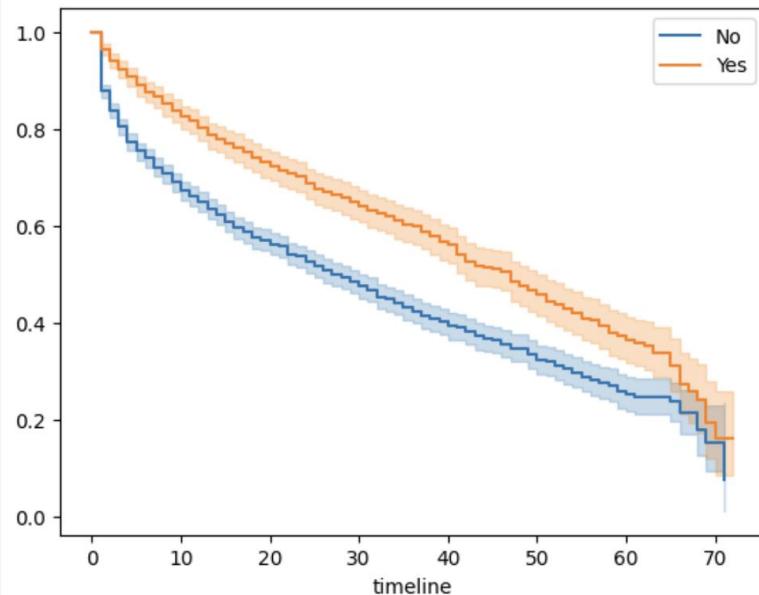
```
[90]: plot_km('onlineSecurity')
```



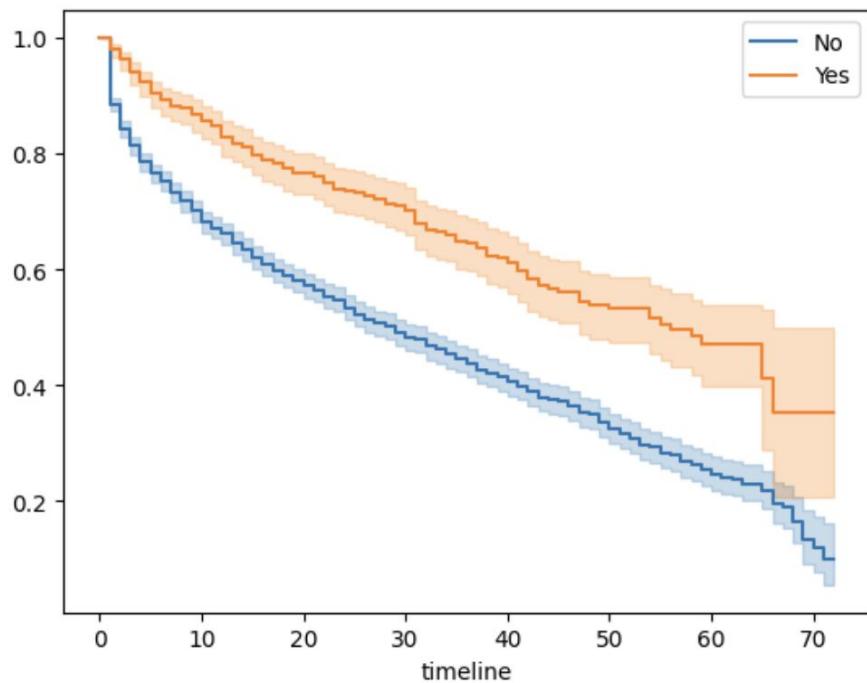
```
[91]: plot_km('onlineBackup')
```



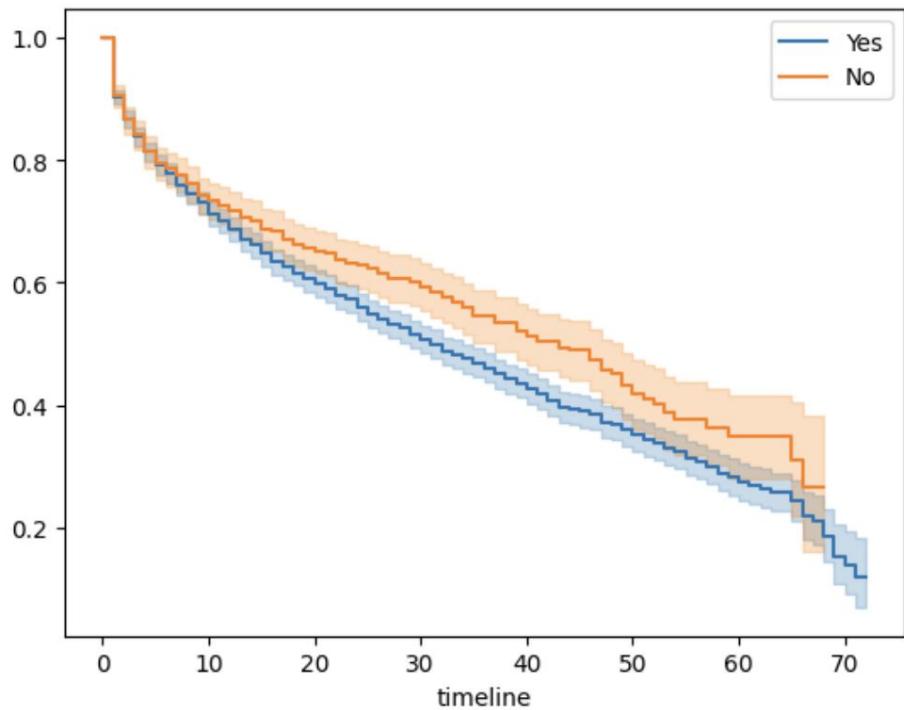
```
plot_km('deviceProtection')
```



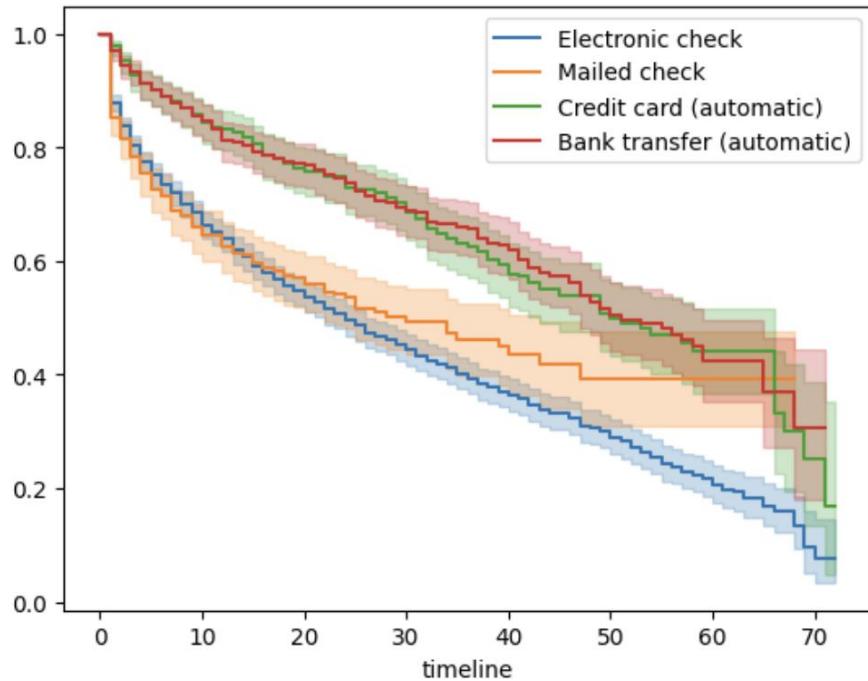
```
[]: plot_km('techSupport')
```



```
[]: plot_km('paperlessBilling')
```



```
[195]: plot_km('paymentMethod')
```



```
[105]: def get_survival_probs(col,val):  
    ix = telco_pd[col] == val  
    return kmf.fit(T[ix],C[ix],label=val)  
# Extract survival probabilities for customers with internetService = DSL  
sp_internet_dsl = get_survival_probs('internetService','DSL')  
pd.DataFrame(sp_internet_dsl.survival_function_at_times(range(0,10)))
```

```
[105]:  
      DSL  
0 1.000000  
1 0.902698  
2 0.864380  
3 0.834702  
4 0.810522  
5 0.794352  
6 0.783900  
7 0.776362  
8 0.768486  
9 0.750833
```

get_survival_probs(col, val)根据指定的协变量（如 internetService）及其特定值（如 DSL），筛选出该群体的客户数据，并计算其生存概率

```

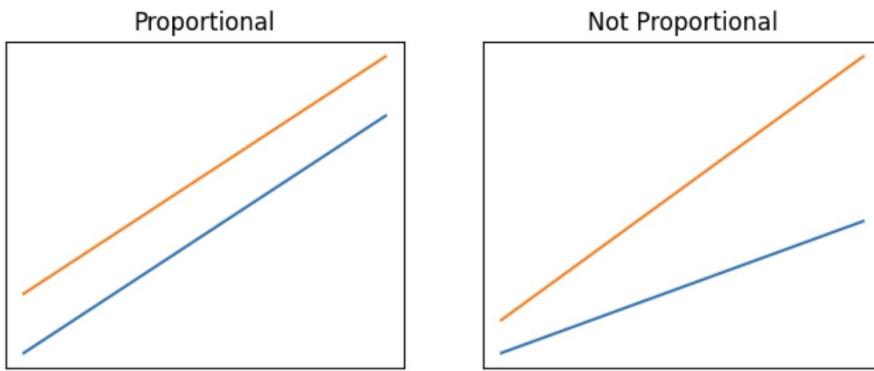
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(8,3))

ax1.plot([1,2,3,4,5],[1,2,3,4,5])
ax1.plot([1,2,3,4,5],[2,3,4,5,6])
ax1.title.set_text('Proportional')
ax1.axes.get_xaxis().set_visible(False)
ax1.axes.get_yaxis().set_visible(False)

ax2.plot([1,2,3,4,5],[1,2,3,4,5])
ax2.plot([1,2,3,4,5],[2,4,6,8,10])
ax2.title.set_text('Not Proportional')
ax2.axes.get_xaxis().set_visible(False)
ax2.axes.get_yaxis().set_visible(False)

```



图表分析： ax1 的数据展示了比例关系的变化，两个折线的增长趋势一致。

ax2 的数据展示了非比例关系的变化，第二条线增长更快，显示出不同比例的变化。

```

1]: # Import libraries
import pandas as pd
import seaborn as sns
import numpy as np

from lifelines.fitters.coxph_fitter import CoxPHFitter
from lifelines.statistics import proportional_hazard_test
from lifelines import KaplanMeierFitter
# Load the telco silver table
telco_pd = spark.table('silver_monthly_customers').toPandas()
# Encode columns of interest
encode_cols = ['dependents','internetService','onlineBackup','techSupport','paperlessBilling']

encoded_pd = pd.get_dummies(telco_pd,
                            columns=encode_cols,
                            prefix=encode_cols,
                            drop_first=False)

encoded_pd.head()

```

	customerID	gender	seniorCitizen	partner	tenure	phoneService	multipleLines	onlineSecurity	deviceProtection	streamingTV	... dependents_No	dependents_Y	
0	7590-VHVEG	Female	0.0	Yes	1.0	No	No phone service	No	No	No	...	True	Fal
1	3668-QPYBK	Male	0.0	No	2.0	Yes	No	Yes	No	No	...	True	Fal

`pd.get_dummies()`: 用于将指定的列（如 `dependents`、`internetService` 等）进行独热编码，生成新的列来表

示每个类别的可能值。`drop_first=False` 表示不会删除第一列（用于避免虚拟变量陷阱）。`prefix=encode_cols` 为每个新生成的列加上前缀，区分原始列名。

```
[123]: # Create new dataframe consisting of only the variables needed to fit the model
survival_pd = encoded_pd[['churn','tenure','dependents_Yes','internetService_DSL','onlineBackup_Yes','techSupport_Yes']]
# Cast churn column as a float
survival_pd.loc[:, 'churn'] = survival_pd.loc[:, 'churn'].astype('float')

[124]: cph = CoxPHFitter(alpha=0.05)
cph.fit(survival_pd, 'tenure', 'churn')

[124]: <lifelines.CoxPHFitter: fitted with 3351 total observations, 1795 right-censored observations>

[125]: cph.print_summary()

model          lifelines.CoxPHFitter
duration col    'tenure'
event col       'churn'
baseline estimation      breslow
number of observations      3351
number of events observed      1556
partial log-likelihood      -11315.95
time fit was run   2025-04-13 09:19:46 UTC

      coef  exp(coef)  se(coef)  coef lower 95%  coef upper 95%  exp(coef) lower 95%  exp(coef) upper 95%  cmp to      z      p  -log2(p)
```

`encoded_pd[['churn','tenure','dependents_Yes','internetService_DSL','onlineBackup_Yes','techSupport_Yes']]`: 从编码后的数据集中选择需要用于生存分析模型的变量。

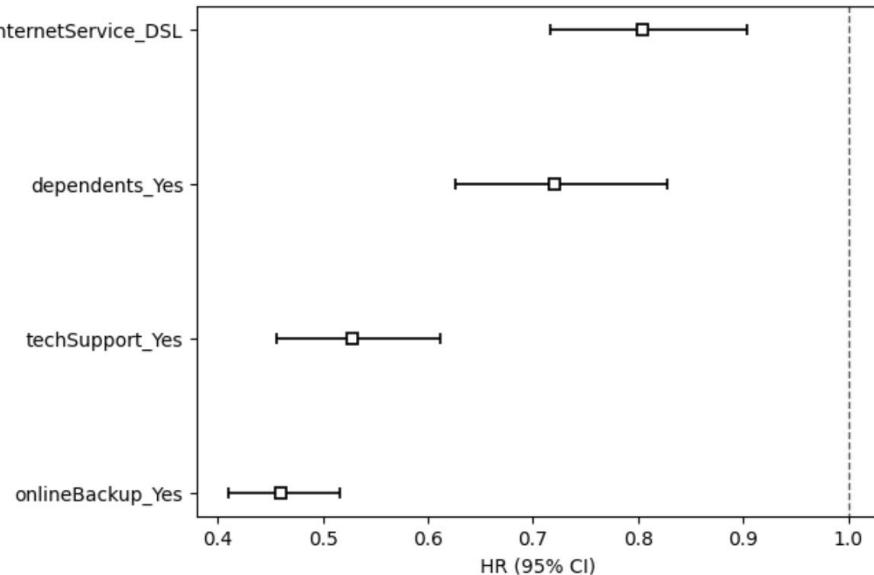
`survival_pd.loc[:, 'churn'] = survival_pd.loc[:, 'churn'].astype('float')`: 将 `churn` 列的数据类型转换为浮动数值型 (`float`)，因为 `Cox` 模型要求响应变量为数值型。

`CoxPHFitter(alpha=0.05)`: 初始化 `Cox` 比例风险模型，设置显著性水平 `alpha=0.05`（即置信区间为 95%）。`cph.fit(survival_pd, 'tenure', 'churn')`: 拟合 `Cox` 比例风险模型，其中 `tenure` 是时间变量（即生

存时间），`churn` 是事件变量（即是否流失）
`cph.print_summary()`: 输出模型的摘要，显示各个变量的回归系数、标准误差、z 值、p 值以及其他统计信息。

```
[126]: cph.plot(hazard_ratios=True)
```

```
[126]: <Axes: xlabel='HR (95% CI)'>
```



`cph.plot(hazard_ratios=True)`: 此函数用来绘制 Cox 比例风险模型的结果图，显示每个特征（变量）的风险比（hazard ratio）。`hazard_ratios=True` 参数指示模型应绘制风险比而非回归系数。

```
128]: cph.check_assumptions(survival_pd,p_value_threshold=0.05,show_plots=True)
```

```
Bootstrapping lowess lines. May take a moment...
```

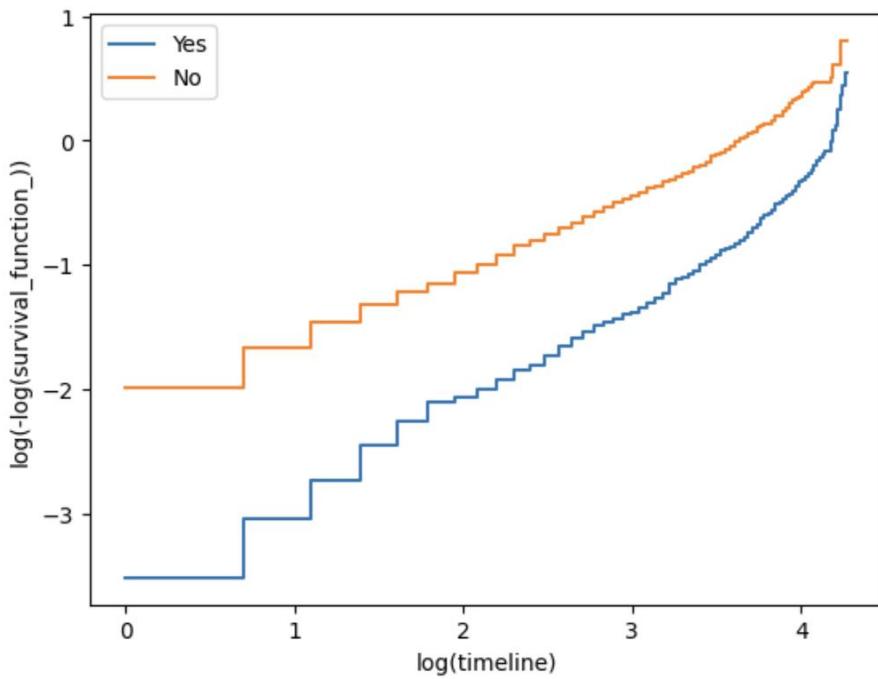
```
Bootstrapping lowess lines. May take a moment...
```

```
The ``p_value_threshold`` is set at 0.05. Even under the null hypothesis of no violations, some covariates will be below the threshold by chance. This is compounded when there are many covariates. Similarly, when there are lots of observations, even minor deviances from the proportional hazard assumption will be flagged.
```

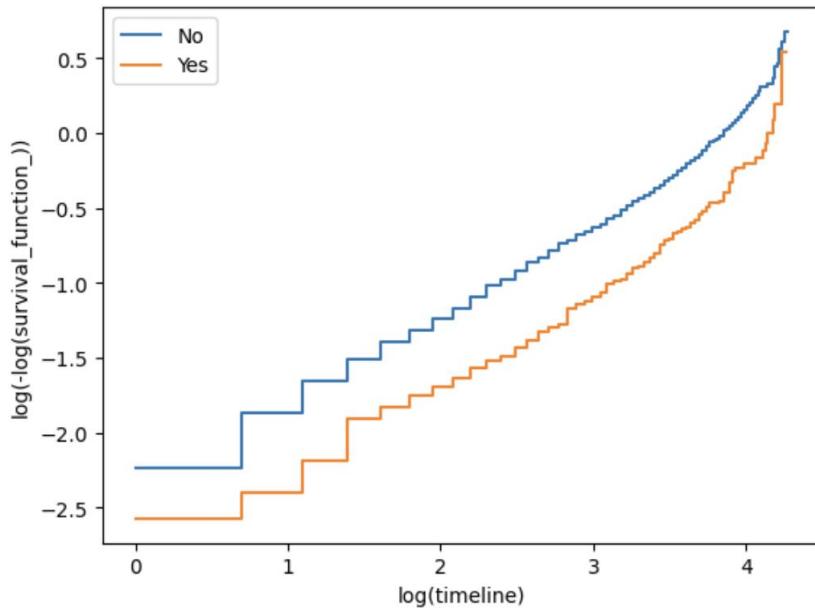
```
With that in mind, it's best to use a combination of statistical tests and visual tests to determine the most serious violations. Produce visual plots using ``check_assumptions(..., show_plots=True)`` and looking for non-constant lines. See link [A] below for a full example.
```

`cph.check_assumptions(survival_pd, p_value_threshold=0.05, show_plots=True)`: 此函数用于检查 Cox 比例风险模型的假设是否成立。Cox 模型的核心假设是“比例风险假设”，即各个特征的效应在整个时间范围内是恒定的。该函数通过对模型拟合的数据进行各种检验，帮助判断这一假设是否成立。

```
[1]: plot_km_loglog('onlineBackup')
```



```
[132]: plot_km_loglog('dependents')
```



```

[1]: survival_pd = encoded_pd[['churn','tenure', 'partner_Yes', 'multipleLines_Yes', \
                             'internetService_DSL', 'onlineSecurity_Yes', 'onlineBackup_Yes', 'deviceProtection_Yes', 'techSupport_Yes', \
                             'paymentMethod_Bank transfer (automatic)', 'paymentMethod_Credit card (automatic)']]

[2]: aft = LogLogisticAFTFitter()
aft.fit(survival_pd, duration_col='tenure', event_col='churn')

[3]: <lifelines.LogLogisticAFTFitter: fitted with 3351 total observations, 1795 right-censored observations>

[4]: # Note: the output is exponentiated because it is initially in log-odds
print("Median Survival Time:{:.2f}".format(np.exp(aft.median_survival_time_)))

Median Survival Time:135.51

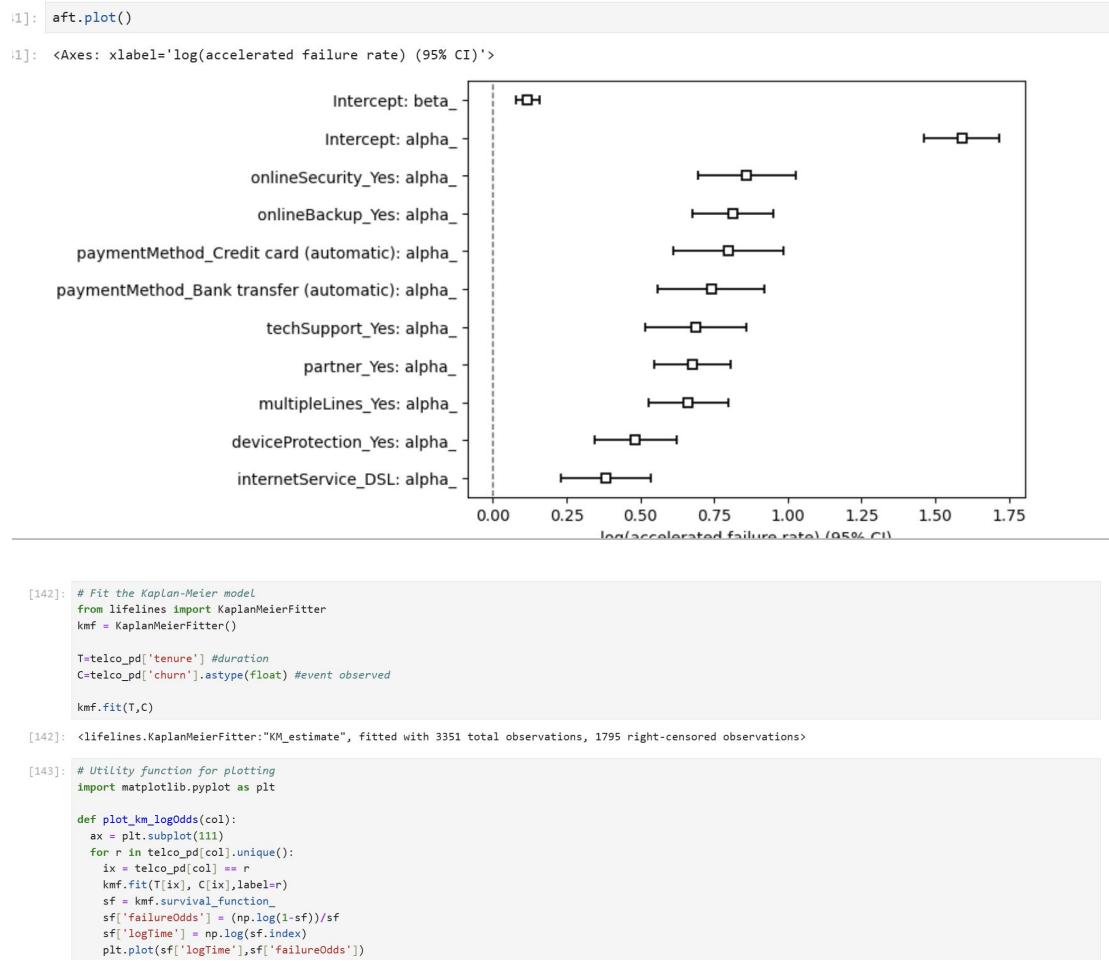
[5]: aft.print_summary()

      model    lifelines.LogLogisticAFTFitter
duration col          'tenure'
event col            'churn'
number of observations      3351
number of events observed     1556
log-likelihood        -6838.36
time fit was run   2025-04-13 09:22:06 UTC

```

aft.fit(survival_pd,duration_col='tenure',event_col='churn'): 使用加速失效时间（AFT）模型对生存数据进行拟合。`duration_col='tenure'`指定了表示生存时间的列，`event_col='churn'`指定了事件发生的列（在这个例子中是客户是否流失）。此函数会拟合模型并估计各个特征对生存时间的影响。

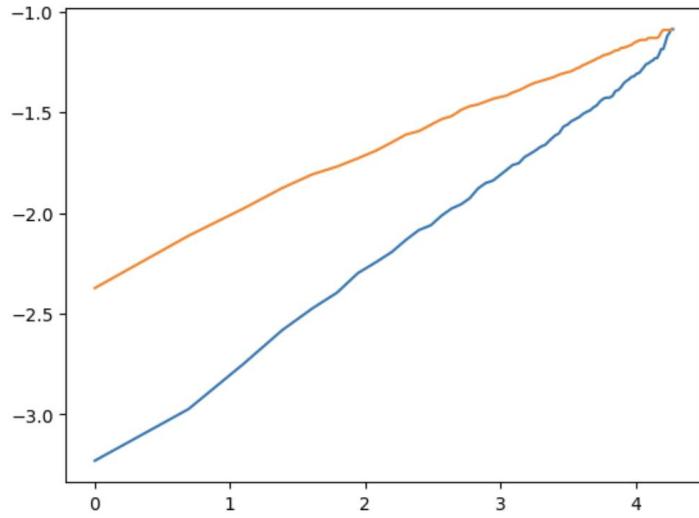
np.exp(aft.median_survival_time_): 由于模型的输出是以对数几率形式表示的，所以通过对中位生存时间进行指数运算，得到实际的生存时间。输出是生存时间的中位数



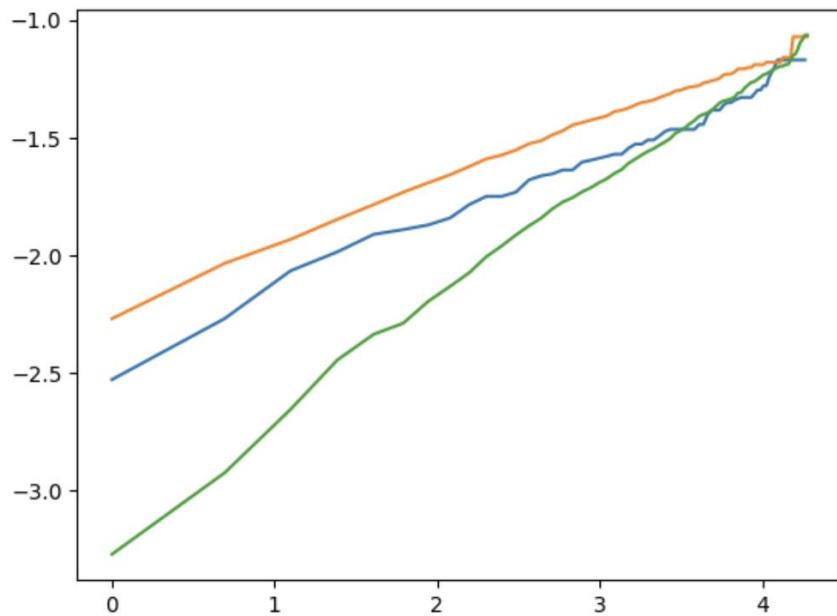
plot_km_logOdds(col): 绘制 Kaplan-Meier 生存曲线的对数时间与失败赔率的关系。对每个组进行 Kaplan-Meier 生存曲线拟合。

计算失败赔率（failure odds），并绘制对数时间（log-time）与失败赔率的图像。

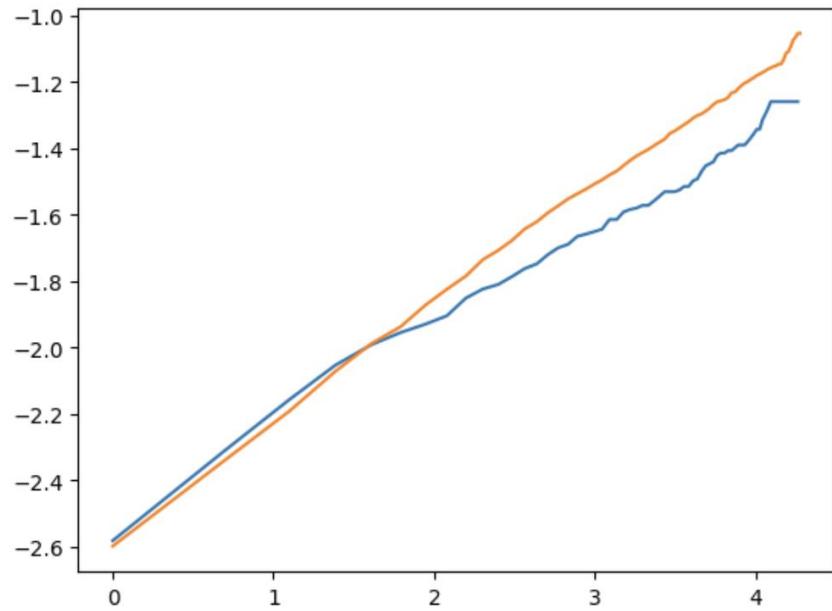
```
[1]: plot_km_logOdds('partner')
```



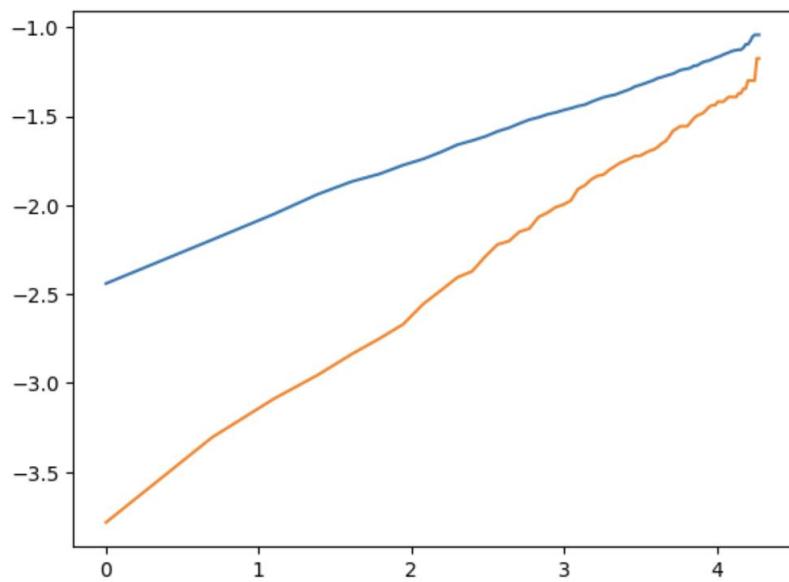
```
[145]: plot_km_logOdds('multipleLines')
```



```
[146]: plot_km_logOdds('internetService')
```



```
[147]: plot_km_logOdds('onlineSecurity')
```



```
[158]: import ipywidgets as widgets
from IPython.display import display

# 要创建的小部件列名
cols = ['dependents_Yes','internetService_DSL','onlineBackup_Yes','techSupport_Yes','partner_Yes','internal rate of return']

# 创建用于保存小部件的字典
widgets_dict = {}

# 遍历每一个字段创建相应的Widget
for col in cols:
    if col.strip() == 'internal rate of return': # 去掉可能的空格
        widgets_dict[col] = widgets.Text(
            value='0.10',
            description=col.strip(),
            style={'description_width': 'initial'}
        )
    else:
        widgets_dict[col] = widgets.Dropdown(
            options=['0', '1'],
            value='0',
            description=col,
            style={'description_width': 'initial'}
        )
display(widgets_dict[col])
```

widgets.Text: 用于创建文本框，用户可以在其中输入值。例如，对于 **internal rate of return**（内部收益率），创建了一个文本框，默认值为 **0.10**。

widgets.Dropdown: 用于创建下拉框，用户可以选择值。这里为每个二元变量（例如 **dependents_Yes**, **internetService_DSL** 等）创建了下拉框，选项为 **0** 或 **1**，默认值为 **0**。

```
] : import pandas as pd

# 从 ipywidgets 获取输入值构造 DataFrame
def get_widget_values():
    widget_dict = {col: widgets_dict[col].value for col in cols}
    return pd.DataFrame.from_dict(widget_dict, orient='index').T
def get_payback_df():
    df = get_widget_values()

    # 注意：列名中有空格的地方处理一下
    irr = float(df['internal rate of return'][0]) / 12

    # 获取生存函数预测 (注意结果是 DataFrame)
    surv_df = round(cph.predict_survival_function(df), 2)

    # 添加初始时刻的生存概率为 1.0
    cohort_df = pd.concat([pd.DataFrame([1.0]), surv_df]).rename(columns={0: 'Survival Probability'})

    cohort_df['Contract Month'] = cohort_df.index.astype('int')
    cohort_df['Monthly Profit for the Selected Plan'] = 30
    cohort_df['Avg Expected Monthly Profit'] = round(cohort_df['Survival Probability'] * cohort_df['Monthly Profit for the Selected Plan'], 2)
    cohort_df['NPV of Avg Expected Monthly Profit'] = round(cohort_df['Avg Expected Monthly Profit'] / ((1 + irr) ** cohort_df['Contract Month']), 2)
    cohort_df['Cumulative NPV'] = cohort_df['NPV of Avg Expected Monthly Profit'].cumsum()
    cohort_df['Contract Month'] = cohort_df['Contract Month'] + 1
```

```

return cohort_df[['Contract Month',
                  'Survival Probability',
                  'Monthly Profit for the Selected Plan',
                  'Avg Expected Monthly Profit',
                  'NPV of Avg Expected Monthly Profit',
                  'Cumulative NPV']].set_index('Contract Month')
from IPython.display import display
import matplotlib.pyplot as plt

def on_calculate_clicked(b):
    df_result = get_payback_df()
    display(df_result)

# 简单画图示例（比如绘制累计NPV）
df_result['Cumulative NPV'].plot(title='Cumulative NPV over Contract Months', figsize=(10,5))
plt.xlabel('Contract Month')
plt.ylabel('Cumulative NPV')
plt.grid(True)
plt.show()

calculate_button = widgets.Button(description="计算现金流回收表")
calculate_button.on_click(on_calculate_clicked)
display(calculate_button)

```

get_widget_values():

获取小部件（`widgets`）中的所有值，返回一个 `DataFrame`。通过 `widgets_dict` 字典保存小部件的值，每个小部件的值将作为 `DataFrame` 的一行。

`get_payback_df()`: 调用 `get_widget_values()` 获取输入值，计算生存概率、预期月利润、NPV（净现值）和累计 NPV，最终返回包含这些计算结果的 `DataFrame`。

`on_calculate_clicked(b)`: 这是按钮点击事件的回调函数。当点击“计算现金流回收表”按钮时，会调用 `get_payback_df()` 计算并展示结果。

同时，通过 `matplotlib` 绘制累计 NPV 随着合约月份的变 化 图 表。

```
[160]: pd.options.display.max_rows = 25
get_payback_df().iloc[0:25]
```

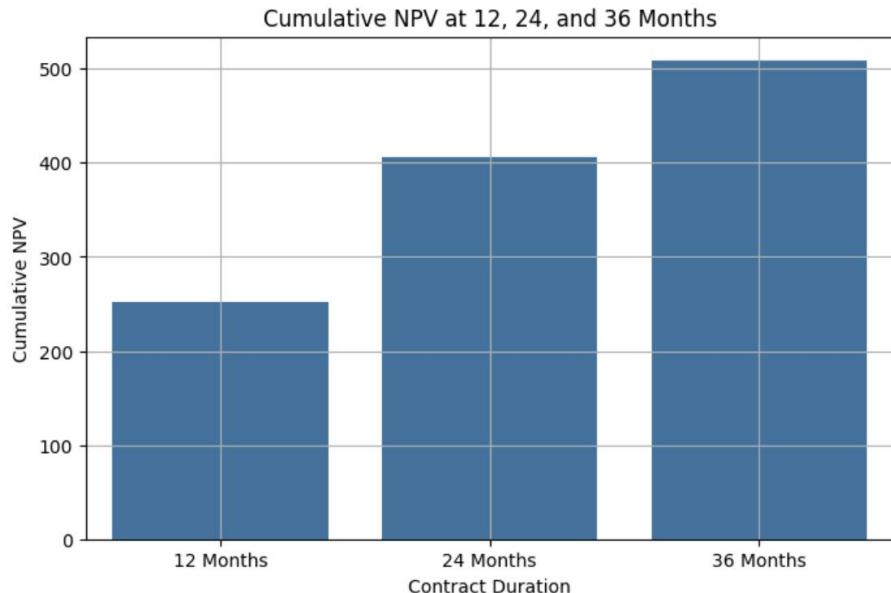
Contract Month	Survival Probability	Monthly Profit for the Selected Plan	Avg Expected Monthly Profit	NPV of Avg Expected Monthly Profit	Cumulative NPV
1	1.00	30	30.0	30.00	30.00
2	0.87	30	26.1	25.88	55.88
3	0.81	30	24.3	23.90	79.78
4	0.77	30	23.1	22.53	102.31
5	0.74	30	22.2	21.48	123.79
6	0.71	30	21.3	20.43	144.22
7	0.69	30	20.7	19.69	163.91

```
[]: import seaborn as sns
import matplotlib.pyplot as plt

# 从结果中选出第12、24、36个月的累计NPV
payback_df = get_payback_df()
selected_months = [11, 23, 35] # 注意: Python 是从 0 开始索引的
labels = ['12 Months', '24 Months', '36 Months']
npv_values = payback_df.iloc[selected_months]['Cumulative NPV'].values

# 构建一个新的 DataFrame 用于画图
plot_df = pd.DataFrame({
    'Contract Duration': labels,
    'Cumulative NPV': npv_values
})

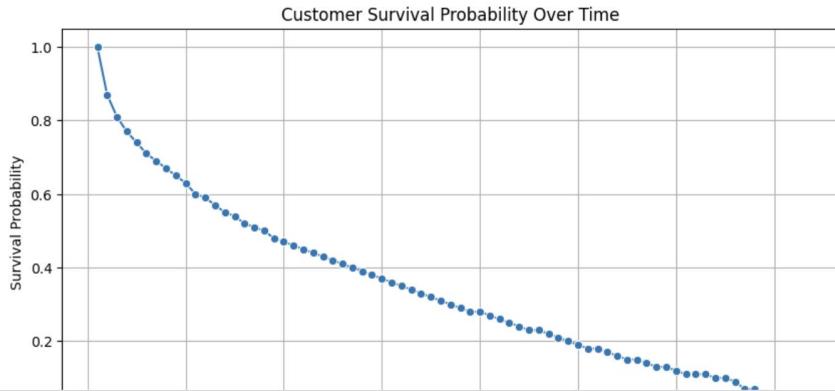
# 画柱状图
plt.figure(figsize=(8,5))
ax = sns.barplot(x='Contract Duration', y='Cumulative NPV', data=plot_df)
ax.set_title('Cumulative NPV at 12, 24, and 36 Months')
plt.ylabel("Cumulative NPV")
plt.xlabel("Contract Duration")
plt.grid(True)
plt.show()
```



评估投资的回报：累计净现值（Cumulative NPV）是一个重要的财务指标，用于衡量项目在不同时间段的

整体回报。

```
# 绘制生存概率曲线图
plt.figure(figsize=(10, 5))
sns.lineplot(x=payback_df.index, y=payback_df['Survival Probability'], marker='o')
plt.title('Customer Survival Probability Over Time')
plt.xlabel('Contract Month')
plt.ylabel('Survival Probability')
plt.ylim(0, 1.05) # 生存概率范围通常在 0~1
plt.grid(True)
plt.show()
```



主要分析目的：生存概率的变化：

生存概率曲线反映了客户在合同期内保持活跃（即没有流失）的概率。随着时间的推移，生存概率应该会逐渐下降，表示客户流失的概率增加。客户流失的趋势：从图中，我们可以看到随着时间的增加，客户的生存概率逐渐减少。这可能意味着在合同期后期，流失率增加，客户的流失情况更加明显。合同期的评估：通过生存概率的变化，投资者或业务人员可以评估合同期的设计是否合理。如果生存概率在较早阶段就大幅下降，可能意味着需要调整客户保持策略或合同条款。

