به‌نام خداوند بخشنده و مهربان

تمرین چهارم: لاجستیک رگرسیون
درس: تحلیل سیستم داده‌های حجیم

استاد: محمدعلی نعمت‌بخش
دستیاران: فاطمه ابراهیمی، پریسا لطیفی، امیر سرتیپی

نام و نام‌خانوادگی: نگین شمس
آدرس گیت: https://github.com/NeginShams/Spark_ML

- **لطفا پاسخ تمارین حتما در سامانه‌ی کوئرا ارسال شود.**
- **لطفا پاسخ‌های خود را در خود سند سوال نوشته و همراه نوتبوک تمرین در کوئرا ارسال کنید.**
- **نام سند ارسالی HW-{homework number}-{Name Family}-{student number}**
- **تمامی فایل‌های مورد نیاز این تمرین در این لینک قابل دسترس است.**
- **خروجی از هر مرحله‌ی تمرین را در سند خود بارگذاری کنید.**

در این تمرین هدف کار با کتاب‌خانه‌ی pyspark و همچنین کتاب‌خانه‌ی یادگیری ماشین آن است.
برای این منظور دیتاستی در اختیار شما قرار گرفته است. اطلاعات کاربران شرکتی در اختیار شما قرار داده شده است. این شرکت اطلاعات چند ماه از کاربرانش را برچسب گذاری کرده است. این برچسب به معنای این است که آیا مشتری شرکت را ترک کرده و دیگر از خدمات آن استفاده می‌کند یا خیر.

- **قدم اول:** دیتاست داده شده را پیش پردازش کنید. مقادیر NA را مقدار دهی کنید تحلیل داده اکتشافی (EDA) را به خوبی انجام دهید. این ستونها براساس ماهیت خود میتواند تولید کننده ویژگیهای بیشتری باشند که ممکن است دقت مدل شما را بالاتر ببرند. در این مرحله همبستگی و ارتباط بین تمام ویژگی هایی که میتوانید استخراج کنید را بررسی کنید. (نمودارهای لازم برای تحلیل دادگان ترسیم شود.)

در ابتدا با استفاده از دستور شکل زیر یک spark session ایجاد می‌شود.

```
import pyspark
from pyspark.sql import SparkSession

sc = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

سپس مطابق شکل زیر با استفاده از دستور sc.read مجموعه داده خوانده می‌شود و در قالب متغیری از نوع pyspark.dataframe ذخیره می‌شود. برای این دستور تعدادی option تعریف شده است. header نشان‌دهنده‌ی عنوان ستون‌ها است و مقدار آن برابر True است. inferschema گزینه‌ای است که اگر مقدار آن برابر false باشد، تمام ستون‌ها به‌صورت متغیر رشته‌ای در نظر گرفته می‌شوند و بنابراین لازم است نوع داده[1] برای هر ستون به‌صورت دستی مشخص شود. در صورتی‌که مقدار این گزینه True باشد، نوع هر متغیر در هنگام خواندن مجموعه داده به‌صورت خودکار مشخص می‌شود. گزینه‌ی sep نیز به‌معنای این است که مقادیر مختلف در مجموعه داده توسط چه کاراکتری جدا می‌شوند. با توجه به این‌که فایل مجموعه داده از نوع csv[2] است، مقدار این گزینه «,» قرار گرفته است.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[4] data = sc.read \
      .option('header', 'True')\
      .option('inferSchema', 'True')\
      .option('sep', ',')\
      .csv('/content/drive/MyDrive/data.csv')
```

با توجه به این‌که دستور shape برای دیتافریم‌های pyspark تعریف نشده است، برای به‌دست آوردن تعداد سطر و ستون در مجموعه داده، از دستور زیر استفاده شده است.

```
print('number of rows: '+ str(data.count()))
print('number of columns: '+ str(len(data.columns)))
```

```
number of rows: 229990
number of columns: 21
```

برای مشاهده‌ی توزیع داده‌ها در دو کلاس مورد نظر، می‌توان از دستور شکل زیر استفاده کرد. با توجه به نتیجه‌ی این دستور، برچسب حدود ۸۵ درصد از داده‌ها برابر «No» می‌باشد یعنی این افراد شرکت را ترک نکرده‌اند. هم‌چنین حدود ۱۴ درصد از داده‌ها برچسب «Yes» دارند. بنابراین، این مجموعه داده متوازن نیست. این موضوع می‌تواند مدل را در تشخیص صحیح دچار مشکل کند.

[1] Data Type
[2] Comma Separated Value

```
data.groupby('Label').count().show()
```

```
+-----+------+
|Label| count|
+-----+------+
| null|   208|
|   No|195878|
|  Yes| 33904|
+-----+------+
```

برای مشاهده‌ی نوع داده‌ی مربوط به هر ویژگی می‌توان از دستور dtypes استفاده نمود. با توجه به نتیجه‌ی اجرای این دستور مشاهده می‌شود که چهار ستون از مجموعه یعنی مقادیر مربوط به ویژگی‌های SeniorCitizen، tenure، MonthlyCharges و TotalCharges از نوع عددی می‌باشند و سایر ویژگی‌ها از نوع رشته هستند. در ادامه به‌صورت جداگانه به تحلیل ویژگی‌های عددی و غیرعددی پرداخته خواهد شد.

```
data.dtypes
```

```
[('customerID', 'string'),
 ('gender', 'string'),
 ('SeniorCitizen', 'double'),
 ('Partner', 'string'),
 ('Dependents', 'string'),
 ('tenure', 'double'),
 ('PhoneService', 'string'),
 ('MultipleLines', 'string'),
 ('InternetService', 'string'),
 ('OnlineSecurity', 'string'),
 ('OnlineBackup', 'string'),
 ('DeviceProtection', 'string'),
 ('TechSupport', 'string'),
 ('StreamingTV', 'string'),
 ('StreamingMovies', 'string'),
 ('Contract', 'string'),
 ('PaperlessBilling', 'string'),
 ('PaymentMethod', 'string'),
 ('MonthlyCharges', 'double'),
 ('TotalCharges', 'double'),
 ('Label', 'string')]
```

با توجه به شکل زیر، مجموعه با توجه به ویژگی‌های عددی و غیرعددی، به دو مجموعه‌ی مجزا تقسیم می‌شود.

```
numeric_features = ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']

categorical_columns = []
for col in data.columns:
  if col not in numeric_features:
    categorical_columns.append(col)

numeric_df = data.drop(*tuple(categorical_columns))
categorical_df = data.drop(*tuple(numeric_features))
```

در ابتدا ویژگی‌های عددی مورد بررسی قرار می‌گیرد. با استفاده از تابع describe می‌توان اطلاعات مفیدی درباره داده‌ها به‌دست آورد.

```
numeric_df.describe().show()
```

```
+-------+------------------+-----------------+------------------+-----------------+
|summary|     SeniorCitizen|           tenure|    MonthlyCharges|     TotalCharges|
+-------+------------------+-----------------+------------------+-----------------+
|  count|            229724|           229765|            229747|           229739|
|   mean|0.21505371663387368|49.43501838835332|126.33354646633397|3700.323704725759|
| stddev| 0.8971977539088728|36.63299614293823| 856.9228133633635|2326.036263404356|
|    min|               0.0|           -598.0|             18.25|             18.8|
|    max|              17.0|             72.0|           14990.0|           8684.8|
+-------+------------------+-----------------+------------------+-----------------+
```

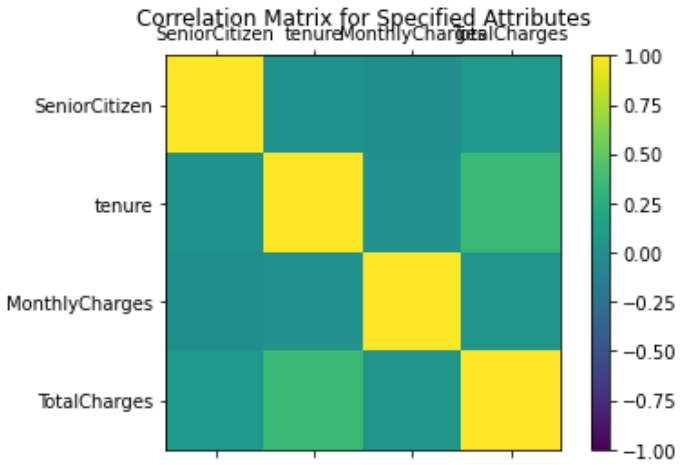
به‌منظور به‌دست آوردن همبستگی (بدون تبدیل دیتافریم به pandas.dataframe) بین ویژگی‌های عددی، از قطعه کد قابل مشاهده در شکل زیر استفاده شده است. برخی از ویژگی‌هایی که همبستگی بالایی دارند اضافی هستند و می‌توان یکی از آن‌ها را حذف نمود.

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.stat import Correlation

my_cols =numeric_df.select(numeric_df.columns)
new_df = my_cols.na.drop()

vector_col = "corr_features"
assembler = VectorAssembler(inputCols=numeric_features,
                            outputCol=vector_col)
myGraph_vector = assembler.transform(new_df).select(vector_col)
matrix = Correlation.corr(myGraph_vector, vector_col)

matrix = Correlation.corr(myGraph_vector, vector_col).collect()[0][0]
corrmatrix = matrix.toArray().tolist()

df = sc.createDataFrame(corrmatrix,numeric_features)
df.show()
```

```
+--------------------+--------------------+--------------------+--------------------+
|       SeniorCitizen|              tenure|      MonthlyCharges|       TotalCharges|
+--------------------+--------------------+--------------------+--------------------+
|                 1.0|0.021968033406082374|-2.35081944281545...|0.07035394107959646|
|0.021968033406082374|                 1.0|0.015090136155424226|0.35527680076386325|
|-2.35081944281545...|0.015090136155424226|                 1.0|0.06140509301362273|
| 0.07035394107959646| 0.35527680076386325| 0.06140509301362273|                 1.0|
+--------------------+--------------------+--------------------+--------------------+
```

هم‌چنین برای رسم نمودار ماتریس همبستگی از کد زیر استفاده شده است. با توجه به نتایج، بیش‌ترین همبستگی مربوط به ویژگی‌های tenure و TotalCharges با مقدار ۰.۳۵ می‌باشد.

```python
import matplotlib.pyplot as plt

def plot_corr_matrix(correlations,attr,fig_no):
    fig=plt.figure(fig_no)
    ax=fig.add_subplot(111)
    ax.set_title("Correlation Matrix for Specified Attributes")
    ax.set_xticklabels(['']+attr)
    ax.set_yticklabels(['']+attr)
    cax=ax.matshow(correlations,vmax=1,vmin=-1)
    fig.colorbar(cax)
    plt.show()

plot_corr_matrix(corrmatrix, numeric_features, 234)
```



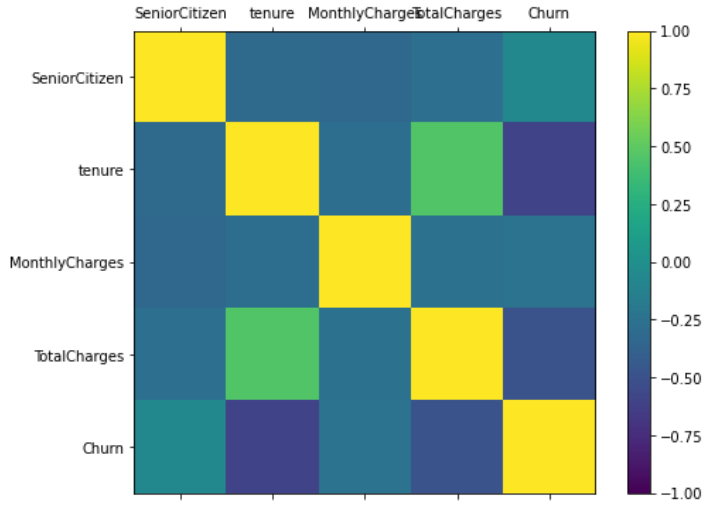
هم‌چنین دانستن میزان همبستگی بین سایر ویژگی‌ها با ویژگی برچسب می‌تواند سودمند باشد. هر چه همبستگی یک ویژگی با برچسب بیش‌تر باشد، آن ویژگی سودمندتر است. برای محاسبه همبستگی ویژگی برچسب با سایر ویژگی‌ها، لازم است نوع این ویژگی از رشته به عدد تبدیل شود. به همین منظور، از یک تابع udf[3] استفاده شده است که مقادیر «No» را به صفر و مقادیر «Yes» را به یک تبدیل می‌کند. ساختار این تابع در شکل زیر قابل مشاهده است.

```python
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

y_udf = udf(lambda y:0 if y=='No' else 1, IntegerType())
df = data.withColumn('Churn', y_udf('Label')).drop('Label')
df = df.drop(*tuple(categorical_columns))
```

سپس می‌توان همبستگی را بین این ویژگی با سایر ویژگی‌ها محاسبه نمود. با توجه به نتایج به‌دست‌آمده، بیشترین همبستگی مربوط به ویژگی‌های tenure و totalCharges می‌باشد.

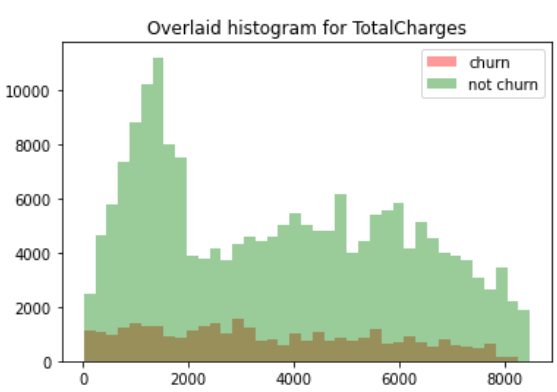
[3] User Defined Funcion

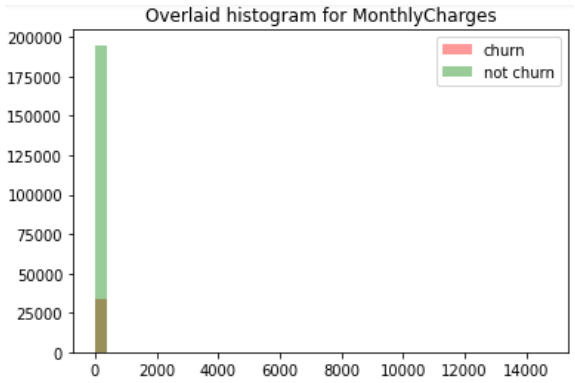
SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn
1.0	-0.30812081595993385	-0.33269149338862763	-0.27004203828113194	-0.06928904197775006
-0.30812081595993385	1.0	-0.2808845267851059	0.4568341249514723 5	-0.5963793102723218
-0.33269149338862763	-0.2808845267851059	1.0	-0.2514596772333894	-0.23512255699051182
-0.27004203828113194	0.45683412495147235	-0.2514596772333894	1.0	-0.4883760070158531
-0.06928904197775006	-0.5963793102723218	-0.23512255699051182	-0.4883760070158531	1.0



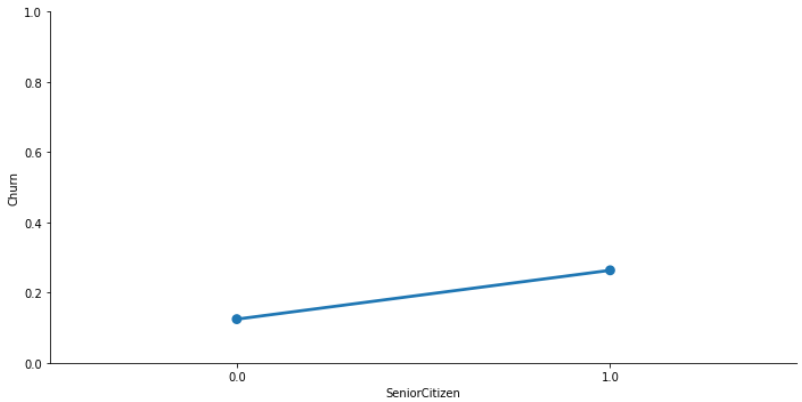
در بین ویژگی‌های عددی، تنها دو ویژگی TotalCharges و MonthlyCharges از نوع متغیر پیوسته هستند. با استفاده از قطعه کد قابل مشاهده در شکل زیر، می‌توان نمودار overlaid histogram را برای این ویژگی‌ها ترسیم نمود. با توجه به این نمودارها، ویژگی MonthlyCharges دریک محدوده‌ی کوچک قرار دارد و توزیع آن برای افرادی که شرکت را ترک کرده یا نکرده‌اند، یکسان است. هم‌چنین توزیع داده‌ها در نمودار مربوط به ویژگی TotalCharges نیز تقریبا یکسان است.

```
import numpy as np
from pyspark.sql.functions import col

for i in ['MonthlyCharges', 'TotalCharges']:
    churn = df.filter(col("Churn")==1).select(i)
    not_churn = df.filter(col("Churn")==0).select(i)
    xmin = min(churn.agg({i:'min'}).collect()[0][0], not_churn.agg({i:'min'}).collect()[0][0])
    xmax = max(churn.agg({i:'max'}).collect()[0][0], not_churn.agg({i:'max'}).collect()[0][0])
    width = (xmax - xmin) / 40
    sns.distplot(churn.toPandas(), color='r', kde=False, bins=np.arange(xmin, xmax, width, dtype=float))
    sns.distplot(not_churn.toPandas(), color='g', kde=False, bins=np.arange(xmin, xmax, width))
    plt.legend(['churn', 'not churn'])
    plt.title('Overlaid histogram for {}'.format(i))
    plt.show()
```

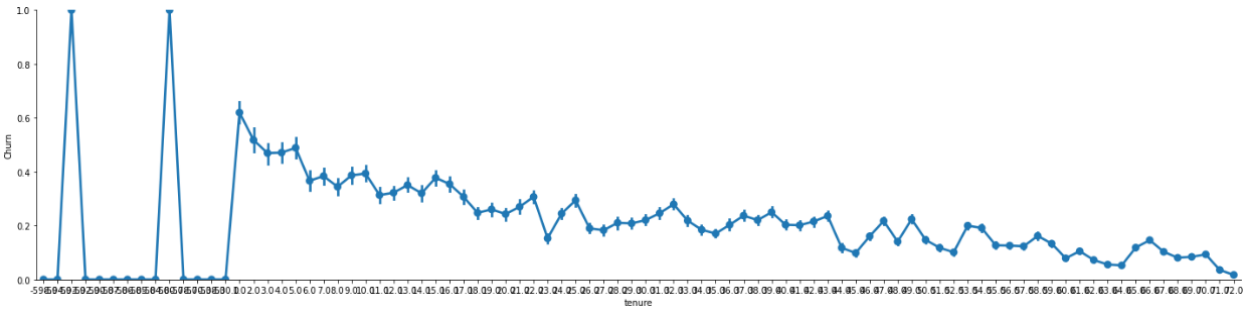




ویژگی‌های tenure و seniorCitizen در دسته‌ی ویژگی‌های عددی ناپیوسته هستند. مقادیر ویژگی seniorCitizen دارای دو مقدار صفر و یک می‌باشد که مشخص می‌کند مشتری مورد نظر یک شهروند درجه یک است یا خیر. با این وجود تعدادی داده‌ی پرت در این ستون وجود دارد. با اجرای دستور distinct() می‌توان مشاهده نمود که مقدار این ستون برای برخی از داده‌ها برابر ۱۷ یا ۱۴ است که صحیح نیست. بنابراین، فعلا از این داده‌ها صرف‌نظر می‌شود. نمودار catplot مربوط به این ویژگی به‌صورت زیر است. با توجه به این نمودار بیش از بیست درصد شهروندان درجه یک و ده درصد شهروندان غیر درجه یک، شرکت را ترک کرده‌اند.



همچنین این نمودار برای ویژگی عددی غیرپیوسته‌ی دیگر یعنی tenure نیز ترسیم شده است. این ویژگی نشان‌دهنده‌ی تعداد ماه‌هایی است که مشتری با شرکت مانده است. با توجه به این نمودار به‌طور کلی هر چه مدت ماندن مشتری‌ها با شرکت بیشتر باشد، خروج آن‌ها از شرکت نیز کمتر است.

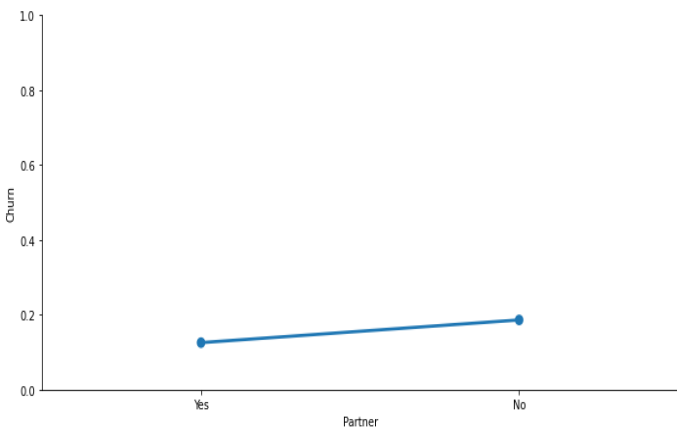


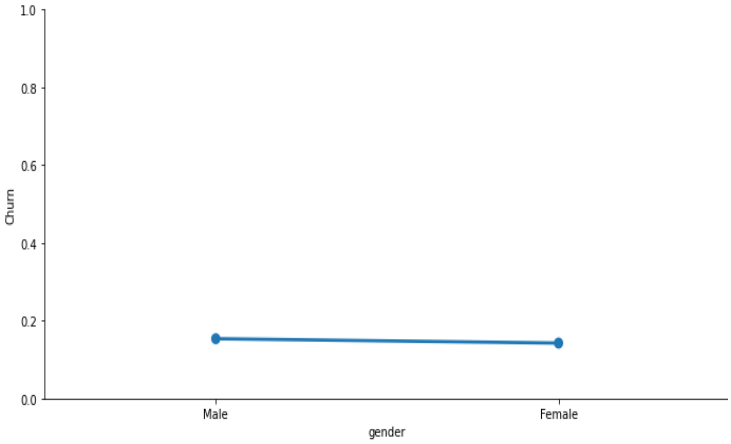
در ادامه به بررسی ویژگی‌های categorical پرداخته می‌شود. با استفاده از کد قابل مشاهده در شکل زیر می‌توان تعداد مقادیر متمایز برای هر ستون را به‌دست آورد.

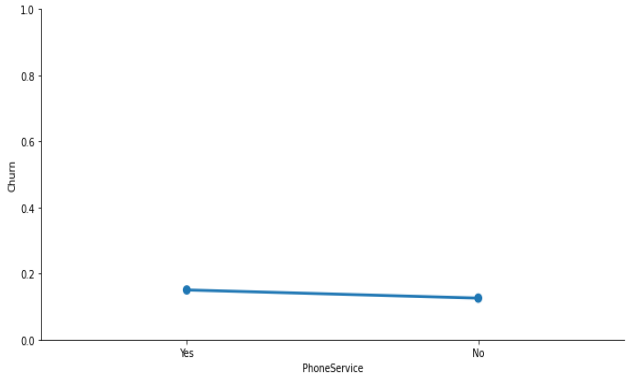
```
for col in categorical_columns:
  count = categorical_df.select(col).distinct().count()
  print(col + ': '+ str(count)+ ' unique values')
```

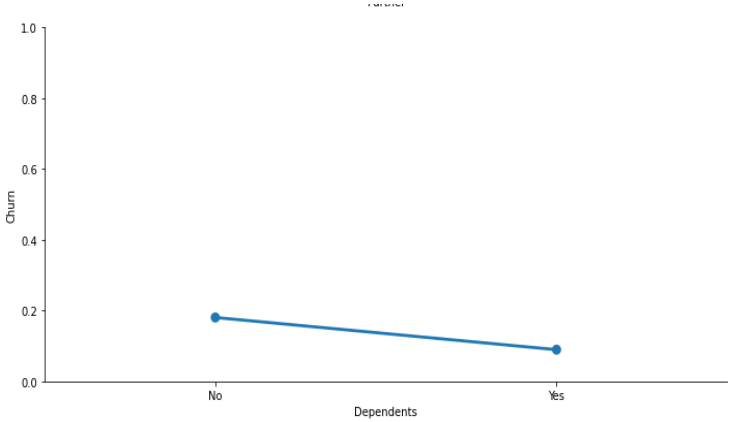
```
customerID: 7066 unique values
gender: 3 unique values
Partner: 3 unique values
Dependents: 3 unique values
PhoneService: 3 unique values
MultipleLines: 4 unique values
InternetService: 4 unique values
OnlineSecurity: 4 unique values
OnlineBackup: 4 unique values
DeviceProtection: 4 unique values
TechSupport: 4 unique values
StreamingTV: 4 unique values
StreamingMovies: 4 unique values
Contract: 4 unique values
PaperlessBilling: 3 unique values
PaymentMethod: 5 unique values
Label: 3 unique values
```

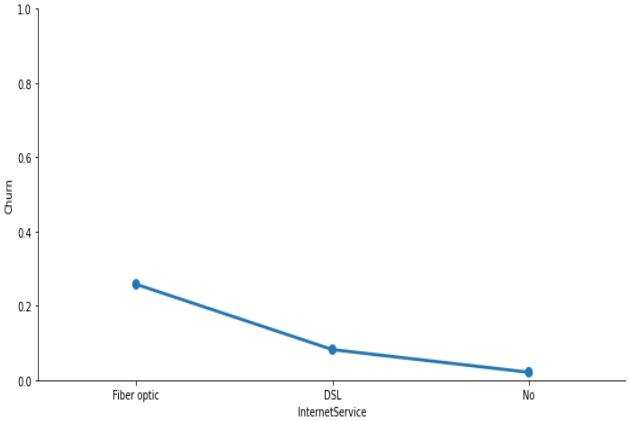
نمودارهای زیر نشان‌دهنده‌ی میزان ترک شرکت به‌ازای مقادیر مختلف ویژگی‌های categorical است. با توجه به این نمودارها، ویژگی‌هایی مانند جنسیت مشتری تاثیر چندانی بر ترک یا عدم ترک شرکت ندارد. هم‌چنین نمودار مربوط به ویژگی‌های streamingTV و streamingMovies بسیار شبیه یکدیگر است، بنابراین می‌توان یکی از این ویژگی‌ها را حذف نمود.

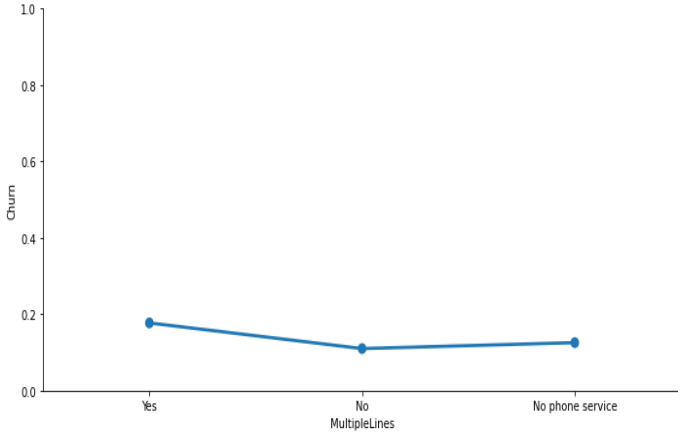


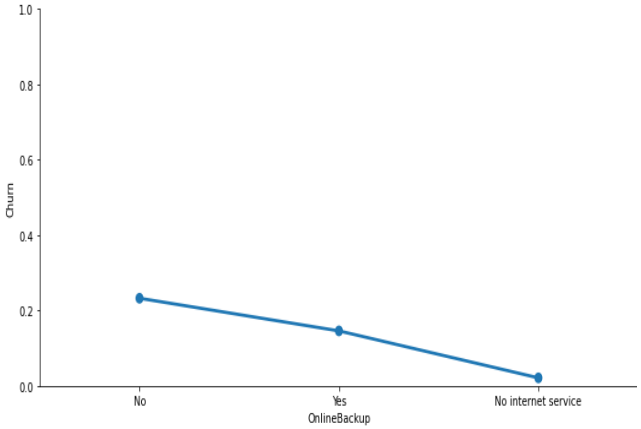


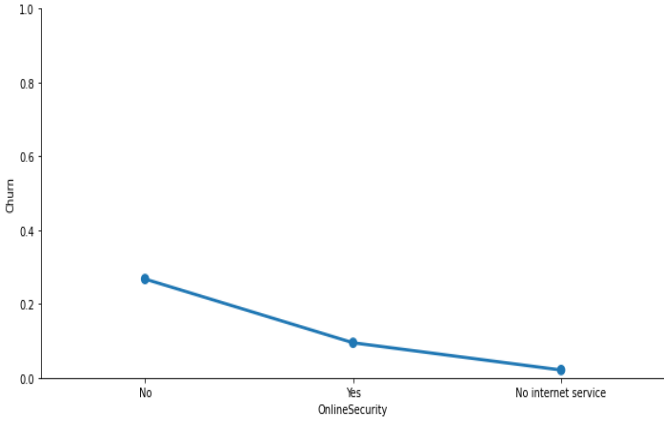

Churn
1.0
0.8
0.6
0.4
0.2
0.0
Yes
No
PhoneService

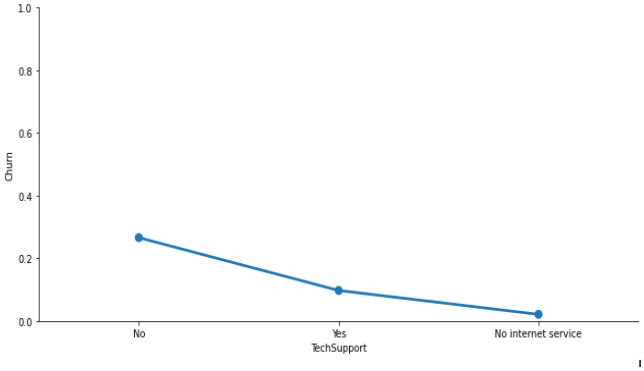

Churn
1.0
0.8
0.6
0.4
0.2
0.0
No
Yes
Dependents

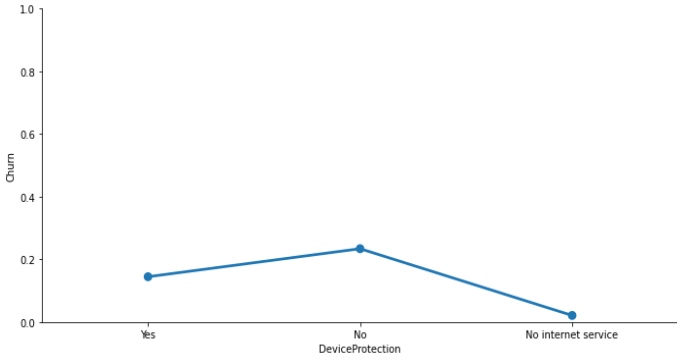

Churn
1.0
0.8
0.6
0.4
0.2
0.0
Fiber optic
DSL
No
InternetService

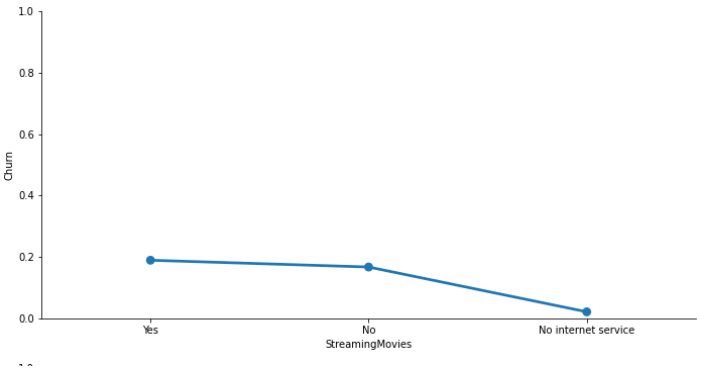

Churn
1.0
0.8
0.6
0.4
0.2
0.0
Yes
No
No phone service
MultipleLines

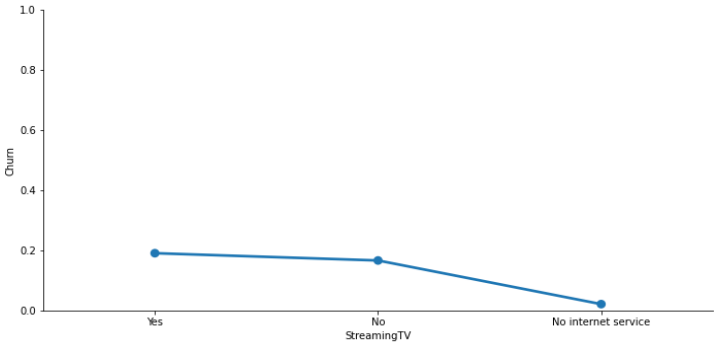

Churn
1.0
0.8
0.6
0.4
0.2
0.0
No
Yes
No internet service
OnlineBackup

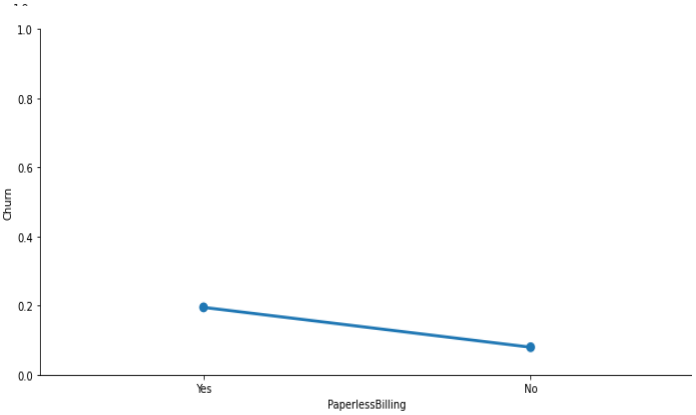

Churn
1.0
0.8
0.6
0.4
0.2
0.0
No
Yes
No internet service
OnlineSecurity

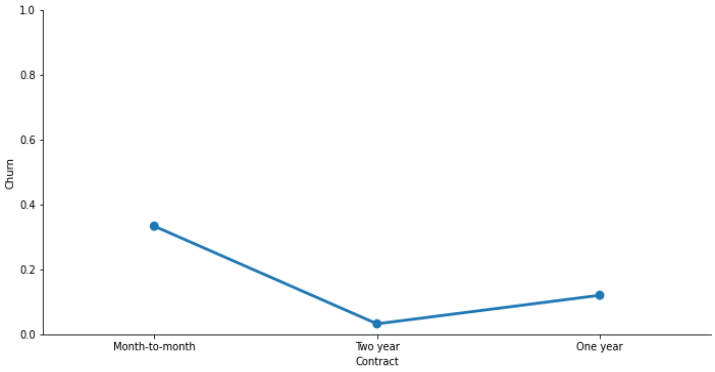

Churn
1.0
0.8
0.6
0.4
0.2
0.0
No
Yes
No internet service
TechSupport

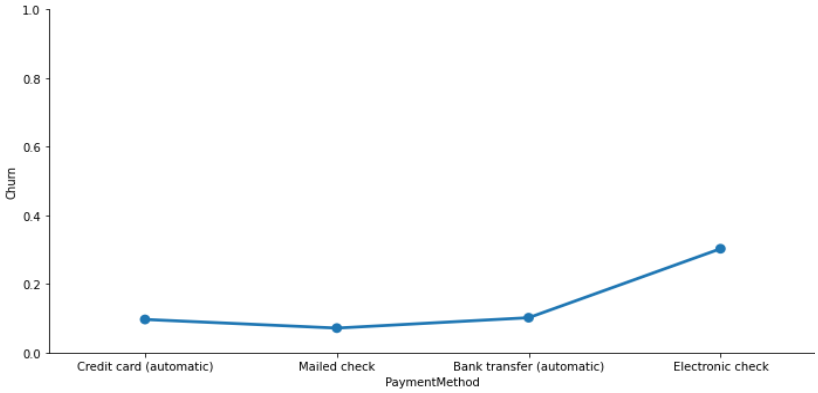

Churn
1.0
0.8
0.6
0.4
0.2
0.0
Yes
No
No internet service
DeviceProtection











داده‌های missing value

با استفاده از دستور زیر می‌توان از تعداد مقادیر missing value در هر ستون باخبر شد. با توجه به این‌که تعداد مقادیر missing value در این مجموعه داده زیاد است، حذف تمام مقادیر missing value سیاست مناسبی نمی‌باشد.

```
for column in data.columns:
  missing_count = data.select([count(when(isnan(column) | col(column).isNull() , True))]).collect()[0][0]
  print(column + ': '+ str(missing_count))
```

```
customerID: 253
gender: 235
SeniorCitizen: 266
Partner: 225
Dependents: 242
tenure: 225
PhoneService: 269
MultipleLines: 263
InternetService: 230
OnlineSecurity: 230
OnlineBackup: 243
DeviceProtection: 254
TechSupport: 264
StreamingTV: 249
StreamingMovies: 219
Contract: 230
PaperlessBilling: 257
PaymentMethod: 246
MonthlyCharges: 243
TotalCharges: 251
Label: 208
```

برخی از داده‌های missing value را می‌توان به‌صورت زیر مقداردهی کرد:

- در صورتی که مقدار ستون phoneService برای فردی «No» باشد، احتمالا مقدار ستون MultipleLines برای این فرد برابر «No phone service» است زیرا شخصی که از هیچ نوع خدمات تلفن استفاده نمی‌کند، چندین خط تلفن نیز ندارد. بنابراین در مقادیر missing value ستون MultipleLines اگر مقدار ویژگی phoneService برابر «No» باشد، می‌توان مقدار را با «No phone service» جایگزین کرد. به همین منظور از کد زیر استفاده شده است. پس از اجرای این دستور، تعداد مقادیر missing value در ستون MultipleLines از ۲۶۳ به ۲۱۹ کاهش می‌یابد.

```
import pyspark.sql.functions as F

data = data.withColumn(
    'MultipleLines',
    F.coalesce(
        F.col('MultipleLines'),
        F.when(F.col('PhoneService') == 'No', 'No phone service')
    )
)

missing_count = data.select([count(when(col('MultipleLines').isNull() , True))]).collect()[0][0]
print(missing_count)
```

```
219
```

برعکس چنین عملیاتی نیز امکان‌پذیر است. یعنی در صورتی که مقدار ستون MultipleLines برابر « No phone service» باشد، مقدار ستون phoneService باید «No» باشد.

```
data = data.withColumn(
    'PhoneService',
    F.coalesce(
        F.col('PhoneService'),
        F.when(F.col('MultipleLines') == 'No phone service', 'No')
    )
)

missing_count = data.select([count(when(col('PhoneService').isNull() , True))]).collect()[0][0]
print(missing_count)
```

```
225
```

- در صورتی که مقدار ستون InternetService برای فردی برابر «No» باشد، مقدار ستون‌های OnlineSecurity، OnlineBackup، DeviceProtection، TechSupport، StreamingTV و StreamingMovies برابر مقدار «No internet service» خواهد بود. بنابراین، می‌توان با توجه به این نکته برخی از مقادیر missing value در این ستون‌ها را مقداردهی نمود.

```
cols = ['DeviceProtection', 'OnlineBackup', 'TechSupport', 'OnlineSecurity', 'StreamingTV', 'StreamingMovies' ]

for c in cols:
  data = data.withColumn(
      c,
      F.coalesce(
          F.col(c),
          F.when(F.col('InternetService') == 'No', 'No internet service')
      )
  )

for c in cols:
  missing_count = data.select([count(when(col(c).isNull() , True))]).collect()[0][0]
  print('new missing values count of '+ c + 'is: '+ str(missing_count))
```

```
new missing values count of DeviceProtectionis: 203
new missing values count of OnlineBackupis: 194
new missing values count of TechSupportis: 207
new missing values count of OnlineSecurityis: 182
new missing values count of StreamingTVis: 202
new missing values count of StreamingMoviesis: 168
```

هم‌چنین می‌توان برخی از مقادیر missing value را به‌روش فوق برای ویژگی InternetService نیز مقداردهی نمود. اگر مقدار هر یک از ستون‌های OnlineSecurity ، OnlineBackup، DeviceProtection، TechSupport، StreamingTV و StreamingMovies برابر «No internet service» باشد، مقدار ویژگی InternetService برای آن داده باید برابر «No» باشد.

```
cols = ['DeviceProtection', 'OnlineBackup', 'TechSupport', 'OnlineSecurity', 'StreamingTV', 'StreamingMovies' ]


data = data.withColumn(
    'InternetService',
    F.coalesce(
        F.col('InternetService'),
        F.when((F.col('DeviceProtection') == 'No internet service')|
               (F.col('OnlineBackup') == 'No internet service')|
               (F.col('TechSupport') == 'No internet service')|
               (F.col('OnlineSecurity') == 'No internet service')|
               (F.col('StreamingTV') == 'No internet service')|
               (F.col('StreamingMovies') == 'No internet service')
               , 'No')
    )
)

missing_count = data.select([count(when(col('InternetService').isNull() , True))]).collect()[0][0]
print('new missing values count of InternetService: '+ str(missing_count))
```

```
new missing values count of InternetService: 159
```

- می‌توان برخی از مقادیر missing value در ستون TotalCharges را با استفاده از مقادیر ستون‌های MonthlyCharges و Tenure مقداردهی کرد. مقدار ستون TotalCharges برای هر سطر تقریبا مشابه مقدار به‌دست‌آمده از ضرب ستون MonthlyCharges و Tenure یعنی مدت قرارداد مشتری با شرکت و هزینه ماهانه است.

```
from pyspark.sql.functions import coalesce
data = data.withColumn("TotalCharges",coalesce(data.TotalCharges, (data.tenure * data.MonthlyCharges)))

missing_count = data.select([count(when(col('TotalCharges').isNull() , True))]).collect()[0][0]
print('new missing values count of TotalCharges: '+ str(missing_count))
```

```
new missing values count of TotalCharges: 122
```

- به‌طریق مشابه می‌توان برخی از مقادیر null در ستون tenure را با تقسیم مقدار متناظر در ستون TotalCharges بر مقدار موجود در ستون MonthyCharges و سپس استفاده از تابع ceil به‌دست آورد.

```
from pyspark.sql.functions import ceil

data = data.withColumn("tenure",coalesce(data.tenure, (ceil(data.TotalCharges / data.MonthlyCharges))))

missing_count = data.select([count(when(F.col('tenure').isNull() , True))]).collect()[0][0]
print('new missing values count of tenure: '+ str(missing_count))
```

```
[162] data = data.withColumn("MonthlyCharges",coalesce(data.MonthlyCharges, (ceil(data.TotalCharges / data.tenure))))

      missing_count = data.select([count(when(F.col('MonthlyCharges').isNull() , True))]).collect()[0][0]
      print('new missing values count of MonthlyCharges: '+ str(missing_count))

      new missing values count of MonthlyCharges: 47
```

- تعداد ۲۰۸ داده در این مجموعه فاقد برچسب نهایی (Label) هستند، تعداد این داده‌ها در مقایسه با حجم کل دیتاست ناچیز است (حدود یک درصد از کل مجموعه داده). به‌علاوه برای تعداد زیادی از داده‌های بدون برچسب، بخشی از سایر ویژگی‌ها نیز درج نشده است. بنابراین می‌توان این رکوردها را از مجموعه حذف نمود.

```
[148] data=data.where(data['Label'].isNull()== False)
```

- پس از انجام عملیات مذکور، داده‌های null باقی‌مانده در ستون‌های categorical و ویژگی‌های tenure و SeniorCitizen با پرتکرارترین عنصر در آن ستون (mode) جایگزین شده‌اند.

```
columns = categorical_columns
columns.remove('customerID')
columns.remove('Label')
columns.append('tenure')
columns.append('SeniorCitizen')

for column in categorical_columns:
  grouped = data.groupBy(column).count()
  grouped = grouped.withColumn("count", grouped["count"]).orderBy('count', ascending=False)
  mode = grouped.first()[0]
  data = data.na.fill(value=mode,subset=[column])
```

- در ادامه باقی‌مانده‌ی مقادیر null در ویژگی montlyCharges با میانگین اعداد این ستون جایگزین می‌شود. پس از اجرای این دستور تمام مقادیر null در تمام ستون‌ها به جز TotalCharges مقداردهی می‌شوند. سپس مقادیر null باقی‌مانده در ستون TotalCharges با ضرب مقادیر متناظر در ستون‌های tenure و montlyCharges مقداردهی می‌شود.

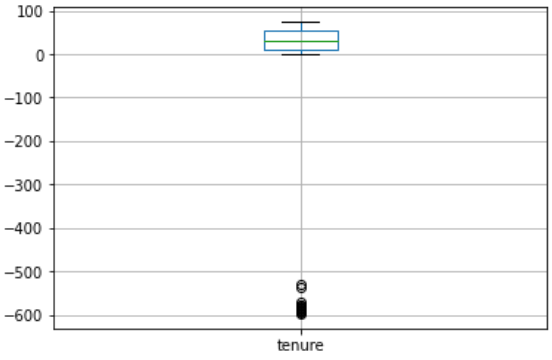
```
replacing with mean

df_mean = data.agg({'MonthlyCharges': 'avg'}).collect()
mean = df_mean[0][0]
data = data.na.fill(value=mean,subset=['MonthlyCharges'])
```

- در مجموعه داده‌ی مورد بررسی تعداد زیادی سطر تکراری وجود دارد. می‌توان با استفاده از دستور زیر داده‌های تکراری را حذف نمود.

```
data = data.distinct()
```

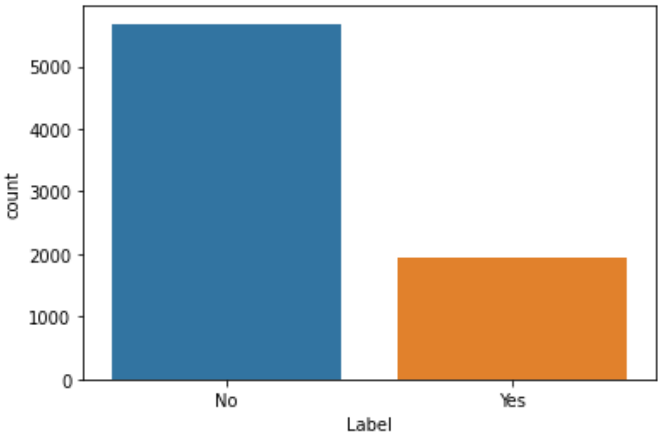
- با رسم نمودار boxplot برای ویژگی‌ها می‌توان به وجود یا عدم وجود داده‌های پرت در مجموعه پی برد. دو ویژگی seniorCitizen و tenure حاوی داده‌های پرت هستند. تعداد محدودی از داده‌های ستون tenure دارای مقدار ۱۴ یا ۱۷ می‌باشند. هم‌چنین برخی از داده‌ها در ستون tenure دارای مقدار منفی هستند که غیرمنطقی می‌باشد. با استفاده از دستور زیر می‌توان این داده‌ها را حذف نمود.

```
cleaned_data=cleaned_data.where((col('SeniorCitizen') != 14.0) | (col('SeniorCitizen') != 17.0) )
cleaned_data=cleaned_data.where(cleaned_data['tenure']> 0)
```



- تعداد کلاس‌ها در مجموعه داده برابر نیست و این سبب می‌شود که مجموعه داده نامتوازن[۴] باشد و مدل در تشخیص یکی از کلاس‌ها خوب عمل نکند. با استفاده از روش oversampling میتوان تا حدی این مشکل را برطرف نمود.

[4] Imbalanced




```
] from pyspark.sql.functions import col, explode, array, lit

  major_df = cleaned_data.filter(col("label") == 'No')
  minor_df = cleaned_data.filter(col("label") == 'Yes')
  ratio = int(major_df.count()/minor_df.count())
  print("ratio: {}".format(ratio))
  a = range(ratio)

  oversampled_df = minor_df.withColumn("dummy", explode(array([lit(x) for x in a]))).drop('dummy')

  combined_df = major_df.unionAll(oversampled_df)

  cleaned_data = combined_df
```

- **قدم دوم:** عملیات feature engineering را به خوبی برای دادگان خود انجام دهید و دلیل انتخاب هریک از ستون‌ها یا عدم انتخاب آن‌ها را به صورت منطقی بیان کنید. (با نمودار و تحلیل آن، با کمک EDA انجام شده)

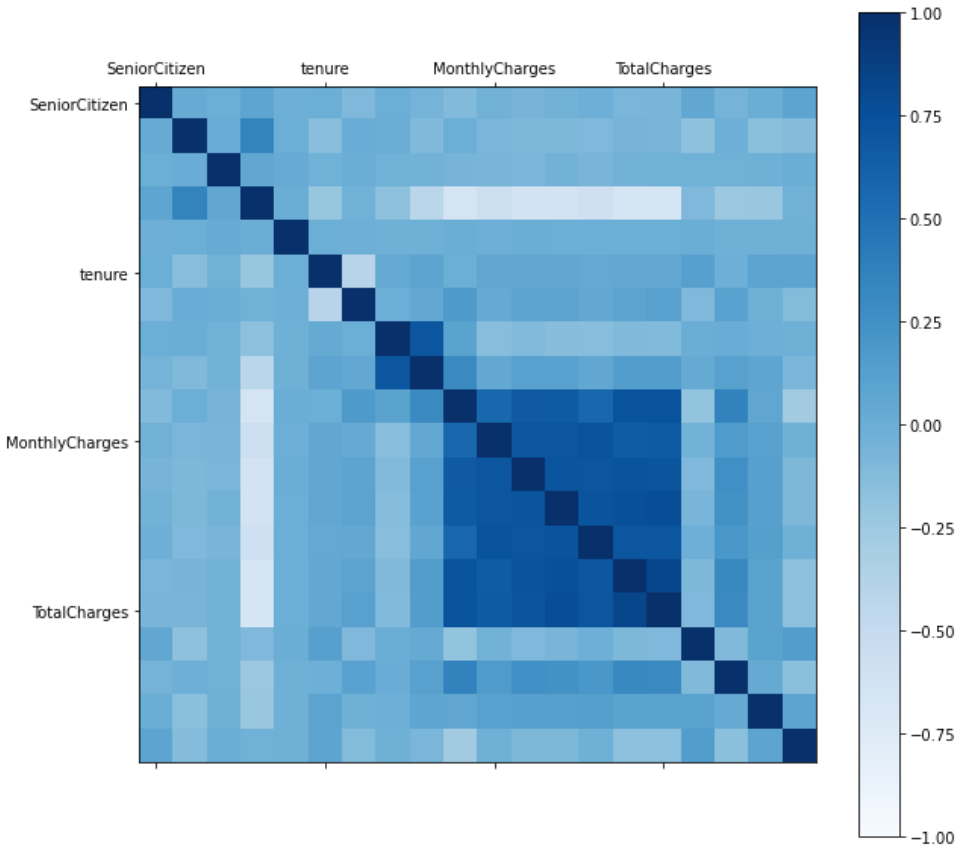
- ستون مربوط به ویژگی customerID شناسه‌ای است که به مشتریان تعلق می‌گیرد و حاوی اطلاعات سودمندی درباره ترک یا عدم ترک شرکت توسط مشتری نیست. بنابراین، می‌توان این ستون را در گام اول حذف نمود.

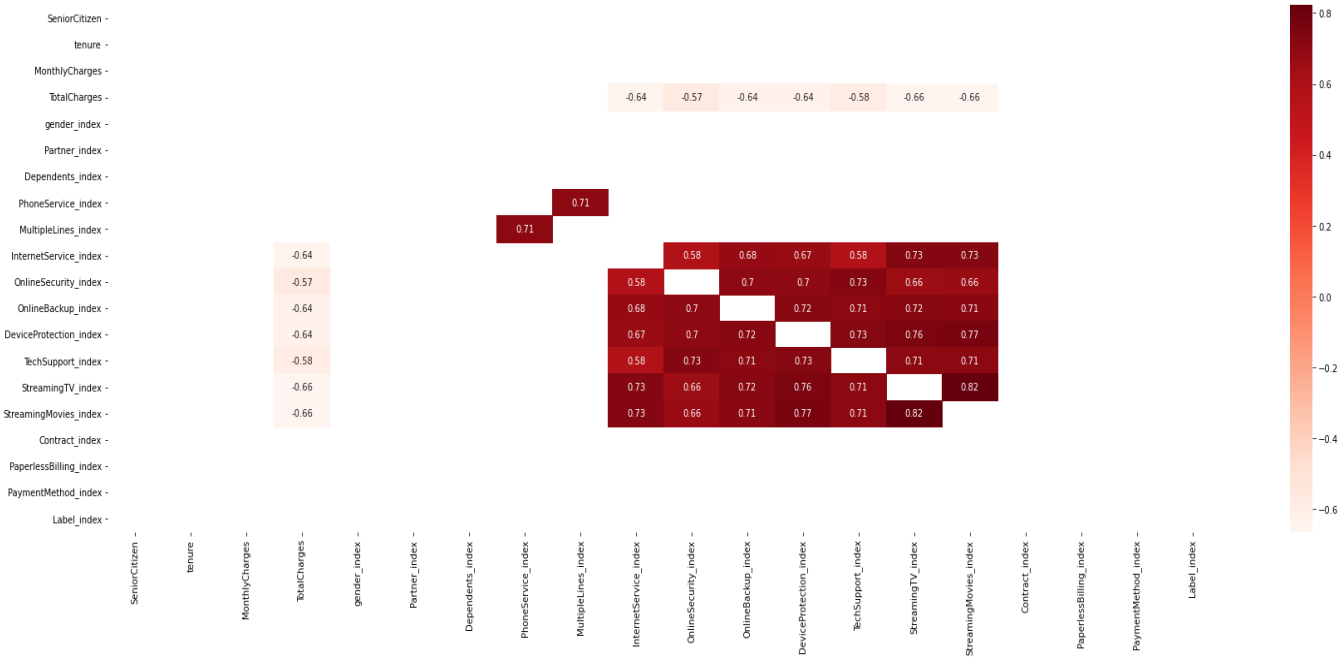
- ستون gender نیز رابطه‌ی زیادی با ستون برچسب ندارد. مقدار ضریب همبستگی پیرسون بین این ویژگی و برچسب برابر ۰.۰۰۷ می‌باشد. هم‌چنین با حذف این ویژگی دقت مدل افزایش می‌یابد. لذا این ستون نیز حذف شده است.

- با تبدیل ویژگی‌های categorical به مقادیر عددی می‌توان همبستگی موجود میان تمام ویژگی‌های مجموعه داده را مشاهده نمود. با استفاده از ماژول stringIndexer می‌توان مقادیر categorical موجود در دیتافریم اسپارک را به مقادیر عددی تبدیل نمود.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer

indexers = [StringIndexer(inputCol=column, outputCol=column+"_index").fit(data) for column in categorical_columns ]

pipeline = Pipeline(stages=indexers)
df_r = pipeline.fit(data).transform(data)
```

سپس می‌توان نمودار همبستگی را برای تمام ویژگی‌های موجود در دیتاست به‌دست آورد.





با توجه به مقادیر به‌دست‌آمده، ویژگی‌های StreamingTV و StremingMovies بیشترین میزان همبستگی را دارند. مقدار ضریب همبستگی برای این دو ویژگی برابر ۰.۸۲ می‌باشد که به یک خیلی نزیک است. هم‌چنین ویژگی StreamingTV با پنج ویژگی دیگر همبستگی بالای ۰.۵ دارد. بنابراین حذف ویژگی StreamingTV احتمالا سودمند خواهد بود.

- **افزودن ویژگی جدید**

ستون customerID از دو بخش عددی و رشته‌ای تشکیل شده است. تعداد زیادی از داده‌ها در بخش دوم این ستون که به‌صورت یک رشته می‌باشد، مشترک هستند. بنابراین می‌توان با استفاده از یک تابع udf این بخش را از شناسه‌ی کاربران جدا کرده و در یک ستون جدید قرار داد.

```
from pyspark.sql.functions import udf

@udf
def extractor(id):
  if id is None:
      return None
  else:
      return id.split('-')[1]

data = data.withColumn("customerTitle", extractor('customerID'))
```

- **قدم سوم:** الگوریتم Logestic Regression را بر روی داده‌های خود اعمال کنید.

به‌منظور اعمال Logestic Regression بر روی داده‌ها، ابتدا با استفاده از StringIndexer ویژگی‌های categorical به عدد تبدیل می‌شوند. سپس این مقادیر با استفاده از OneHotEncoder کدگذاری می‌شوند. سپس ویژگی برچسب نیز با استفاده از StringIndexer به مقادیر عددی تبدیل می‌شود. سپس ویژگی‌ها با استفاده از VectorAssembler به بردار تبدیل می‌شوند. به‌منظور نرمال‌سازی مقادیر می‌توان از StandardScaler استفاده نمود. برای استفاده‌ی بهتر از چندین transformer و estimator و مشخص کردن جریان کار، می‌توان از pipeline استفاده نمود. برای این کار لازم است transformerها به متغیر stages که ورودی مدل pipeline است اضافه شوند. در پایان می‌توان با استفاده از کتابخانه‌های pyspark.ml.classification LogisticRegression. مدل را آموزش داده و سپس با کتابخانه BinaryClassificationEvaluator آن را ارزیابی نمود.

```
from pyspark.ml.feature import (VectorAssembler,VectorIndexer,
                                OneHotEncoder,StringIndexer, StandardScaler)
from pyspark.ml import Pipeline

stages = []

numeric_features = ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']
categorical_columns =  [item[0] for item in cleaned_data.dtypes if item[1].startswith('string')]
categorical_columns.remove('Label')

for feature in categorical_columns:
  stringIndexer = StringIndexer(inputCol=feature, outputCol=feature + 'Index')
  encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()],outputCols=[feature + 'Vec'])
  stages+=[stringIndexer, encoder]

labelIndexer = StringIndexer(inputCol='Label', outputCol='label')
stages += [labelIndexer]
input_features = [c + 'Vec' for c in categorical_columns]+ numeric_features
assembler = VectorAssembler(inputCols= input_features, outputCol='features')
stages+=[assembler]
scaler = StandardScaler(inputCol='features', outputCol='standard_features')
stages+=[scaler]

pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(cleaned_data)
assembler_df = pipelineModel.transform(cleaned_data)
```

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator

log_reg= LogisticRegression(featuresCol='features',labelCol='label')

train_data, test_data = assembler_df.randomSplit([0.8, 0.2])
fit_model = log_reg.fit(train_data)
results = fit_model.transform(test_data)

my_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction',
                                        labelCol='label')
results.select('label','prediction')
AUC = my_eval.evaluate(results)
print("AUC score is : ",AUC)

accuracy = results.filter(results.label == results.prediction).count() / float(results.count())
print("Accuracy : ",accuracy)
```

```
AUC score is :  0.9936902154390099
Accuracy :  0.9981020528371037
```

- **قدم چهارم:** دقت مدل خود را ارزیابی کنید. (در این مرحله شما باید مراحل آزمایش، تعداد دادگان ترین و تست، احتمال صحیح بودن یک برچسب که مدل پیشبینی کرده است، را تعیین کنید)
- نتایج مدل قبل و بعد از پیش‌پردازش را مقایسه کنید.

در صورتی‌که عناصر تکراری از مجموعه داده‌ها حذف نشود، یعنی داده‌ها در حدود ۲۲۹۷۸۲ باشد، دقت دو مدل تقریبا برابر خواهد بود. بیست درصد داده‌ها به تست اختصاص می‌یابد. شکل زیر دقت مدل در صورت عدم پردازش داده‌ها می‌باشد.

```
AUC score is :  0.9952824572949717
Accuracy :  0.998455850369726
```

در صورتی‌که داده‌های تکراری از مجموعه داده حذف شود، اندازه‌ی دیتاست در حالت بدون پردازش ۷۶۳۰ و در حالت با پردازش ۹۵۶۲ است (زیرا از روش oversampling برای متوازن‌سازی مجموعه استفاده شده است). دقت مدلی که با داده‌های پردازش‌شده آموزش یافته است در حدود ۸۴ درصد به‌دست می‌آید و دقت مدلی که با داده‌های بدون پردازش آموزش یافته است به ۷۶ درصد کاهش می‌یابد (۸۰ درصد داده‌ها به آموزش و ۲۰ درصد به ارزیابی اختصاص یافته است). شکل‌های زیر به‌ترتیب نشان‌دهنده‌ی دقت مدل در حالت با پردازش و بدون پردازش است.

```
AUC score is :  0.863814154960147
Accuracy :  0.8458289334741288
```

```
AUC score is :  0.6752841950861752
Accuracy :  0.7641509433962265
```

به‌عنوان معیارهای دیگر برای ارزیابی مدل می‌توان به precision، recall و F1 اشاره کرد. شکل زیر نشان‌دهنده‌ی مقادیر به‌دست‌آمده برای این معیارها در حالت با پردازش است.

```
Precision : 0.7335423197492164
Recall: : 0.9273447820343461
F1: 0.8191365227537923
```