



하반기 팀 프로젝트

순천향대학교 컴퓨터공학과
20184102 신주용

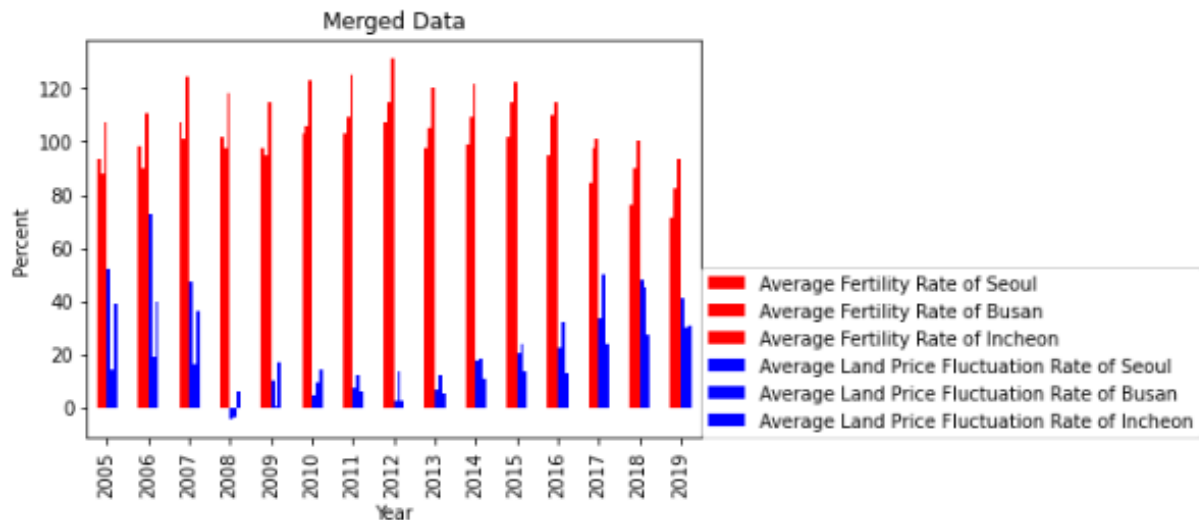


주택가 변동률과 출산율의 상관관계

- 지난번 사용했던 데이터는 자가변동률 데이터
 - 실제로 분석시에는 거의 동일한 결과
- 해당 데이터와 주택 가격, 주택가 변동률 등의 정보가 담긴 데이터를 추가
- 카프카를 사용한 스파크 스트리밍을 구현
- 로지스틱 회귀 알고리즘을 사용하여 머신러닝 처리를 구현

주택가 변동률과 출산율의 상관관계

이전에 진행했던 프로젝트를 요약한 그래프입니다.
지가변동률과 출산율에 대한 전국의 데이터를 가져와 1년 단위로 평균을 도출하여
그래프에 모두 합병한 모습입니다.
이번 프로젝트에서는 지난번의 추론을 증명하는 방식으로 설명하도록 하겠습니다.



데이터 스트리밍



데이터 스트리밍

데이터 스트리밍 처리를 위해 카프카를 사용하였습니다.



주키퍼-카프카, 주피터, 스파크를 종속성 설정한 후에
스파크 패키지로써 연동해줍니다.

마스터 노드에서 프로듀서가 토픽을 생성하고
브로커 포트로 파일 혹은 메시지를 전송하면
스파크에서 생성한 컨슈머가 메시지를 받아들이는 형식으로
사용하였습니다.

데이터 스트리밍

아래와 같이 마스터 노드에서 브로커 포트를 통해 전송합니다.

```
/kafka-console-producer.sh --broker-list localhost:92 --topic topic < /home/bigdata/sparkdata/TP/seoul_pol.csv
```

파일을 전송하여 수신하는 방식 / 텍스트로 전송하고 수신하는 방식
모두 적용 가능합니다.

```
... 25 more
[I 11:08:35.898 NotebookApp] Saving file at /TP-1-Copy1.ipynb
[I 11:10:35.907 NotebookApp] Saving file at /TP-1-Copy1.ipynb      (0 + 1) / 1
[Stage 1:>]                                                         (0 + 1) / 1
```

스파크에서 컨슈머가 작동중일 때 프로듀서에서 메시지가 도착하면
위처럼 콘솔에 스테이지가 나타나고
우측과 같이 메시지가 전달됩니다.

```
##### 2015,0.183,0.162,0.205,0.223,0.284,0.254,0.277,0.255,0.222,0.234,0.24,0.203
##### 2014,0.2,0.127,0.401,0.326,0.242,0.198,0.202,0.107,0.201,0.155,0.123,0.104
##### 2014,0.064,0.313,0.105,0.365,0.238,0.175,0.242,0.183,0.358,0.203,0.139,0.149
##### 2014,0.068,0.018,0.118,0.121,0.11,0.173,0.224,0.082,0.201,0.124,0.125,0.116
##### 2014,0.091,0.12,0.42,0.248,0.126,0.085,0.095,0.085,0.171,0.403,-0.084,0.075
##### 2014,0.084,0.141,0.096,0.107,0.096,0.197,0.222,0.194,0.197,0.097,0.197,0.179
##### 2014,0.103,0.128,0.099,0.152,0.103,0.126,0.113,0.076,0.144,0.08,0.126,0.103
##### 2014,0.402,0.135,0.108,0.105,0.094,0.178,0.142,0.103,0.114,0.148,0.15,0.173
##### 2014,0.031,0.172,0.215,0.28,0.1,0.016,0.087,0.046,0.184,0.14,0.025,0.121
##### 2014,0.148,0.14,0.342,0.128,0.08,0.09,0.099,0.212,0.191,0.099,0.185,0.157
##### 2014,0.401,0.144,0.066,0.095,0.105,0.05,0.098,0.075,0.155,0.186,0.113,0.158
##### 2014,0.313,0.077,0.121,0.149,0.083,0.038,0.108,0.108,0.126,0.078,0.11,0.107
##### 2014,0.206,0.073,0.211,0.228,0.188,0.241,0.246,0.192,0.253,0.263,0.188,0.198
##### 2014,0.06,0.183,0.167,0.172,0.111,0.158,0.223,0.216,0.234,0.285,0.221,0.125
##### 2014,0.081,0.298,0.234,0.195,0.131,0.21,0.218,0.27,0.292,0.388,0.263,0.222
##### 2014,0.266,0.209,0.177,0.162,0.183,0.228,0.138,0.046,0.214,0.29,0.092,0.281
##### 2014,0.344,0.246,0.188,0.179,0.215,0.106,0.117,0.071,0.264,0.255,0.171,0.047
##### 2014,0.218,0.108,0.327,-0.057,0.17,0.156,0.211,0.085,0.096,0.208,0.14,0.184
##### 2014,0.079,0.119,0.238,0.092,0.267,0.094,0.26,0.127,0.149,0.143,0.157,0.118
##### 2014,0.058,0.102,0.227,0.205,0.094,0.205,0.173,0.156,0.271,0.078,0.086,0.168
##### 2014,0.065,0.125,0.4,0.166,0.152,0.119,0.111,0.118,0.1,0.112,0.226,0.145
##### 2014,0.053,0.147,0.233,0.204,0.18,0.089,0.129,0.105,0.106,0.112,0.169,0.137
##### 2014,0.401,0.267,0.413,0.192,0.327,0.275,0.296,0.29,0.403,0.411,0.226,0.217
##### 2014,0.211,0.3,0.51,0.311,0.211,0.283,0.276,0.283,0.534,0.536,0.401,0.281
##### 2014,0.217,0.331,0.41,0.437,0.205,0.152,0.256,0.305,0.411,0.468,0.22,0.174
##### 2014,0.158,0.194,0.12,0.187,0.144,0.142,0.2,0.197,0.23,0.214,0.213,0.174
##### 2013,-0.009,0.054,0.069,0.154,0.138,0.068,0.041,0.025,0.102,0.086,0.202,0.265
##### 2013,-0.033,0.049,0.095,0.127,0.151,-0.003,-0.02,-0.073,0.069,0.112,0.11,0.019
##### 2013,-0.023,-0.013,-0.035,-0.626,-0.495,-0.238,-0.195,-0.228,-0.128,-0.1,-0.044,-0.007
##### 2013,-0.117,0.053,0.111,-0.069,-0.058,0.069,-0.083,-0.091,0.06,0.178,0.196,0.028
##### 2013,-0.127,0.045,0.069,0.194,-0.065,0.064,0.025,-0.075,0.058,0.071,0.103,0.092
##### 2013,-0.109,0.028,0.032,0.167,-0.083,0.121,-0.071,0.067,-0.05,0.154,0.2,0.083
##### 2013,-0.138,0.028,0.07,-0.053,0.116,0.108,-0.106,0.058,-0.028,0.1,0.087,0.215
##### 2013,-0.058,0.001,0.01,0.111,0.157,0.026,-0.118,-0.128,-0.114,0.11,0.097,0.088
##### 2013,-0.138,0.028,0.06,0.13,0.093,0.003,-0.082,-0.057,0.108,0.128,0.136,0.067
##### 2013,-0.152,0.054,0.021,-0.032,0.154,0.034,-0.139,0.038,0.041,0.055,0.052,0.09
##### 2013,-0.145,0.074,0.011,0.166,0.13,-0.045,-0.066,-0.081,0.154,0.027,0.06,0.072
##### 2013,-0.112,0.069,0.062,0.133,0.177,0.127,-0.066,-0.092,0.095,0.081,0.135,0.167
##### 2013,-0.076,0.008,0.086,0.113,0.136,0.102,-0.087,-0.086,0.023,0.117,0.107,0.094
##### 2013,-0.07,0.052,0.149,0.15,0.191,0.137,-0.068,-0.052,0.048,0.1,0.107,0.213
##### 2013,-0.178,0.043,0.068,0.16,0.197,0.137,-0.068,-0.097,0.049,0.165,0.282,0.085
##### 2013,-0.181,-0.045,0.052,0.191,0.224,0.154,-0.05,-0.068,0.208,0.21,0.214,0.225
##### 2013,-0.093,-0.021,0.105,0.142,0.157,0.119,-0.068,0.008,0.058,0.136,0.069,0.038
##### 2013,-0.087,-0.02,0.07,0.213,0.19,0.115,-0.068,0.015,0.059,0.153,0.196,0.095
##### 2013,-0.074,-0.007,0.086,0.125,0.113,0.068,-0.009,-0.019,0.04,0.156,0.062,0.053
##### 2013,-0.069,0.019,0.038,0.105,0.176,0.134,-0.055,-0.037,0.043,0.146,0.055,0.1
##### 2013,-0.094,0.013,0.05,0.123,0.157,0.131,-0.063,-0.03,0.011,0.153,0.05,0.307
##### 2013,0.018,0.065,0.206,0.21,0.24,0.035,-0.07,0.023,0.223,0.317,0.257,0.278
##### 2013,0.025,0.112,0.171,0.327,0.294,0.189,-0.01,0.033,0.327,0.384,0.437,0.302
##### 2013,-0.02,0.149,0.137,0.365,0.278,0.241,0.12,0.037,0.434,0.436,0.398,0.394
##### 2013,-0.023,0.083,0.015,0.086,0.106,0.086,0.086,0.086,0.086,0.086,0.086,0.086
```

데이터 스트리밍

```
In [13]: from kafka import KafkaConsumer
         from kafka import KafkaProducer
```

```
In [14]: df = spark \
         .readStream \
         .format("kafka") \
         .option("kafka.bootstrap.servers", "localhost:9092") \
         .option("subscribe", "topic") \
         .option("checkpointLocation", "/test/checkpoint") \
         .option("failOnDataLoss", "false") \
         .load()

res = df.select('*')

df.printSchema()

root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)
```

```
query = res.writeStream.outputMode("append") \
         .format("console") \
         .option("checkpointLocation", "/test/checkpoint") \
         .option("path", '/TPDATA') \
         .start()
```

```
query.awaitTermination()
```

```
: query = res.writeStream.outputMode("append") \
         .format("csv") \
         .option("checkpointLocation", "/test/checkpoint") \
         .option("path", '/TPDATA') \
         .start()
```

```
query.awaitTermination()
```

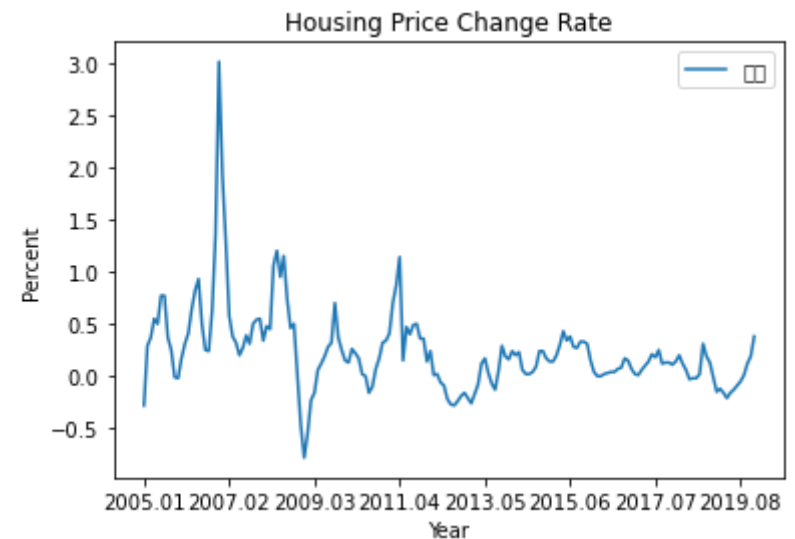
메시지를 전송 받을 컨슈머 입니다. 동일한 9092 포트를 사용하고,
readStream으로 받은 데이터를 writeStream으로 콘솔에 출력해줍니다.
메모리 옵션을 사용하면 메모리에 저장하여 사용하는 것이 가능합니다.
구현 단계에서는 자원상의 한계와 속도 문제 때문에
format에서 csv로 형식을 지정해주어 HDFS에 저장한 후 사용하였습니다.

데이터 분석

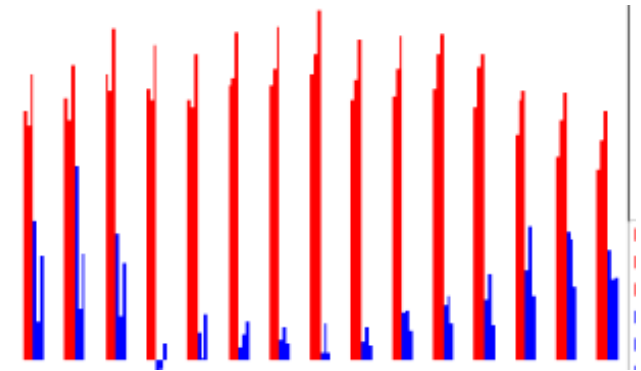


데이터 분석

```
house_schema = [StructField('시점', StringType(), True), StructField('전국', DoubleType(), True)]  
structure = StructType(fields = house_schema)  
house_data = spark.read.option("header", True).csv('/TPDATA/house.csv', encoding='euc-kr', schema = structure)  
five = house_data.select(col("시점"), col("전국"))  
fivplt = five.toPandas().plot(kind = 'line', x = '시점', y = ['전국'], xlabel = 'Year', ylabel = 'Percent')  
plt.title('Housing Price Change Rate')
```



위가 집값의 변동률에 대한 그래프,
아래의 파란 바 그래프가 지가변동률 그래프입니다.
이번에는 위의 집값 데이터를 사용하여 모든 작업을 수행하였습니다.
데이터의 소스는 한국부동산원 부동산통계정보시스템(R-ONE)과
국가통계포털 입니다.



데이터 분석

```
newSch = [StructField('Year_Month', StringType(), True), StructField('CrudeB', DoubleType(), True), StructField('HPCR', DoubleType(), True)]
structure = StructType(fields = newSch)
rate_new = spark.read.option("header", True).csv('/TPDATA/testfor.csv', encoding='euc-kr', schema = structure)
```

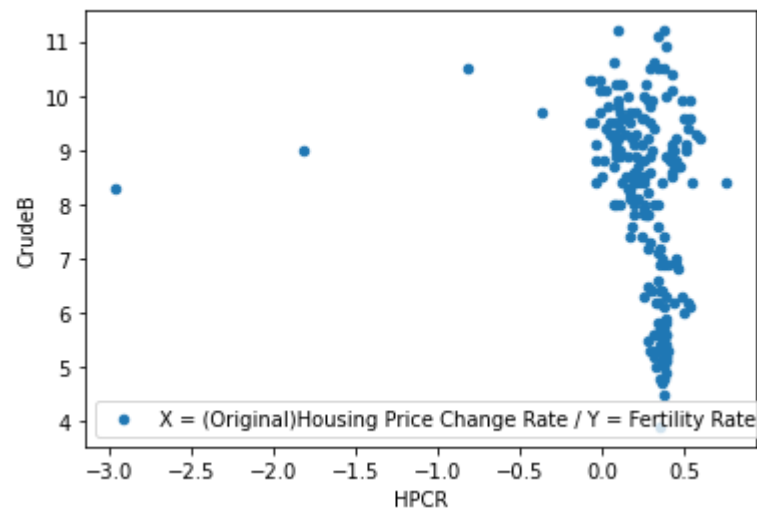
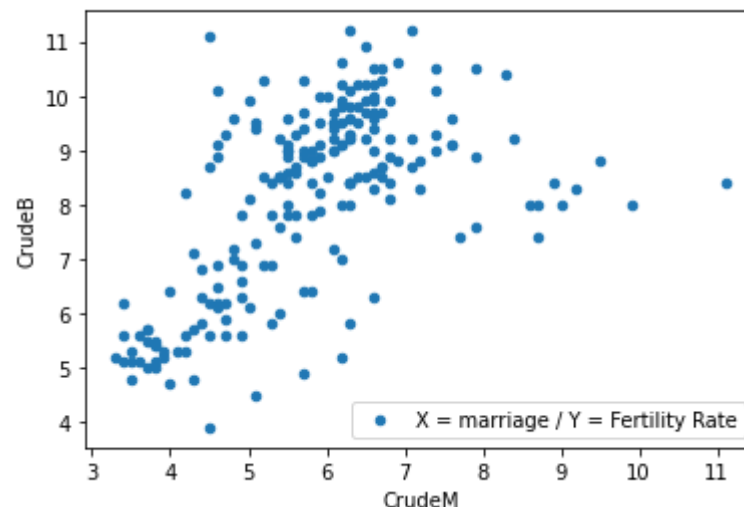
```
old_panda = rate_data.select('HPCR', 'CrudeB').toPandas()
rate_panda = rate_new.select('HPCR', 'CrudeB').toPandas()
marriage = rate_data.select('CrudeM', 'CrudeB').toPandas()

marriage.plot(kind = 'scatter', x = 'CrudeM', y = 'CrudeB', label = 'X = marriage / Y = Fertility Rate') # 산점도 추출
old_panda.plot(kind = 'scatter', x = 'HPCR', y = 'CrudeB', label = 'X = (Original)Housing Price Change Rate / Y = Fertility Rate') # 산점도 추출
rate_panda.plot(kind = 'scatter', x = 'HPCR', y = 'CrudeB', label = 'X = Housing Price Change Rate / Y = Fertility Rate') # 산점도 추출
```

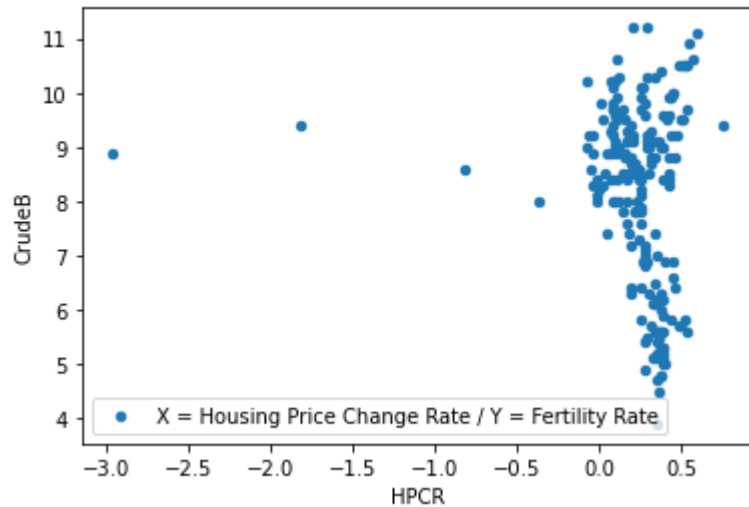
**ML처리 전에 육안으로 데이터 분포의 파악을 위해
산점도 출력을 해보았습니다.**

**우측 상단은 결혼률과 출산률의 상관관계 파악을 위해 작성하였습니다.
X가 결혼률, Y가 출산률로 우상향을 그리는 그래프임을 알 수 있습니다.**

**우측 하단은 집값변동률과 출산률에 관한 그래프입니다.
예상과는 달리 0.5 근처에 점들이 모여있는 것을 알 수 있습니다.**



데이터 분석



이는 집값의 변동과 자녀 계획 검토 사이의 시간차로,
임신 기간은 약 10개월 정도이므로
출산할 때의 주택 가격과, 출산 여부를 결정할 때의 주택 가격이
최소 1년 이상 차이나는 것을 알 수 있습니다.

왼쪽의 사진은 1년 정도의 시간차를 두고 다시 작성한 그래프입니다.
이전보다 조금 더 0.0에 가깝게 분산된 것을 알 수 있습니다.

이진 로지스틱 회귀 분석 구현



이진 로지스틱 회귀 분석

```
ML_schema = [StructField('Year_Month', StringType(), True), StructField('Birth', DoubleType(), True),
              StructField('CrudeB', DoubleType(), True), StructField('Death', DoubleType(), True),
              StructField('CrudeD', StringType(), True), StructField('Nat', DoubleType(), True),
              StructField('NatRate', StringType(), True), StructField('Marriage', DoubleType(), True),
              StructField('CrudeM', DoubleType(), True), StructField('divorce', DoubleType(), True),
              StructField('CrudeDiv', DoubleType(), True), StructField('Self', DoubleType(), True),
              StructField('HPCR', DoubleType(), True)]

structure = StructType(fields = ML_schema)
rate_data = spark.read.option("header", True).csv('/TPDATA/FR_P0.csv', encoding='euc-kr', schema = structure)

rate_data = rate_data.dropna().withColumn("CrudeB_Rate", (rate_data.CrudeB) >= 8)
rate_data = rate_data.withColumn("CrudeM_Rate", (rate_data.CrudeM) >= 5)
rate_data = rate_data.withColumn("CrudeDiv_Rate", (rate_data.CrudeDiv) >= 2.3)
rate_data = rate_data.withColumn("HPCR_Rate", (rate_data.HPCR) > 0)
rate_data = rate_data.withColumn("CrudeB_Rate", col("CrudeB_Rate").cast("string")) # 자료형 전환
rate_data = rate_data.withColumn("CrudeM_Rate", col("CrudeM_Rate").cast("string"))
rate_data = rate_data.withColumn("CrudeDiv_Rate", col("CrudeDiv_Rate").cast("string"))
rate_data = rate_data.withColumn("HPCR_Rate", col("HPCR_Rate").cast("string"))

rate_data.select('Year_Month', 'HPCR', 'CrudeB').show()
```

**스트리밍으로 받은 파일을 불러와 스키마를 지정하고
카테고리형으로 사용할 자료들을 선택해 열을 추가해줍니다.
기준점이 될 지표를 출력하면 우측과 같은 결과가 나옵니다.**

Year_Month	HPCR	CrudeB
2005.01	0.157	10.0
2005.02	0.123	9.6
2005.03	0.278	9.9
2005.04	0.455	9.2
2005.05	0.434	8.7
2005.06	0.75	8.4
2005.07	0.363	8.4
2005.08	0.464	8.7
2005.09	0.15	9.3
2005.10	0.299	9.0
2005.11	0.384	8.9
2005.12	0.312	8.0
2006.01	0.295	9.8
2006.02	0.386	10.0
2006.03	0.541	9.9
2006.04	0.519	9.4
2006.05	0.517	9.1
2006.06	0.453	8.8
2006.07	0.428	8.5
2006.08	0.435	9.0

only showing top 20 rows

이진 로지스틱 회귀 분석

```
train, test = rate_data.randomSplit([0.7, 0.3])

print("Training : %d, Test : %d" % (train.count(), test.count()))

from pyspark.ml.feature import StringIndexer, OneHotEncoder

categoricalCols = ["HPCR_Rate", "CrudeM_Rate", "CrudeDiv_Rate"]

stringIndexer = StringIndexer(inputCols=categoricalCols, outputCols=[x + "Index" for x in categoricalCols])

encoder = OneHotEncoder(inputCols=stringIndexer.getOutputCols(), outputCols=[x + "OHE" for x in categoricalCols])

labelToIndex = StringIndexer(inputCol="CrudeB_Rate", outputCol="label")

stringIndexerModel = stringIndexer.fit(train)
stringIndexerModel.transform(train).show()
```

훈련용 데이터 7, 테스트 데이터 3으로
데이터프레임을 분할해줍니다.

피처 전처리를 수행합니다. 앞서 설정해준 3개의 카테고리형 값을
StringIndexer추정기에 삽입하여 숫자로 변환,
그 숫자를 다시 원 핫인코더에 삽입하여 이진 벡터로 변환합니다.
라벨(타겟)이 될 출산율은 0을 기준으로 true/false로
나누어졌습니다.

아래 stringIndexer의 fit 메서드를 사용하여 모델을 리턴합니다.

	CrudeB	CrudeM	Nat	NatRate	Marriage	CrudeM	divorce	CrudeDiv	Se	HPCR	CrudeB_Rate	Crud
	HPCR_RateIndex	CrudeM_RateIndex	CrudeDiv_RateIndex									
5.01	5.3119197.01	4.6126928.01	6.5112071.01	2.9159.67870115	0.2781	true1						
7.01	5.3115518.01	3.9125348.01	6.319999.01	2.5160.00595092	0.4551	true1						
	0.01	0.01	0.01									
	5.1114851.01	3.6129505.01	7.1110636.01	2.6160.30646414	0.4341	true1						
	0.01	0.01	0.01									
	8.4114620.01	3.7127173.01	6.8110531.01	2.6160.76871412	0.751	true1						
	0.01	0.01	0.01									
	8.4119377.01	4.7115281.01	3.7121804.01	2.6161.23566365	0.3631	true1						
	0.01	0.01	0.01									
	8.7119485.01	4.7116518.01	4.23344.01	3.0161.46053266	0.4641	true1						
	0.01	0.01	0.01									
	9.0121268.01	5.1116015.01	3.9123867.01	2.6161.6091837	0.2991	true1						
	0.01	0.01	0.01									
	8.9120869.01	5.2114868.01	3.7131693.01	2.6161.59867564	0.3841	true1						
	0.01	0.01	0.01									
	8.0121907.01	5.3111177.01	2.7140804.01	2.4161.70038579	0.3121	true1						
	0.01	0.01	0.01									
	10.0119145.01	5.1118377.01	4.9122319.01	2.9162.14736168	0.3861	true1						
	0.01	0.01	0.01									
	9.4120360.01	5.1117306.01	4.3124364.01	2.4163.06011749	0.5191	true1						
	0.01	0.01	0.01									
	9.1120446.01	4.9117294.01	4.2131420.01	2.5163.64432608	0.5171	true1						
	0.01	0.01	0.01									
	8.8119005.01	4.7116342.01	4.1129038.01	2.7163.96566795	0.4531	true1						
	0.01	0.01	0.01									
	9.0120147.01	4.9117310.01	4.2125182.01	2.7164.27742515	0.4351	true1						
	0.01	0.01	0.01									
	9.6119730.01	4.9118741.01	4.7119207.01	2.5164.68505273	0.5021	true1						
	0.01	0.01	0.01									
	9.2120911.01	5.2116199.01	4.133716.01	2.6167.54217884	0.61	true1						
	0.01	0.01	0.01									
	8.4121654.01	5.2113181.01	3.2146099.01	2.3168.84326637	0.5531	true1						
	0.01	0.01	0.01									
	10.5122723.01	5.5120981.01	5.32882.01	2.6169.73067767	0.3721	true1						
	0.01	0.01	0.01									
	10.5119445.01	5.2120040.01	5.3127806.01	2.6170.13773971	0.3381	true1						
	0.01	0.01	0.01									
	10.5120817.01	5.22761.01	5.5127313.01	2.5170.40257509	0.2951	true1						
	0.01	0.01	0.01									

이진 로지스틱 회귀 분석

```
from pyspark.ml.feature import VectorAssembler

# "Death", "Marriage", "CrudeM", "divorce", "CrudeDiv"
numericCols = ["Nat", "CrudeDiv", "Death", "Nat", "divorce"]
assemblerInputs = [c + "OHE" for c in categoricalCols] + numericCols
vecAssembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
```

numericCols에 담긴 숫자로 된 열, 원 핫 인코딩이 된 열들을
VectorAssembler 변환기에서 하나의 벡터로 결합합니다.

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol="features", labelCol="label", regParam=1)
```

위에서 OUTPUT으로 설정한 features를 피쳐로 하고
regParam(페널티 항)을 1로 하는
로지스틱 회귀 분석 모델을 정의합니다.

```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[stringIndexer, encoder, labelToIndex, vecAssembler, lr])
pipelineModel = pipeline.fit(train)

predDF = pipelineModel.transform(test)
```

앞에서 정의된 모든 과정을 하나의 파이프라인에 구축하고
훈련 데이터를 fit하여 파이프라인 학습 모델을 구축합니다.
앞서 분할해둔 데이터를 적용합니다.

이진 로지스틱 회귀 분석

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(metricName = "areaUnderROC")
print("예측 정확도 : ", evaluator.evaluate(predDF))
```

예측 정확도 : 0.9298941798941799

생성한 모델에 테스트 데이터를 삽입하여 정확도를 분석합니다.

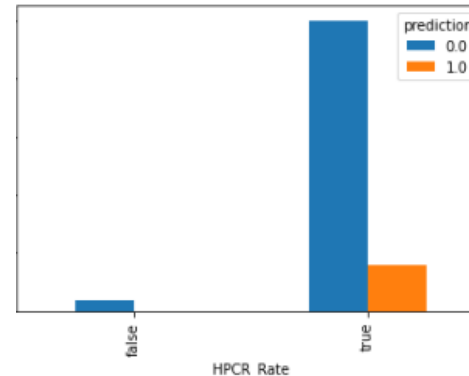
92.9%로 굉장히 높은 편인데

이는 데이터의 구성때문입니다.

**실제로 대한민국의 출산율이 해가 갈수록 낮아지고 있어
대부분의 지표가 false를 가리키고 있습니다.
이로 인해 예측 정확도가 굉장히 높게 책정됩니다.**

```
HPCRRATEDF = predDF.select('HPCR_Rate', 'prediction').groupby('HPCR_Rate', 'prediction').count()
HPCRRATEDF_pd = HPCRRATEDF.toPandas().pivot_table('count', index='HPCR_Rate', columns='prediction')
HPCRRATEDF_pd.plot.bar()
```

<AxesSubplot: xlabel='HPCR_Rate'>



현재의 모델을 가지고 예측한 값입니다.
집값이 내린 데이터는 거의 없어 분석하기 어려움이 있고
집값이 오를때, 출산율이 떨어진다는 예측이
압도적으로 많습니다

이진 로지스틱 회귀 분석

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.7, 2.0]) #
.addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) #
.build()

cv = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=3)

cvModel = cv.fit(train)

cvPredDF = cvModel.transform(test)

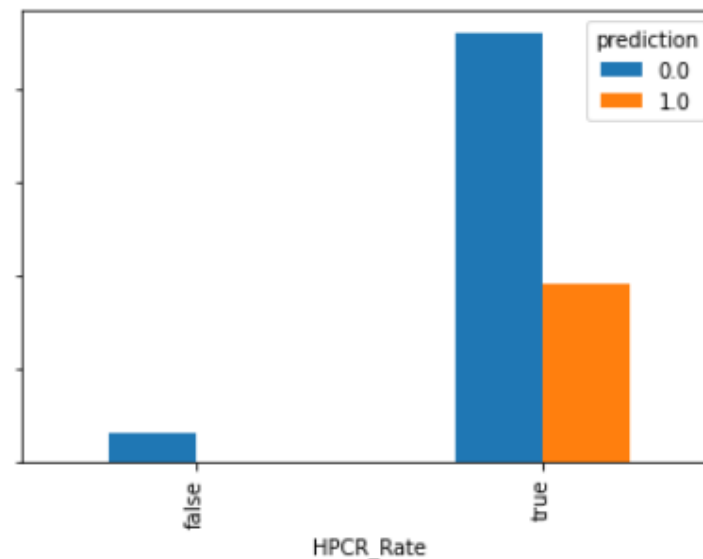
print("예측 정확도: ", evaluator.evaluate(cvPredDF))
```

예측 정확도: 0.9604863221884499

CrossValidator를 사용하여 하이퍼 파라미터 튜닝을 적용했습니다.

**앞서 생성한 파이프라인에 6개의 파라미터를 삽입하여 튜닝을 적용해줍니다.
동일하게 훈련 데이터를 fit하고, 테스트 데이터에도 모델을 적용합니다.
예측 정확도가 앞선 버전에 비해 비약적으로 상승한 것을 알 수 있습니다.**

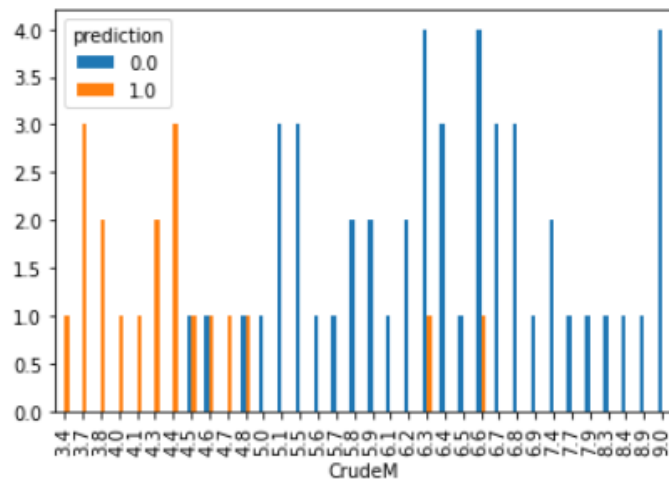
**우측은 하이퍼파라미터 튜닝이 적용된 모델의 예측입니다.
앞선 모델에 비해 출산율이 오를 확률이 소폭 상승하였습니다.**



이진 로지스틱 회귀 분석

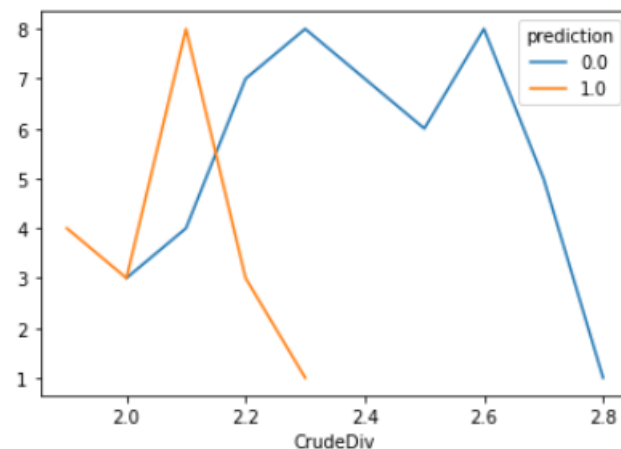
```
tpocc_df = cvPredDF.select('CrudeM', 'prediction').groupby('CrudeM', 'prediction').count().toPandas()
tpocc_pv = tpocc_df.pivot_table('count', index='CrudeM', columns='prediction')
tpocc_pv.plot.bar()
```

<AxesSubplot: xlabel='CrudeM'>



```
In [77]: tpocc_df = cvPredDF.select('CrudeDiv', 'prediction').groupby('CrudeDiv', 'prediction').count().toPandas()
tpocc_pv = tpocc_df.pivot_table('count', index='CrudeDiv', columns='prediction')
tpocc_pv.plot.line()
```

Out [77]: <AxesSubplot: xlabel='CrudeDiv'>



좌측이 결혼률, 우측이 이혼률에 관한 예측입니다.

결혼률 3.4~4.8까지는 출산율이 올라갈 확률이 올라가고, 이후로는 출산율이 내려갈 확률이 더 높습니다.
이혼률의 2.3까지는 출산율이 올라갈 확률이 높지만, 이후로는 출산율이 낮아질 확률이 압도적으로 높습니다.

모니터링



NEW,NEW_SAVING,SUBMITTED,ACCEPTED,RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total
24	0	1	23	3	5 GB	8 GB	0 B	3	8

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Clusters
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:4096, vCores:4>	0

Dump scheduler logs 1 min

Application Queues



Total Resource Preempted: <memory:0, vCores:0>

Total Number of Non-AM Containers Preempted: 0

Total Number of AM Containers Preempted: 0

Resource Preempted from Current Attempt: <memory:0, vCores:0>


Number of Non-AM Containers Preempted from Current Attempt: 0

Aggregate Resource Allocation: 16232221 MB-seconds, 9512 vcore-seconds

Aggregate Preempted Resource Allocation: 0 MB-seconds, 0 vcore-seconds

application_1654886291530_0024	bigdata	PySparkShell	SPARK	default	0	Sun Jun 12 04:37:14 +0900 2022	Sun Jun 12 04:37:15 +0900 2022	Mon Jun 13 05:44:18 +0900 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1654886291530_0023	bigdata	PySparkShell	SPARK	default	0	Sat Jun 11 21:43:03 +0900 2022	Sat Jun 11 21:43:04 +0900 2022	Sun Jun 12 04:36:41 +0900 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1654886291530_0022	bigdata	PySparkShell	SPARK	default	0	Sat Jun 11 19:53:30 +0900 2022	Sat Jun 11 19:53:31 +0900 2022	Sat Jun 11 21:42:32 +0900 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1654886291530_0021	bigdata	PySparkShell	SPARK	default	0	Sat Jun 11 19:31:04 +0900 2022	Sat Jun 11 19:31:05 +0900 2022	Sat Jun 11 19:52:59 +0900 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1654886291530_0020	bigdata	PySparkShell	SPARK	default	0	Sat Jun 11 19:29:06 +0900 2022	Sat Jun 11 19:30:28 +0900 2022	Sat Jun 11 19:30:32 +0900 2022	FINISHED	SUCCEEDED	N/A	N/A	N/A

```
2022-06-12 20:50:00,685 INFO org.apache.hadoop.hdfs.server.datanode.DirectoryScanner: Scan Results: BlockPool BP-1908648111-192.168.100.200-1649171775038 Total blocks: 1152, missing metadata files: 0, missing block files: 0, missing blocks in memory: 0, mismatched blocks: 0
2022-06-12 20:57:56,918 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Successfully sent block report 0xd997b815419f86d, containing 1 storage report(s), of which we sent 1. The reports had 1152 total blocks and used 1 RPC(s). This took 1 msec to generate and 1 msec for RPC and NN processing. Got back one command: FinalizeCommand/5.
2022-06-12 20:57:56,918 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Got finalize command for block pool BP-1908648111-192.168.100.200-1649171775038
2022-06-12 21:39:45,770 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving BP-1908648111-192.168.100.200-1649171775038:blk_1073743681_2857 src: /192.168.100.200:49278 dest: /192.168.100.200:9866
2022-06-12 21:39:45,776 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /192.168.100.200:49278, dest: /192.168.100.200:9866, bytes: 29, op: HDFS_WRITE, cliID: DFSCliet_NONMAPREDUCE_1359881897.16, offset: 0, srvid: 6e372c1d-1afb-4f85-9d29-31b9c095dde5, blockid: BP-1908648111-192.168.100.200-1649171775038:blk_1073743681_2857, duration(ns): 3187591
2022-06-12 21:39:45,776 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder: BP-1908648111-192.168.100.200-1649171775038:blk_1073743681_2857, type=HAS_DOWNSTREAM_IN_PIPELINE, downstreams=1:[192.168.100.201:9866] terminating
2022-06-12 21:39:45,483 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving BP-1908648111-192.168.100.200-1649171775038:blk_1073743682_2858 src: /192.168.100.200:49288 dest: /192.168.100.200:9866
2022-06-12 21:39:45,492 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /192.168.100.200:49288, dest: /192.168.100.200:9866, bytes: 534, op: HDFS_WRITE, cliID: DFSCliet_NONMAPREDUCE_1462426861.16, offset: 0, srvid: 6e372c1d-1afb-4f85-9d29-31b9c095dde5, blockid: BP-1908648111-192.168.100.200-1649171775038:blk_1073743682_2858, duration(ns): 5299311
2022-06-12 21:39:45,492 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder: BP-1908648111-192.168.100.200-1649171775038:blk_1073743682_2858, type=HAS_DOWNSTREAM_IN_PIPELINE, downstreams=1:[192.168.100.201:9866] terminating
2022-06-12 21:42:46,708 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving BP-1908648111-192.168.100.200-1649171775038:blk_1073743683_2859 src: /192.168.100.200:49310 dest: /192.168.100.200:9866
2022-06-12 21:42:46,714 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /192.168.100.200:49310, dest: /192.168.100.200:9866, bytes: 29, op: HDFS_WRITE, cliID: DFSCliet_NONMAPREDUCE_1359881897.16, offset: 0, srvid: 6e372c1d-1afb-4f85-9d29-31b9c095dde5, blockid: BP-1908648111-192.168.100.200-1649171775038:blk_1073743683_2859, duration(ns): 3170789
2022-06-12 21:42:46,714 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder: BP-1908648111-192.168.100.200-1649171775038:blk_1073743683_2859, type=HAS_DOWNSTREAM_IN_PIPELINE, downstreams=1:[192.168.100.201:9866] terminating
2022-06-12 21:42:46,757 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving BP-1908648111-192.168.100.200-1649171775038:blk_1073743684_2860 src: /192.168.100.200:49314 dest: /192.168.100.200:9866
2022-06-12 21:42:46,763 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /192.168.100.200:49314, dest: /192.168.100.200:9866, bytes: 534, op: HDFS_WRITE, cliID: DFSCliet_NONMAPREDUCE_1462426861.16, offset: 0, srvid: 6e372c1d-1afb-4f85-9d29-31b9c095dde5, blockid: BP-1908648111-192.168.100.200-1649171775038:blk_1073743684_2860, duration(ns): 3106207
2022-06-12 21:42:46,763 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder: BP-1908648111-192.168.100.200-1649171775038:blk_1073743684_2860, type=HAS_DOWNSTREAM_IN_PIPELINE, downstreams=1:[192.168.100.201:9866] terminating
2022-06-12 21:42:47,374 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving BP-1908648111-192.168.100.200-1649171775038:blk_1073743685_2861 src: /192.168.100.200:49320 dest: /192.168.100.200:9866
2022-06-12 21:42:47,379 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /192.168.100.200:49320, dest: /192.168.100.200:9866, bytes: 29, op: HDFS_WRITE, cliID: DFSCliet_NONMAPREDUCE_1359881897.16, offset: 0, srvid: 6e372c1d-1afb-4f85-9d29-31b9c095dde5, blockid: BP-1908648111-192.168.100.200-1649171775038:blk_1073743685_2861, duration(ns): 2257272
2022-06-12 21:42:47,380 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder: BP-1908648111-192.168.100.200-1649171775038:blk_1073743685_2861, type=HAS_DOWNSTREAM_IN_PIPELINE, downstreams=1:[192.168.100.201:9866] terminating
```



감사합니다

