

Báo cáo kỹ thuật công nghệ và dữ liệu lớn

Thành viên nhóm

Nguyễn Thị Ngọc Lan - 23020390
Tạ Giang Thuỳ Loan - 23020397
Hoàng Ngọc Điệp - 23020357

Đề bài

Classifying in KNIME to identify big spenders in Catch the Pink Flamingo

Bảng phân công

Github: <https://github.com/bokuwakira1411/BigData/blob/main/part-00000-72b64e25-854b-4d07-b3fa-bf386e06a3b4-c000.csv>

Tên	Việc được phân công
Nguyễn Thị Ngọc Lan	Chọn và xử lý các đặc trưng, huấn luyện và đánh giá mô hình trên Spark
Tạ Giang Thuỳ Loan	Gán nhãn dữ liệu và trực quan hoá dữ liệu bằng KMINE
Hoàng Ngọc Điệp	Phân tích dữ liệu và trực quan hoá dữ liệu bằng PD

Tóm tắt bài toán và cách xử lý

Bài toán xác định big spenders trong trò Catch the Pink Flamingo cần trải qua các công đoạn chính:

- Chuẩn bị dữ liệu và gán nhãn
- Phân tích dữ liệu và đánh giá
- Tiền xử lý dữ liệu và chọn đặc trưng
- Huấn luyện mô hình và đánh giá trên tập test

Một người được coi là big spender nếu người đó chi tiêu tổng số tiền lớn hơn hoặc bằng 1 ngưỡng nhất định (Rule Engine hoặc KMeans). Khi huấn luyện mô hình, dữ liệu được dùng chỉ là 3 session đầu tiên của mỗi người dùng nhằm mục đích dự đoán sớm rằng người đó có phải một big spender không. Và với những bài toán như thế này, big spender thì ít nhưng họ lại tạo ra rất nhiều doanh thu cho chủ thầu vậy nên phải để mô hình cố gắng phân tích được chính xác nhiều big spender nhất có thể mặc dù dữ liệu có bị mất cân bằng để đưa ra được chiến lược tiếp thị phù hợp.

Nhãn: big spender - big spender, low spender - người dùng không chi tiêu nhiều

1 Phân tích dữ liệu

1.1 Gán nhãn và trực quan hoá bằng KMINE

Để gán nhãn bằng KMINE, trước hết có 2 hướng tiếp cận là Rule Engine (theo luật người dùng định nghĩa) và theo KMeans. Sau khi phân tích và lựa chọn ta sẽ thu được 1 tập chứa các nhãn tương ứng với big spender và low spender.

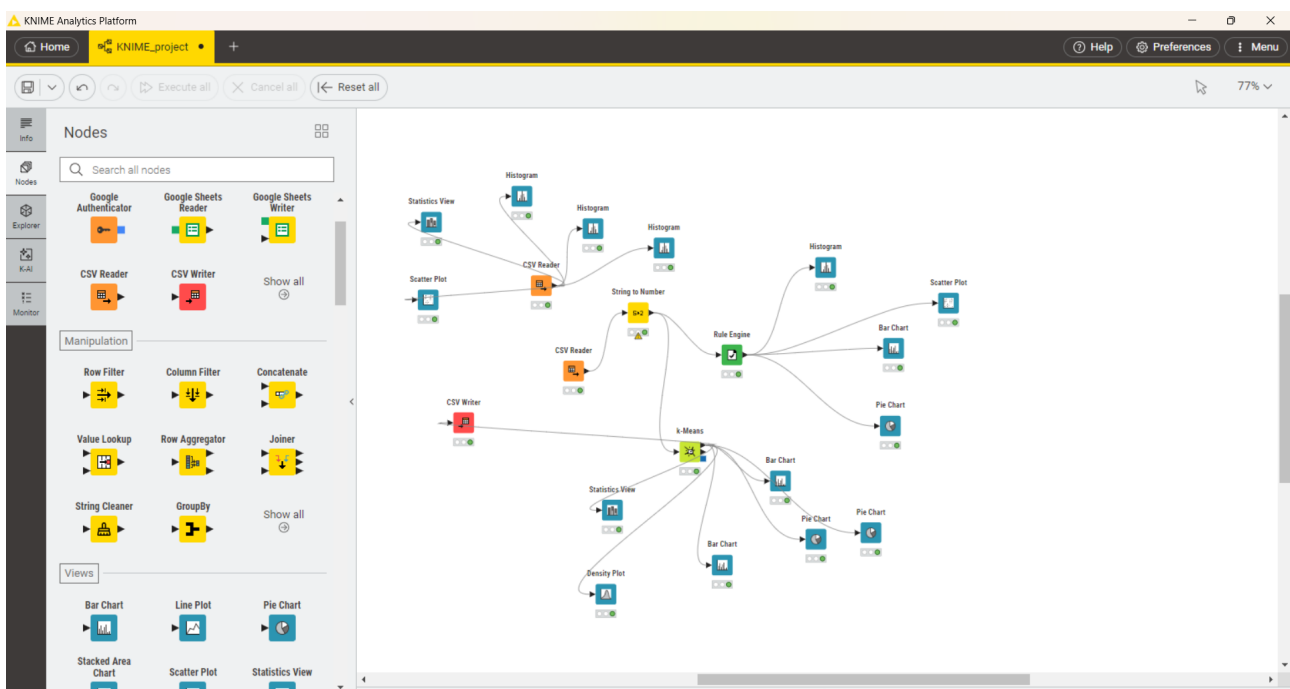
Theo Rule Engine, ta có thể lấy 5% số người chi nhiều tiền nhất để định nghĩa big spender, với mức chi tiêu mốc là 70 đô. Và với KMeans ta sẽ chọn số cluster là 2. Kết quả cho thấy KMeans có vẻ nhận biết được nhiều big spender hơn so với Rule Engine theo luật tự định nghĩa

```
[ ] quantile = user_stats.approxQuantile("total_spent", [0.95], 0.01)[0]
```

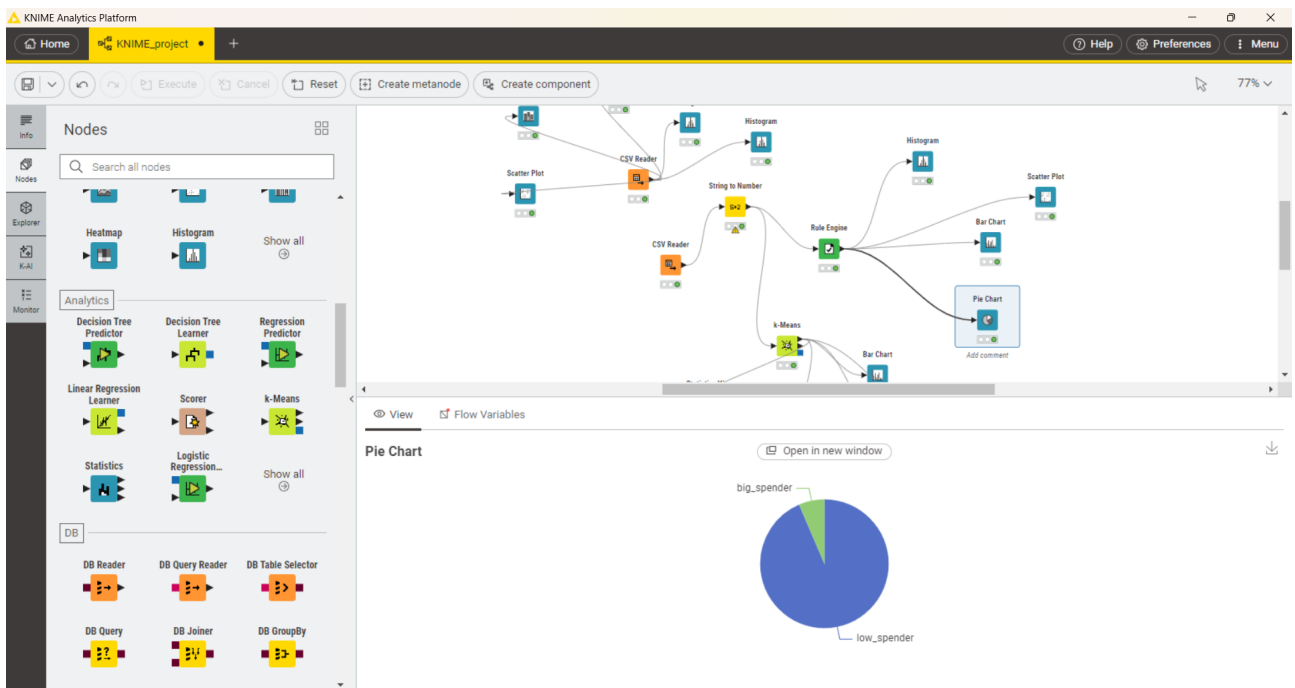
```
[ ] quantile
```

↔ 70.0

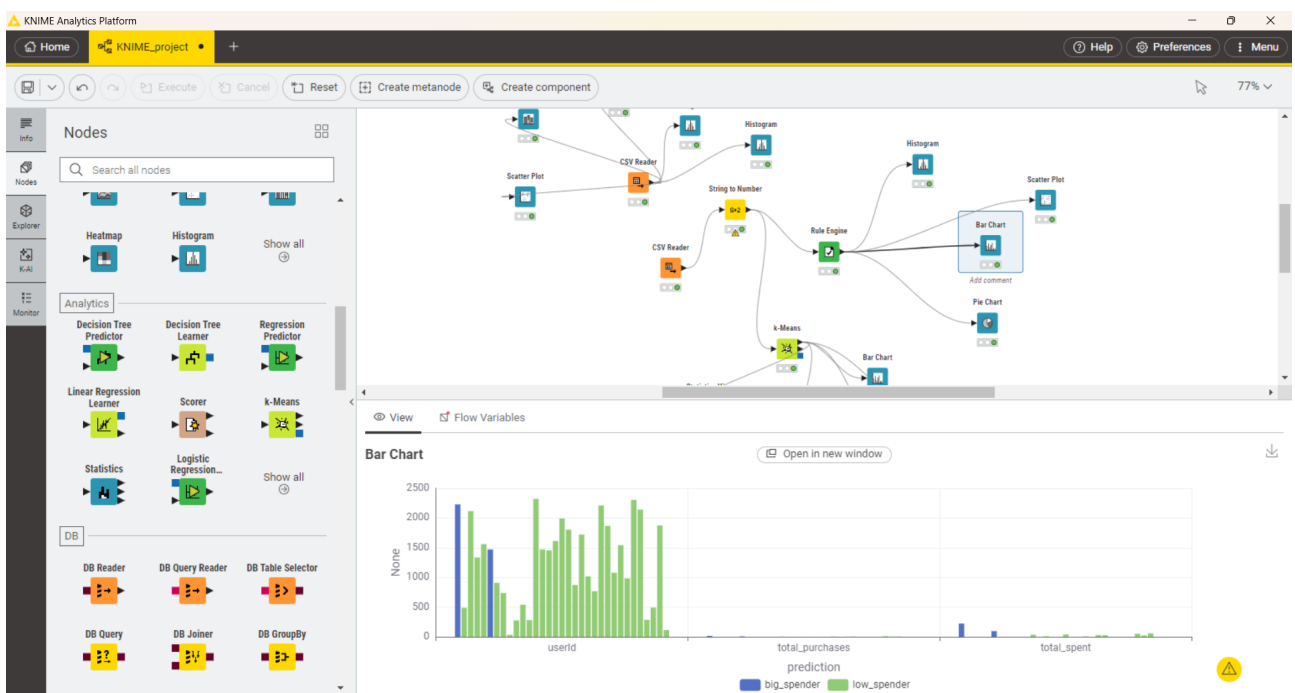
Hình 1: Tính ngưỡng chi tiêu để phân loại phân loại big spender và low spender theo Rule Engine



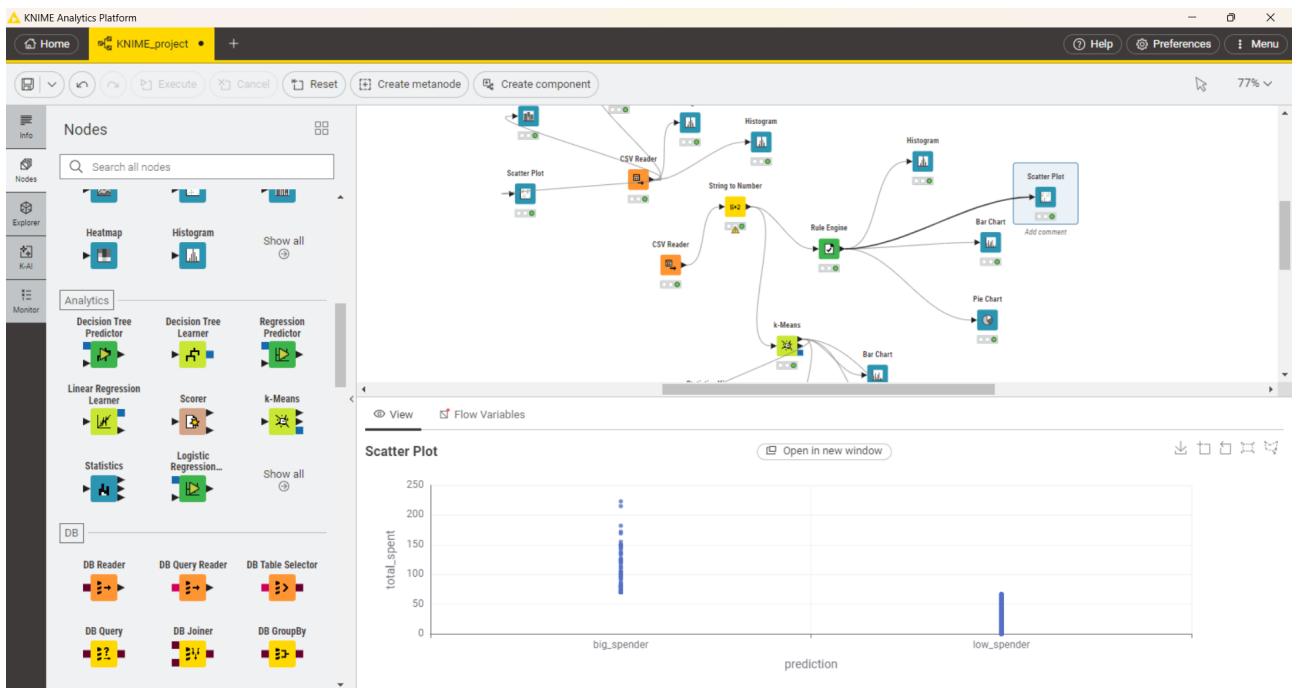
Hình 2: Giao diện làm việc với KMINE



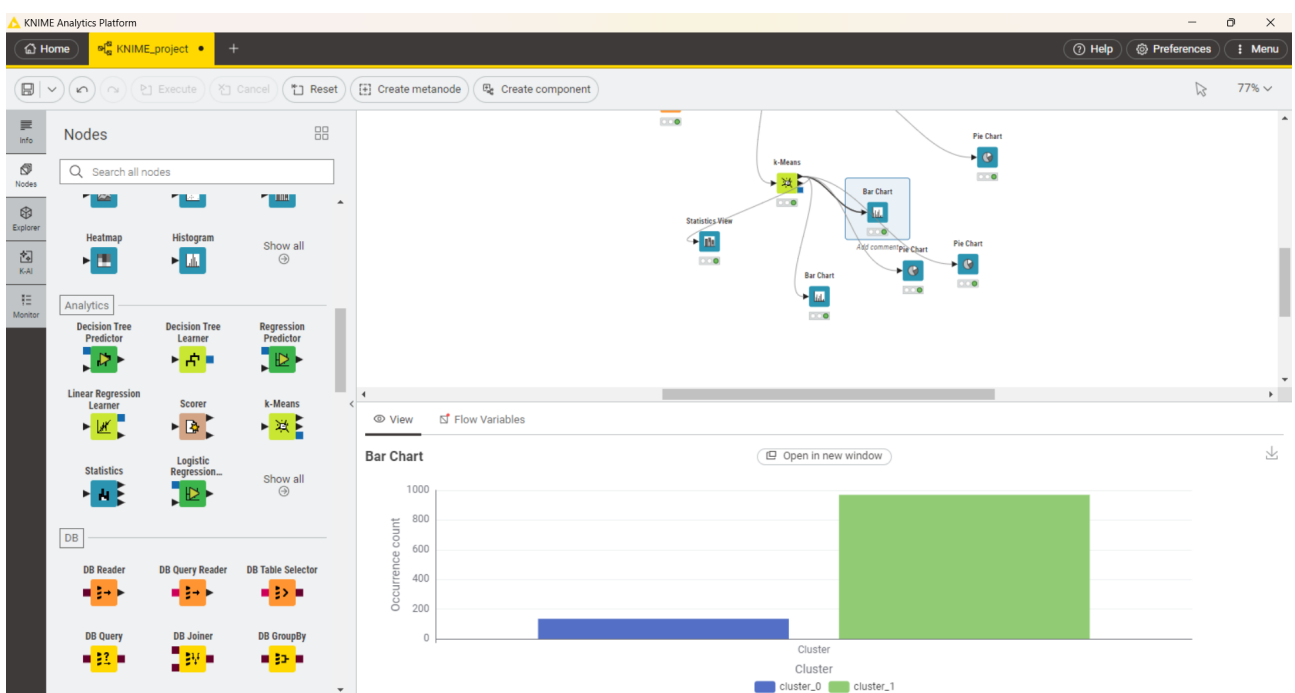
Hình 3: Biểu đồ tròn nếu phân big spender và low spender theo Rule Engine



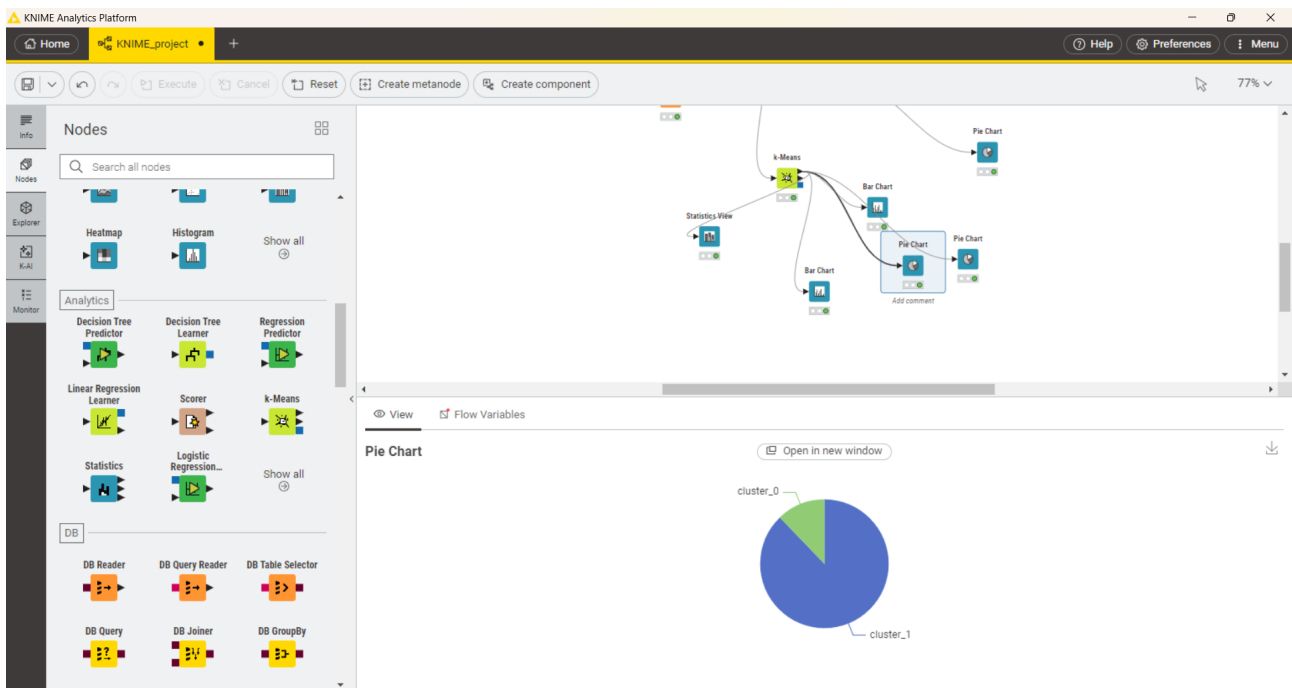
Hình 4: Biểu đồ cột nếu phân big spender và low spender theo Rule Engine



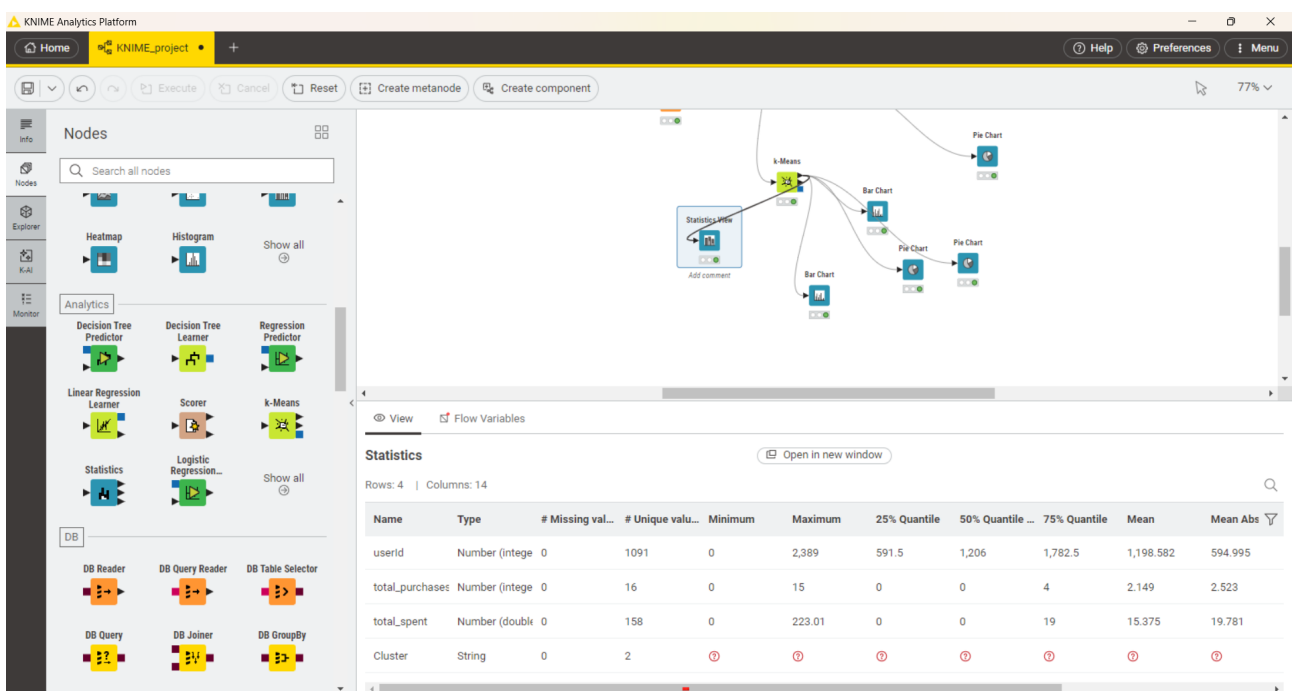
Hình 5: Biểu đồ phân tán nếu phân big spender và low spender theo Rule Engine



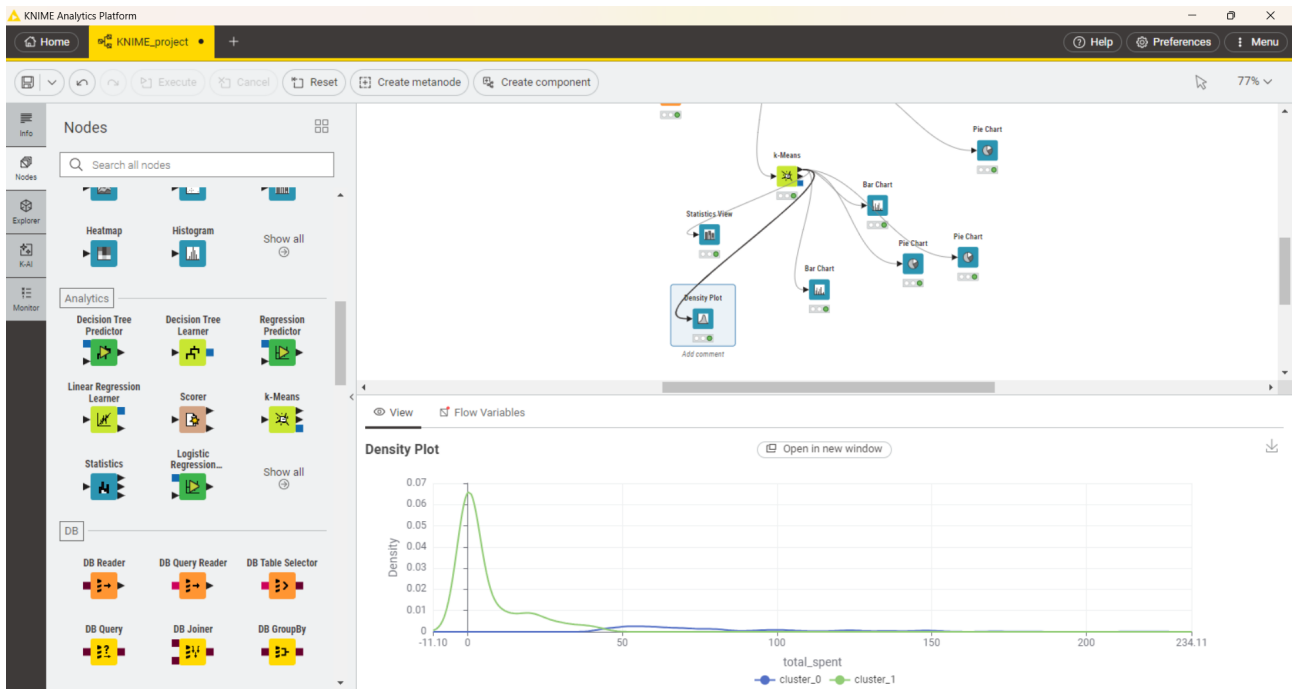
Hình 6: Biểu đồ cột nếu phân big spender (cluster 0) và low spender (cluster 1) theo KMeans



Hình 7: Biểu đồ tròn nếu phân big spender (cluster 0) và low spender (cluster 1) theo KMeans



Hình 8: Các chỉ số thống kê nếu phân big spender (cluster 0) và low spender (cluster 1) theo KMeans



Hình 9: Biểu đồ đường theo nhãn big spender (cluster 0) và low spender (cluster 1) theo KMeans

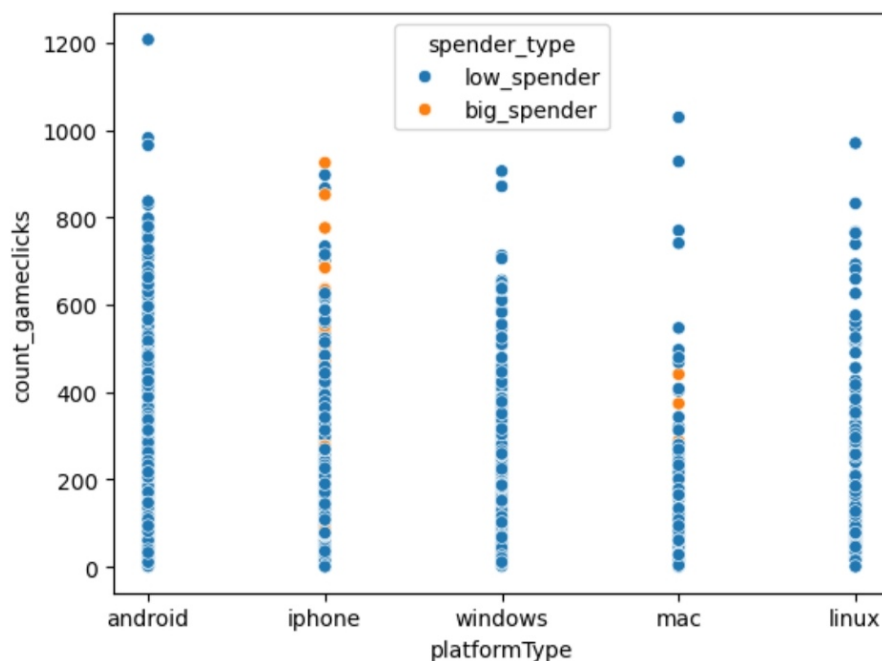
#	RowID	userid	platformType	total_purchases	total_spent	Cluster
1	Row0	2229	iphone	15	223.01	cluster_0
2	Row1	487	iphone	0	0	cluster_1
3	Row2	2116	android	0	0	cluster_1
4	Row3	1335	windows	0	0	cluster_1
5	Row4	1560	android	0	0	cluster_1
6	Row5	1471	iphone	8	96	cluster_0
7	Row6	908	mac	0	0	cluster_1
8	Row7	738	iphone	0	0	cluster_1
9	Row8	32	android	0	0	cluster_1
10	Row9	275	android	0	0	cluster_1
11	Row10	540	android	0	0	cluster_1
12	Row11	281	android	5	34	cluster_1

Hình 10: Bảng dữ liệu cuối cùng phân nhãn theo KMeans

1.2 Phân Tích Dữ Liệu

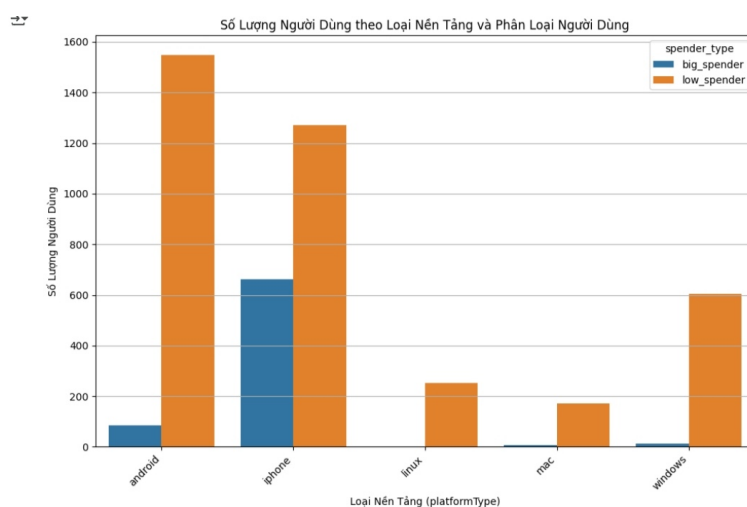
Sau khi hoàn tất các bước tiền xử lý dữ liệu, bao gồm việc xử lý các giá trị NULL trong cột count_buyId và avg_price (bằng cách thay thế bằng 0 và chuyển đổi kiểu dữ liệu phù hợp), tạo cột made_purchase để xác định các phiên có giao dịch, và tạo cột mục tiêu spender_type (big spender/low spender) dựa trên total_spent của mỗi người dùng, em đã tiến hành phân tích sâu hơn để hiểu rõ hơn về các đặc điểm của dữ liệu và mối quan hệ giữa các biến.

Nhận xét: Biểu đồ phân tán Hình 11: Biểu đồ phân tán số lượt nhấp chuột trong game theo loại nền tảng và phân loại người chi tiêu cho thấy sự phân bố của count_gameclicks đối với hai nhóm **big**



Hình 11: Biểu đồ phân tán: Số Lượt Nhấp Chuột Trong Game theo Loại Nền Tảng và Phân Loại Người Chi Tiêu.

spender và **low spender** trên các nền tảng khác nhau. Nhìn chung, không có sự khác biệt hoàn toàn về số lượt nhấp chuột giữa hai nhóm trên đối với các nền tảng. Người dùng Android và iPhone, thuộc cả hai nhóm chi tiêu, đều có những phiên với số lượt nhấp chuột rất cao. **Kết Luận:** count_gameclicks có thể không phải là yếu tố quyết định duy nhất để phân loại người dùng.

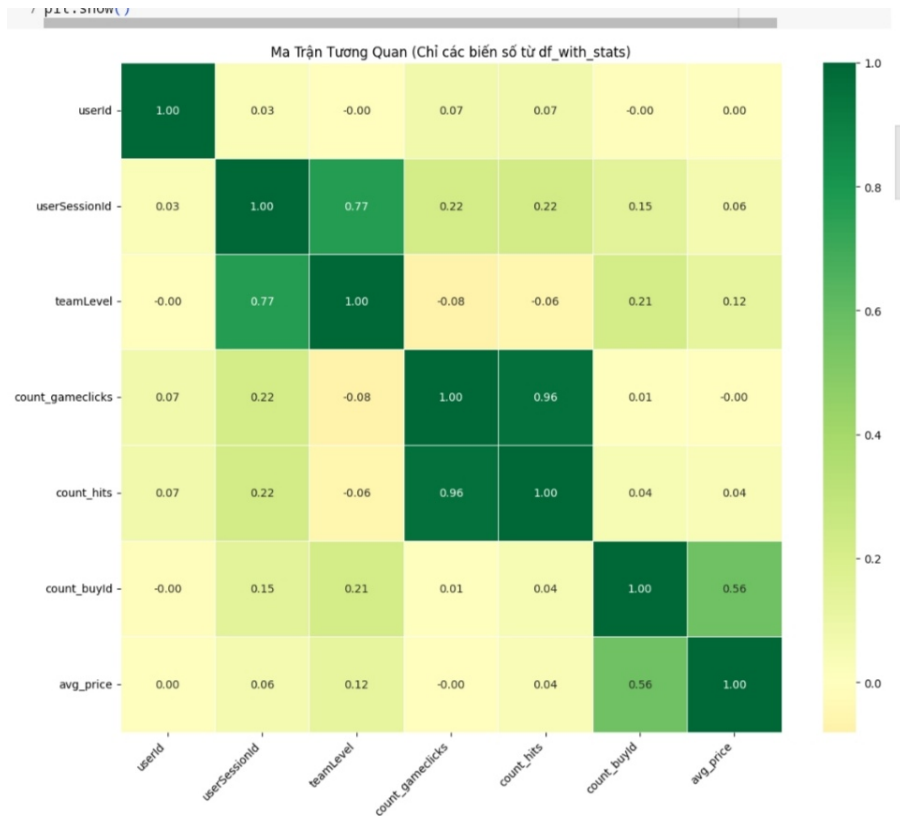


Hình 12: Biểu đồ cột: Số Lượng Người Dùng theo Loại Nền Tảng và Phân Loại Người Chi Tiêu.

Nhận xét: Biểu đồ cột (Hình 12) Biểu đồ cột Số Lượng Người Dùng theo Loại Nền Tảng và Phân Loại Người Chi Tiêu trực quan hóa số lượng phiên người dùng được phân loại là low spenders và big spender trên từng nền tảng.

- Đa số phiên được xác định của big spender diễn ra trên nền tảng iPhone và Android.
- **low spenders đa dạng nền tảng:** Nhóm low spenders xuất hiện trên tất cả các nền tảng, với số lượng lớn nhất trên iPhone và Android.

Kết Luận: Nền tảng (platformType) là một đặc trưng cực kỳ quan trọng. Người dùng sử dụng iPhone và Android có tỉ lệ trở thành big spender cao hơn các nền tảng (platform) khác.

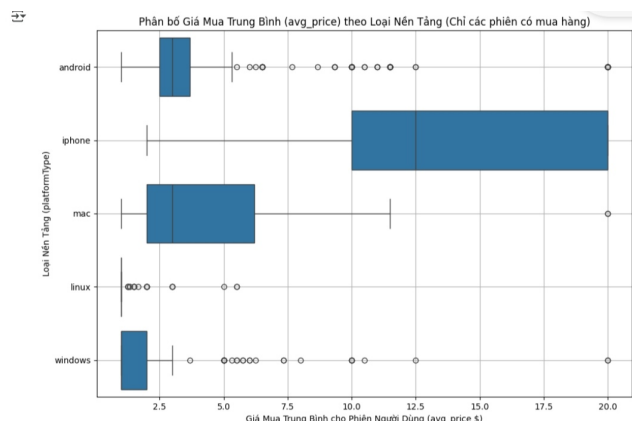


Hình 13: Heatmap: Ma Trận Tương Quan giữa các Chỉ Số.

Nhận xét: Ma trận tương quan (Hình 13) giữa các biến số ban đầu cho thấy:

- count_gameclicks có mối tương quan dương rất mạnh (0.96) với count_hits.
- userSessionId có tương quan dương khá cao với teamLevel (0.77).
- avg_price có tương quan dương yếu với count_buyId (0.56).
- Quan trọng nhất đối với bài toán, avg_price gần như không có tương quan tuyến tính đáng kể với các chỉ số hoạt động khác trong game.

Phân tích phân bố giá mua trung bình (avg_price) theo nền tảng (platformType)



Hình 14: Biểu đồ hộp: Phân bố Giá Mua Trung Bình theo Loại Nền Tảng (Chỉ các phiên có mua hàng).

Nhận xét: Khi chỉ xem xét các phiên có giao dịch mua hàng ($avg_price > 0$), biểu đồ hộp (Hình 14) của avg_price theo từng nền tảng cho thấy:

- Người dùng iPhone có xu hướng có khoảng tứ phân vị và các giá trị ngoại lệ ở mức giá mua trung bình cao hơn rõ rệt so với các nền tảng khác.

- **Các nền tảng khác:** Android và Windows có phân bố giá rộng hơn ở các mức thấp và trung bình. Mac và Linux có ít dữ liệu hơn nhưng cũng tập trung ở các mức giá thấp hơn iPhone.

Kết luận sơ bộ từ Phân tích dữ liệu:


Qua các phân tích trên, có thể thấy platformType (đặc biệt là iPhone) và avg_price (hoặc các biến liên quan đến tổng chi tiêu) là những đặc trưng rất hứa hẹn cho việc phân loại big spender và low spenders. Các hành vi trong game như count_gameclicks hay teamLevel có thể đóng vai trò hỗ trợ nhưng không phải là yếu tố quyết định duy nhất.

2 Tiền xử lý dữ liệu và chọn đặc trưng

2.1 Chọn đặc trưng

Nhóm lần lượt các trường dữ liệu theo spender type để biết được mối liên hệ

```
[40] df_with_stats.groupBy("spender_type", "teamLevel").count().orderBy("spender_type", "teamLevel").show()
```




spender_type	teamLevel	count
big_spender	1	82
big_spender	2	94
big_spender	3	102
big_spender	4	117
big_spender	5	128
big_spender	6	129
big_spender	7	117
big_spender	8	1
low_spender	1	375
low_spender	2	432
low_spender	3	506
low_spender	4	570
low_spender	5	639
low_spender	6	690
low_spender	7	634
low_spender	8	3

Hình 15: Bảng phân bố số lượng người chơi dựa trên spender_type và teamLevel

Nhóm spender type theo team level: ta có thể thấy người chơi theo từng loại spender phân bố khá đều theo từng team level, và có một xu hướng tăng khi team level tăng lên, đây có thể là một đặc trưng có ích nên giữ lại.

```
[42] df_with_stats.groupBy("spender_type", "platformType").count().orderBy("spender_type", "platformType").show()
```



spender_type	platformType	count
big_spender	android	86
big_spender	iphone	663
big_spender	mac	7
big_spender	windows	14
low_spender	android	1549
low_spender	iphone	1271
low_spender	linux	252
low_spender	mac	172
low_spender	windows	605

Hình 16: Bảng phân bố số lượng người chơi dựa trên spender_type và platformType

Nhóm spender type theo platform type: ta có thể thấy sự phân biệt rõ rệt về lượng người big spender và low spender ở các nền tảng (big spender tập trung hoàn toàn ở iphone), đặc trưng này cũng nên giữ lại. Tương tự với

các đặc trưng khác, phân tích đánh giá xem có nên giữ lại hay thay thế

2.2 Tiền xử lý dữ liệu

Vì mục tiêu của bài toán là nhanh chóng dự đoán được người chơi có tiềm năng là big spender không nên ta sẽ chỉ lấy 3 session đầu của mỗi người chơi, và không lấy toàn bộ dữ liệu để huấn luyện. Tránh trường hợp rò rỉ dữ liệu, do ta phân nhãn hoàn toàn dựa trên tổng lượng giá trị người dùng đã dùng.

```
[45] from pyspark.sql.window import Window
      from pyspark.sql.functions import row_number

      window = Window.partitionBy("userId").orderBy("userSessionId")
      df_early = df_with_stats.withColumn("session_rank", row_number().over(window)).filter("session_rank <= 3")

df_early.show()
```

userId	platformType	userSessionId	teamLevel	count_gameclicks	count_hits	count_buyId	avg_price	made_purchase	total_purchases	total_spent	spender_type	session_rank
0	iphone	23473	1	237	28	0	0.0	0	0	0.0	low_spender	1
0	iphone	24943	2	353	35	0	0.0	0	0	0.0	low_spender	2
0	iphone	28377	3	381	32	0	0.0	0	0	0.0	low_spender	3
1	android	5835	1	73	7	0	0.0	0	8	19.0	low_spender	1
1	android	7847	2	95	14	0	0.0	0	8	19.0	low_spender	2
1	android	10041	3	93	9	2	3.0	1	8	19.0	low_spender	3
2	iphone	12107	3	7	0	0	0.0	0	0	0.0	low_spender	1
2	iphone	12411	4	39	3	0	0.0	0	0	0.0	low_spender	2
2	iphone	15899	5	35	6	0	0.0	0	0	0.0	low_spender	3
6	iphone	28628	7	90	5	0	0.0	0	0	0.0	low_spender	1
8	iphone	27410	4	25	1	0	0.0	0	3	33.0	low_spender	1

Hình 17: Bảng các đặc trưng của dữ liệu trong 3 session đầu tiên

Vì người chơi trải qua nhiều session, nên khi lấy dữ liệu để làm đặc trưng thì phải tìm cách gom nhóm lại, ở đây nhóm chúng em chọn cách tính trung bình cho các đặc trưng mang giá trị số có thể tính như lượt click, lượt hit trong game,...

```
[51] agg_df = df.groupBy("userId", "platformType").agg(
      F.avg("count_gameclicks").alias("avg_clicks"),
      F.avg("count_hits").alias("avg_hits"),
      F.sum("count_buyId").alias("total_early_buys"),
      F.avg("avg_price").alias("early_avg_price"),
      F.max("made_purchase").alias("has_early_purchase"),
      F.count("*").alias("session_count")
    )

agg_df.show()
```

userId	platformType	avg_clicks	avg_hits	total_early_buys	early_avg_price	has_early_purchase	session_count
0	iphone	323.6666666666667	31.666666666666668	0	0.0	0	3
1	android	87.0	10.0	2	1.0	1	3
2	iphone	27.0	3.0	0	0.0	0	3
6	iphone	90.0	5.0	0	0.0	0	1
8	iphone	103.5	12.0	3	5.5	1	2
9	iphone	54.0	6.333333333333333	2	6.666666666666667	1	3
10	linux	719.0	81.66666666666667	4	0.6666666666666666	1	3
12	iphone	78.66666666666667	7.333333333333333	2	10.0	1	3

Hình 18: Bảng các đặc trưng của dữ liệu sau khi đã gom nhóm

Và riêng team level, nhóm em nhận thấy số lượng người chơi có vẻ tăng lên theo team level, và do người chơi có thể tham gia ở những team level khác nhau ở những session khác nhau nên cũng cần gom nhóm lại. Ở đây nhóm em thay thế danh sách các team level cho mỗi người dùng thành max team level (giá trị team level lớn nhất).

```
df_with_stats.groupBy('spender_type', 'teamLevel').count().orderBy('teamLevel', 'spender_type').show()
```

spender_type	teamLevel	count
big_spender	1	82
low_spender	1	375
big_spender	2	94
low_spender	2	432
big_spender	3	102
low_spender	3	506
big_spender	4	117
low_spender	4	570
big_spender	5	128
low_spender	5	639
big_spender	6	129
low_spender	6	690
big_spender	7	117
low_spender	7	634
big_spender	8	1
low_spender	8	3

Hình 19: Bảng phân bố số lượng người chơi dựa trên spender_type và teamLevel

```
[57] team_feature = df_temp.groupBy("userId").agg(
    max("teamLevel").cast("int").alias("max_teamLevel")
)
```

```
[58] final_features = df_temp.join(team_feature, on="userId", how="left")
```

```
[59] final_features = final_features.drop('teamLevel').distinct()
```

```
final_features.show()
```

	userId	platformType	avg_clicks	avg_hits	total_early_buys	early_avg_price	session_count	spender_type	max_teamLevel
0		iphone	323.6666666666667	31.66666666666668	0	0.0	3	low_spender	3
1		android	87.0	10.0	2	1.0	3	low_spender	3
2		iphone	27.0	3.0	0	0.0	3	low_spender	5
6		iphone	90.0	5.0	0	0.0	1	low_spender	7
8		iphone	103.5	12.0	3	5.5	2	low_spender	5
9		iphone	54.0	6.333333333333333	2	6.666666666666667	3	big_spender	4
10		linux	719.0	81.66666666666667	4	0.6666666666666666	3	low_spender	3
12		iphone	78.66666666666667	7.333333333333333	2	10.0	3	big_spender	3
13		android	172.0	16.5	4	4.0	2	low_spender	7
14		linux	93.0	6.0	0	0.0	1	low_spender	7

Hình 20: Bảng các đặc trưng sau khi chọn và gom nhóm, thay thế.

2.3 Chuẩn hoá và chuyển đổi kiểu đặc trưng

Các trường chứa dữ liệu kiểu string như spender type và platform type được chuyển đổi về dạng index. Riêng Platform type sẽ được chuyển sang dạng OneHotEncoder sau khi chuyển về dạng index vì để giữ các nền tảng không tính khoảng cách thứ tự.

```
[62] spender_indexer = StringIndexer(inputCol="spender_type", outputCol="spender_type_index")
platform_indexer = StringIndexer(inputCol="platformType", outputCol="platform_index")
final_features = spender_indexer.fit(final_features).transform(final_features)
final_features = platform_indexer.fit(final_features).transform(final_features)

[63] final_features = final_features.drop('platformType', 'spender_type')

final_features.show()
```

userId	avg_clicks	avg_hits	total_early_buys	early_avg_price	session_count	max_teamLevel	spender_type_index	platform_index
0	323.6666666666667	31.66666666666668	0	0.0	3	3	0.0	0.0
1	87.0	10.0	2	1.0	3	3	0.0	1.0
2	27.0	3.0	0	0.0	3	5	0.0	0.0
6	90.0	5.0	0	0.0	1	7	0.0	0.0
8	103.5	12.0	3	5.5	2	5	0.0	0.0
9	54.0	6.333333333333333	2	6.666666666666667	3	4	1.0	0.0
10	719.0	81.66666666666667	4	0.6666666666666666	3	3	0.0	3.0
12	78.66666666666667	7.333333333333333	2	10.0	3	3	1.0	0.0
13	172.0	16.5	4	4.0	2	7	0.0	1.0
14	93.0	6.0	0	0.0	1	7	0.0	3.0

```
[65] encoder = OneHotEncoder(
    inputCols=["platform_index"],
    outputCols=["platform_ohe"]
)
final_features = encoder.fit(final_features).transform(final_features)

[66] final_features = final_features.drop('platform_index')
```

Hình 21: Các đặc trưng sau khi chuyển đổi

Chuẩn hoá các kiểu dữ liệu số như lượt click trung bình, lượt hit trung bình,... sau khi đã được gom nhóm và trích tạo như ở phần trên.

```
numeric_cols = [
    "avg_clicks",
    "avg_hits",
    "total_early_buys",
    "early_avg_price",
    "session_count",
    "max_teamLevel"
]

pre_assembled = VectorAssembler(inputCols=numeric_cols, outputCol="numeric_vector").transform(final_features)

scaler = StandardScaler(inputCol="numeric_vector", outputCol="scaled_numeric", withMean=True, withStd=True)
scaler_model = scaler.fit(pre_assembled)
scaled_df = scaler_model.transform(pre_assembled)
final_assembler = VectorAssembler(
    inputCols=[
        "scaled_numeric",
        "max_teamLevel", "platform_ohe"],
    outputCol="features"
)
final_df = final_assembler.transform(scaled_df)
```

Do dữ liệu bị mất cân bằng, thường thì mọi người chơi đều hướng tới miễn phí hoặc chỉ trả số lượng tiền nhỏ nên ta cần tạo thêm một cột trọng số để cân bằng lại

```
[25] df_with_stats.groupBy('spender_type').count().show()
```

```

+-----+-----+
|spender_type|count|
+-----+-----+
| big_spender|  770|
| low_spender| 3849|
+-----+-----+

```

Hình 22: Tỷ lệ big spender - low spender

```
[90] from pyspark.sql.functions import when, lit
total = 3849 + 770
weight1 = total / (2 * 3849)
weight2 = total / (2 * 770)
train_df = train_df.withColumn(
    "classWeightCol",
    when(train_df["spender_type_index"] == 0, lit(weight1))
    .otherwise(lit(weight2))
)
```

3 Huấn luyện mô hình và đánh giá

3.1 Huấn luyện mô hình

Vì đây là bài toán liên quan đến dữ liệu mất cân bằng, ta cần những mô hình có thể thêm trọng số và đủ mạnh để không bị thiên về bên low spender hay big spender.

```

▶ rf = RandomForestClassifier(
    featuresCol="features",
    labelCol="spender_type_index",
    weightCol="classWeightCol",
    numTrees=90,
    maxDepth=15,
    maxBins=64
)
rf_model = rf.fit(train_df)
train_preds = rf_model.transform(train_df)

acc_eval = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)
train_acc = acc_eval.evaluate(train_preds)

auc_eval = BinaryClassificationEvaluator(
    labelCol="spender_type_index",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)

```

```

train_auc = auc_eval.evaluate(train_preds)

print(f"[Train] Random Forest - Accuracy = {train_acc:.4f}, AUC = {train_auc:.4f}")

test_preds = rf_model.transform(test_df)

acc_eval = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)
test_acc = acc_eval.evaluate(test_preds)

auc_eval = BinaryClassificationEvaluator(
    labelCol="spender_type_index",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)
test_auc = auc_eval.evaluate(test_preds)

print(f"[Test] Random Forest - Accuracy = {test_acc:.4f}, AUC = {test_auc:.4f}")

```

```

➡ [Train] Random Forest - Accuracy = 0.9878, AUC = 0.9993
  [Test] Random Forest - Accuracy = 0.9402, AUC = 0.9699

```

Hình 23: Kết quả và tham số tốt nhất huấn luyện bằng Random Forest

```

▶ from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(
    featuresCol="features",
    labelCol="spender_type_index",
    weightCol="classWeightCol",
    maxIter=100
)

lr_model = lr.fit(train_df)
predictions = lr_model.transform(train_df)
evaluator = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)

accuracy = evaluator.evaluate(predictions)
print(f"Train Accuracy (LogReg) = {accuracy:.4f}")

auc_eval = BinaryClassificationEvaluator(
    labelCol="spender_type_index",
    rawPredictionCol="rawPrediction",
    metricName="areaUnderROC"
)


```

```

auc = auc_eval.evaluate(predictions)
print(f"AUC (LogReg Weighted) = {auc:.4f}")
predictions = lr_model.transform(test_df)
evaluator = MulticlassClassificationEvaluator(
    labelCol="spender_type_index",
    predictionCol="prediction",
    metricName="accuracy"
)

accuracy = evaluator.evaluate(predictions)
print(f"Test Accuracy (LogReg) = {accuracy:.4f}")
auc = auc_eval.evaluate(predictions)
print(f"AUC (LogReg Weighted) = {auc:.4f}")

```


 Train Accuracy (LogReg) = 0.9290
 AUC (LogReg Weighted) = 0.9647
 Test Accuracy (LogReg) = 0.9022
 AUC (LogReg Weighted) = 0.9587

Hình 24: Kết quả và tham số tốt nhất huấn luyện bằng Logistic Regression

```

 gbt = GBTCClassifier(
    featuresCol="features",
    labelCol="spender_type_index",
    weightCol="classWeightCol",
    maxIter=40,
    maxDepth=5,
    stepSize=0.7
)
model = gbt.fit(train_df)
predictions = model.transform(train_df)
accuracy = accuracy_eval.evaluate(predictions)
auc = auc_eval.evaluate(predictions)
print(f"GBT Train - Accuracy = {accuracy:.4f}, AUC = {auc:.4f}")
predictions = model.transform(test_df)
accuracy = accuracy_eval.evaluate(predictions)
auc = auc_eval.evaluate(predictions)
print(f"GBT Test - Accuracy = {accuracy:.4f}, AUC = {auc:.4f}")

```

 GBT Train - Accuracy = 0.9956, AUC = 0.9999
 GBT Test - Accuracy = 0.9348, AUC = 0.9526

Hình 25: Kết quả và tham số tốt nhất huấn luyện bằng Gradient Boosted Trees

3.2 Đánh giá mô hình

Tập train và test được chia theo người chơi, có nghĩa là giữa 2 tập sẽ không có bất kì người chơi nào được trùng lặp. Và tập test cũng sẽ được gom nhóm, chuẩn hoá, trích xuất và chuyển đổi đặc trưng tương tự như tập train.

Mô hình	Accuracy trên tập test	AUC trên tập test
Logistic Regression	0.9022	0.9587
Random Forest	0.9402	0.9699
GBT	0.9348	0.9526

Hình 26: Bảng tổng hợp kết quả trên tập test

Kết luận: Trong ba mô hình được so sánh, Random Forest cho thấy hiệu quả tổng thể vượt trội khi đạt Accuracy cao nhất trên tập kiểm tra (0.9402) đồng thời duy trì AUC ở mức cao (0.9699). Điều này cho thấy mô hình không chỉ dự đoán chính xác với đa số mẫu mà còn phân biệt tốt giữa hai lớp trong bài toán phân loại mất cân bằng. So với Logistic Regression và GBT, Random Forest có tốt nhất về cả độ chính xác và khả năng phân biệt, do đó được xem là mô hình phù hợp nhất cho bài toán dự đoán loại người chơi tiềm năng có thể là big spender