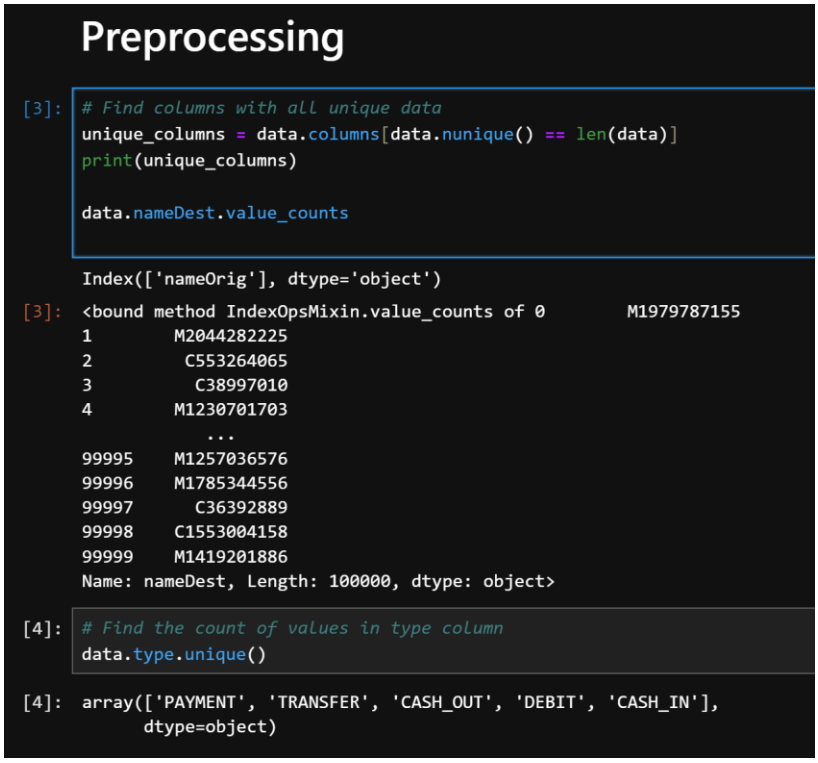
گزارش چالش ها – پروژه درخت تصمیم – هوش مصنوعی – یزدان ماستری فراهانی

در این مرحله چندین چالش موجود بود که برای حل آنها ابتدا به بررسی ستون های داده پرداختم. از آنجایی که ستون داده nameOrig تماما شامل داده های منحصر به فرد بوده و نمیتوان آنها را از هم جدا کرد و یا دسته بندی کرد، از بررسی این ستون در ادامه کار به طور کلی صرف نظر کردم.



Preprocessing

```
[3]: # Find columns with all unique data
unique_columns = data.columns[data.nunique() == len(data)]
print(unique_columns)

data.nameDest.value_counts
```

```
Index(['nameOrig'], dtype='object')
```

```
[3]: <bound method IndexOpsMixin.value_counts of 0        M1979787155
1        M2044282225
2         C553264065
3          C38997010
4        M1230701703
              ...
99995    M1257036576
99996    M1785344556
99997      C36392889
99998    C1553004158
99999    M1419201886
Name: nameDest, Length: 100000, dtype: object>
```

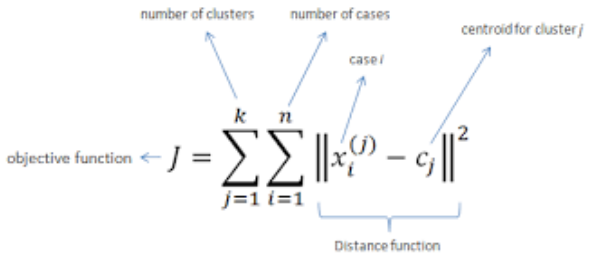
```
[4]: # Find the count of values in type column
data.type.unique()
```

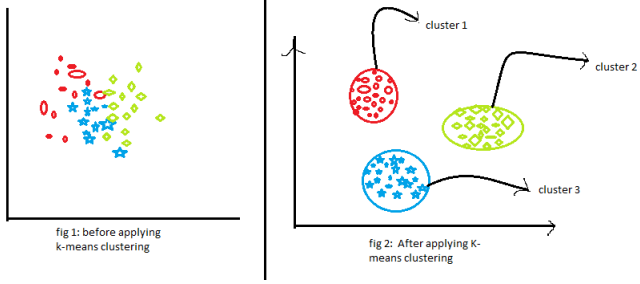
```
[4]: array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

سپس دو دیتای type , nameDest را بر اساس تعداد آنها به اعداد مپ کرده ام.

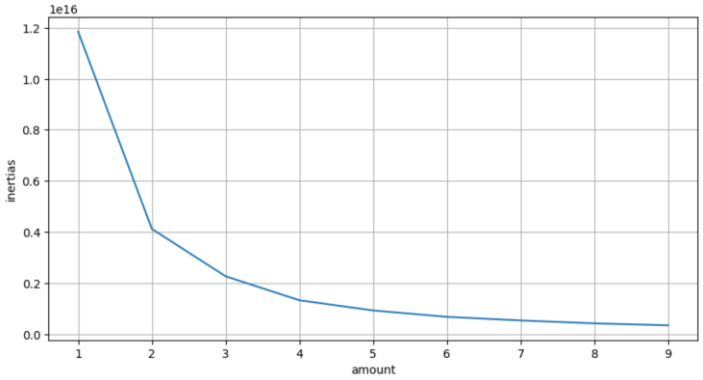
	step	type	amount	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	0	9839.64	170136.0	160296.36	0	0.00	0.00	0
1	1	0	1864.28	21249.0	19384.72	0	0.00	0.00	0
2	1	1	181.00	181.0	0.00	1	0.00	0.00	1
3	1	2	181.00	181.0	0.00	1	21182.00	0.00	1
4	1	0	11668.14	41554.0	29885.86	0	0.00	0.00	0

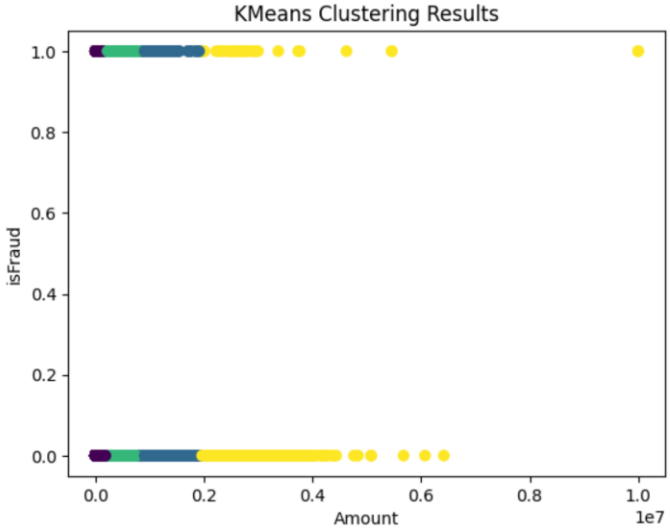
در انتهای این مرحله به گسسته سازی داده های پیوسته پرداخته شده که از انجام راه حل های مرسوم آن صرف نظر کرده و از یک روش جذاب تر به نام k-mean clustering استفاده شده است. این روش به کلاستر بندی داده ها بر حسب پراکندگی و تراکم آنها میپردازد.





با استفاده از این متد ابتدا بهترین تعداد را برای کلاستر ها با توجه به رسم تغییرات با اعداد مختلف برای کلاستر را در نظر گرفته و سپس با توجه به تغییر نکردن محصوص آنها برای دسته بندی های بیشتر از 4 تا، عدد 4 را برای تعداد آنها انتخاب کرده و خروجی آن را با استفاده از scattered نشان دادم.

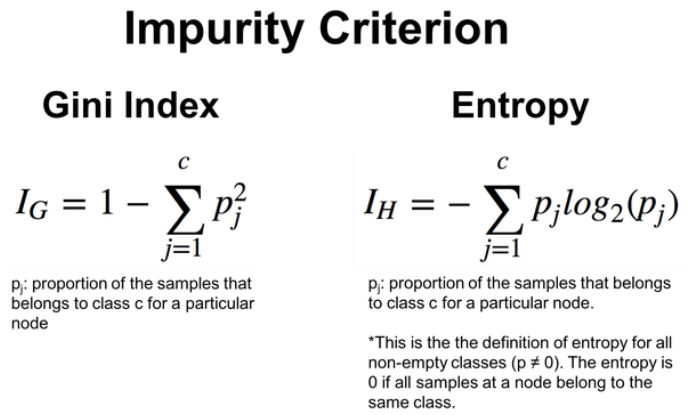




سپس داده ها را به سه دسته train(70%), test(15%), Valid(15%) به صورت کاملا رندم تقسیم کرده تا از آنها در مقایسه میزان دقت درخت استفاده کنیم. (البته در این مورد از valid در هیچ بخش از کد استفاده نشده است)

```
train size: (70000, 9)
test size: (15000, 9)
validation size: (15000, 9)
```

سپس به ساخت درخت پرداخته شده که سخت ترین بخش کار بود. در اینجا ابتدا دو تابع entropy و gini_index برای محاسبه بهترین فیچر برای قرار گیری در ریشه و نود های بالاتر درخت پیاده سازی شده اند و کلاس درخت را نیز به گونه پیاده سازی کردم که با توجه به ورودی که از کاربر گرفته شده، criteria مناسب را در ساخت درخت اعمال کند.



تابع best_Feature در درخت به انتخاب بهترین فیچر بر اساس خروجی های بدست آمده از gini یا entropy هر فیچر میپردازد.

پاسخ سوال تفاوت entropy و gini-index :

1. بازه تغییرات gini-index از 0 تا 0.5 است در صورتی که در entropy از 0 تا 1 میباشد
2. Entropy با توجه به لگاریتمی بودن تابع آن دارای محاسبات پیچیده تری است و به همین دلیل gini-index سریعتر از آن است.

3. پاسخ های entropy دقت بالاتری نسبت به gini-index دارند.

با همه این تفاوت ها از آنجایی که داده ما در اینجا استاندارد نبوده و اکثرا 0 می‌باشند تفاوت چندانی میان درخت خروجی این دو متد وجود نداشته و هر دو میزان دقت 98.12 را محاسبه میکنند.

```
The accuracy of the prediction was 0.9812
```

```
- Feat: step | Info_Gain: 0.5214 | Entropy: 0.0423
    -- Feat: type | Info_Gain: 0.2094 | Entropy: 0.0403
        --- Feat: amount | Info_Gain: 0.0421 | Entropy: 0.0421
            ---- Feat: oldbalanceOrg | Info_Gain: 0.054 | Entropy: 0.0421
                ----- Feat: newbalanceOrig | Info_Gain: 0.043 | Entropy: 0.043
                    ------ 0
                ----- Feat: newbalanceOrig | Info_Gain: 0.0112 | Entropy: 0.011
                    ------ 0
                    ------ 0
        --- Feat: amount | Info_Gain: 0.0742 | Entropy: 0.0375
            ---- Feat: oldbalanceOrg | Info_Gain: 0.1241 | Entropy: 0.0352
                ----- Feat: newbalanceOrig | Info_Gain: 0.0282 | Entropy: 0.0273
                    ------ 0
                    ------ 0
```

در انتها با توجه به شک کردن به یکی شدن پاسخ هر دو متد و غیر منطقی بودن آن از یک کتابخانه آماده پایتون برای ساخت درخت تصمیم و استفاده خودکار از دو روش به طور مستقل پرداخته شده که نشان میدهد تا چهار رقم اعشار هر دو روش دقت یکسان داشته و پاسخ کد ما صحیح است.

```
Accuracy based on Entropy criterion: 0.9796333333333334
Accuracy based on gini criterion: 0.9796333333333334
```

افزایش دقت درخت : برای افزایش دقت درخت بهترین کار هرس کردن آن است. در اینجا نیز با صرف نظر از ویژگی هایی که در ساخت درخت و بهتر شدن آن کمکی به ما نمیکردند مثل nameOrg و یا محدود کردن بیشتر داده ها مثل داده هایی که لیبل شدند و یا انتخاب صحیح نحوه گسسته سازی داده های پیوسته، میتوان به دقت بالاتری رسید. همچنین محدود کردن عمق درخت نیز میتواند موثر باشد که در اینجا از تمامی موارد استفاده شده است.