چکیده :

در این پژوهش هدف ما ایجاد مدل های معین کلاس بندی از جمله svm, decision trees, random forest و ... و امتحان آن ها بر روی دیتاست کلاهبرداری یا خیر درخواست های تصادفات ماشین ها می باشد. در این جریان ما به بررسی دیتا و سپس بالانس کردن آن پرداختیم و پس از آمادی سازی دیتا برای ترین کردن مدل ، خروجی آن ها را با مدل های مختلف بر اساس stratified cross validation ارائه کردیم. تا بهترین مدل مشخص شود . در این پروسه سعی بر بهبود نتیجه مدل ها داشته ایم.

مقدمه :

از بین مدل های سوپروایزد کلاس بندی کدام مدل می تواند برای دیتاست مشخص مورد پژوهش در این مقاله عملکرد بهتری داشته باشد؟ چه روش هایی برای بهبود عملکرد این مدل ها وجود دارد؟ چه روش گزارش دقتی بیشتر مورد اظمینان است؟ در راستای پاسخ به این سوال های مطرح شده ابتدا به بررسی دیتاست پرداختیم که متشکل از 32 فیچر 1 تارگت و 15420 داده بوده. پس از بررسی ارتباط بین فیچر ها و بررسی بالانس بودن و اعمال سه روش مختلف -over sampling , under sampling و SMOTE و اسکیل کردن های فیچر ها مدل های مختلف مطرح شده را بر روی دیتاست فیت کردیم و سپس خروجی را به stratified cross validation با تشکیل پنج دیتاست مختلف و خروجی گرفتن از آن ها عملکرد را مقایسه کردیم. و سعی در بهبود عملکرد آن ها با روش های میزان سازی پارامتر ، تغییر کرنل ، روش های انسمبل و ... داشته ایم.

روش ها :

ابتدا کتابخانه های بیسیک کار را لود می کنیم.

```
Import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

سپس دیتاست vehicle insurance claim detection را در دیتا فریم df لود می کنیم. سپس برای بدست آوردن تعداد فیچر ها و سایز دیتاست از دستور data.shape استفاده می کنیم که خروجی آن بدین شکل می باشد:

```
(15420, 33)
```

این یعنی دیتا ست ما شامل 33 فیچر است که یکی از آن ها target روش سوپروایزد و باقی فیچر هایی هستند که مدل ه بر اساس آن ها ترین می شوند. و سایز دیتا ست هم 15420 می باشد. حال نیاز است که بدانیم فیچر های ما چه هستند و تارگت را از بین آن ها تشخیص دهیم. پس از دستور df.columns استفاده می کنیم:

```
Index(['Month', 'WeekOfMonth', 'DayOfWeek', 'Make', 'AccidentArea',
       'DayOfWeekClaimed', 'MonthClaimed', 'WeekOfMonthClaimed', 'Sex',
       'MaritalStatus', 'Age', 'Fault', 'PolicyType', 'VehicleCategory',
       'VehiclePrice', 'FraudFound_P', 'PolicyNumber', 'RepNumber',
       'Deductible', 'DriverRating', 'Days_Policy_Accident',
       'Days_Policy_Claim', 'PastNumberOfClaims', 'AgeOfVehicle',
       'AgeOfPolicyHolder', 'PoliceReportFiled', 'WitnessPresent',
'AgentType',
       'NumberOfSuppliments', 'AddressChange_Claim', 'NumberOfCars', 'Year',
       'BasePolicy'],
      dtype='object')
```

از بین فیچر های نوشته شده فیچر تارگت همان FraudFound_P می باشد که خروجی آن احتمالا به شکل yes or no می باشد.

برخی فیچر های ما numeric و برخی از جنس object هستند پس در ادامه کار نیاز داریم که دیتارا encode کنیم که همه آن ها به شکل عددی شوند.

	count	mean	std	min	25%	50%	75%	max
WeekOfMonth	15420.0	2.788586	1.287585	1.0	2.00	3.0	4.00	5.0
WeekOfMonthClaimed	15420.0	2.693969	1.259115	1.0	2.00	3.0	4.00	5.0
Age	15420.0	39.855707	13.492377	0.0	31.00	38.0	48.00	80.0
FraudFound_P	15420.0	0.059857	0.237230	0.0	0.00	0.0	0.00	1.0
PolicyNumber	15420.0	7710.500000	4451.514911	1.0	3855.75	7710.5	11565.25	15420.0
RepNumber	15420.0	8.483268	4.599948	1.0	5.00	8.0	12.00	16.0
Deductible	15420.0	407.704280	43.950998	300.0	400.00	400.0	400.00	700.0
DriverRating	15420.0	2.487808	1.119453	1.0	1.00	2.0	3.00	4.0
Year	15420.0	1994.866472	0.803313	1994.0	1994.00	1995.0	1996.00	1996.0

جدول بالا شهود حدودی از توزیع داده های عددی به ما می دهد. تفاوت فاحش رنج اعداد بالا نشان دهنده این است که در ادامه نیاز به اسکیل کردن داده ها داریم. با بررسی مقدار هایی که هر فیچر می تواند داشته باشد نویز را در دو فیچر 'DayOfWeekClaimed' و 'MonthClaimed' شناسایی می کنیم. با دستور:

```
print('DayOfWeekClaimed',df['DayOfWeekClaimed'].unique())
print('MonthClaimed',df['MonthClaimed'].unique())
```

که خروجی مقابل را به ما می دهد:

```
DayOfWeekClaimed ['Tuesday' 'Monday' 'Thursday' 'Friday' 'Wednesday'
'Saturday' 'Sunday'
 '0']
MonthClaimed ['Jan' 'Nov' 'Jul' 'Feb' 'Mar' 'Dec' 'Apr' 'Aug' 'May' 'Jun'
'Sep' 'Oct'
                                                                    '0']
```

مقدار 0 در این داده ها نویز می باشد که نیاز است حذف شوند پس کد مقابل را اجرا می کنیم:

```
df = df.loc[df['DayOfWeekClaimed']!='0']
df = df.loc[df['MonthClaimed']!='0']
```

و حال برای اینکه ببینیم چه تعداد داده حذف شده اند از دستور df.shape استفاده می کنیم. که خروجی آن به شکل زیر است:

```
(15419, 33)
```

این عدد حاکی از این است که فقط یک دیتا بوده که روز هفته و ماه آن صفر وارد شده بود.

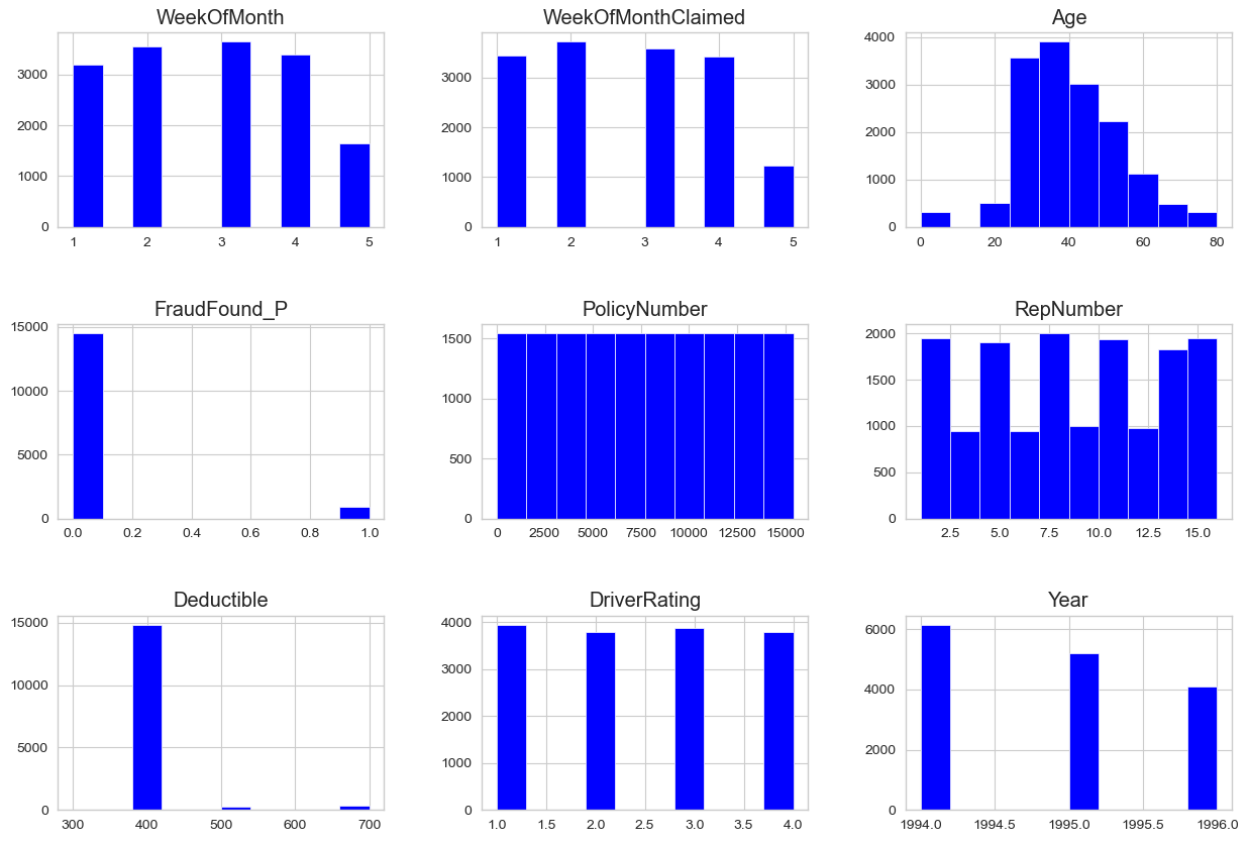
برای یافتن missing value ها دستور df.isna().sum() را اجرا می کنیم. که خروجی آن نشان می هد داده گمشده ای وجود ندارد.

```
Month                     0
WeekOfMonth               0
DayOfWeek                 0
Make                      0
AccidentArea              0
DayOfWeekClaimed          0
MonthClaimed              0
WeekOfMonthClaimed        0
Sex                       0
MaritalStatus             0
Age                       0
Fault                     0
PolicyType                0
VehicleCategory           0
VehiclePrice              0
FraudFound_P              0
PolicyNumber              0
RepNumber                 0
Deductible                0
DriverRating              0
Days_Policy_Accident      0
```

```
Days_Policy_Claim       0
PastNumberOfClaims      0
AgeOfVehicle            0
AgeOfPolicyHolder       0
...
AddressChange_Claim     0
NumberOfCars            0
Year                    0
BasePolicy              0

                        dtype: int64
```

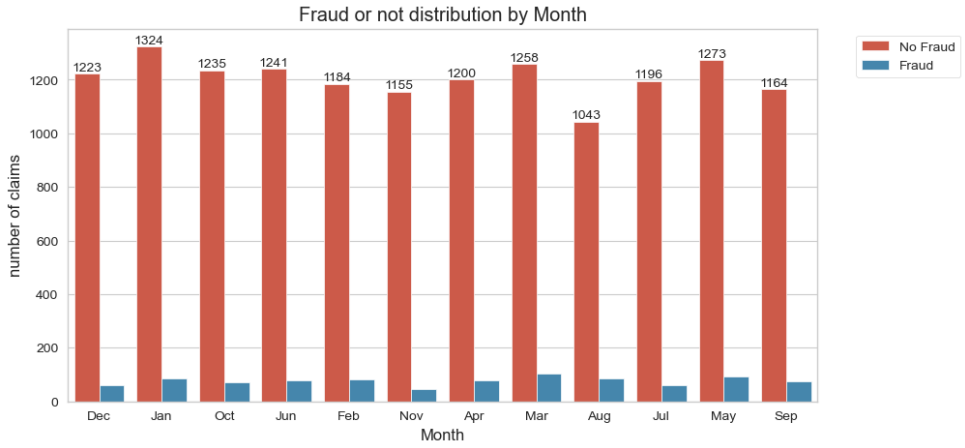
حال برای درک بهتر داده ها به data visualizing می پردازیم.



نمودار های بالا از متغیر های عددی چند چیز را به ما نشان می دهند:

1- نمودار fraudFound_P نشان دهنده ایمبالانس بودن دیتاست که نیاز به بالانس شدن که ادامه دارد.
2- نمودار age نشان دهنده حضور تعدادی داده پرت با توجه به وجود راننده صفر ساله می باشد.
3- در اواسط ماه میزان درخواست برای بیمه تصادفات بیشترین مقدار است.
4- میزان درخواست بیمه تصادفات در بین راننده هایی با rating های مختلف فرق چندانی ندارد.

نمودار های بالا برحسب کل دیتا و فارغ از کلاهبرداری بودن یا نبودن آن ها بوده . برای درک بهتر تفاوت توزیع فیچر ها در کیس های کلاهبرداری یا غیر کلاهبرداری نمودار بهتری را برای فیچر ها رسم می کنیم.



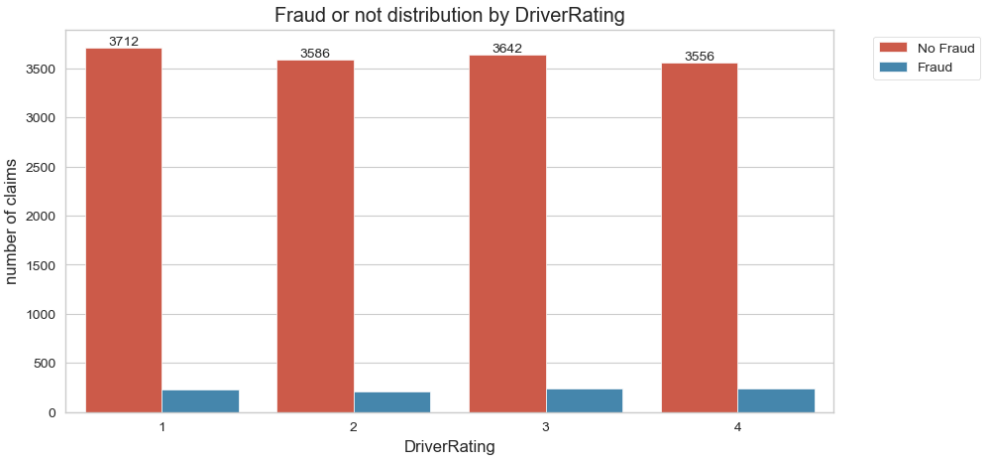
نمودار بالا این توزیع را در ماه های مختلف رسم می کند. چون با دیتای ایم بالانس مواجه هستیم تعداد کیس های کلاهبرداری در هر ماه بسیار کمتر از کیس های غیرکلاهبرداری می باشد. هرچند در ماه های متفاوت مقدار هردوی این ها تفاوت چندانی ندارد اما به طور کلی بیشترین درخواست بیمه برای تصادفات در ماه های jan, mar, may بنظر بیشتر از سایر ماه ها می آید. برای بررسی نرخ کلاهبرداری بودن درخواست کد ratio مربوطه را محاسبه می کنیم.

که خروجی بدین شکل می باشد:

```
Dec 0.04824902723735409
Jan 0.06165839829907867
Oct 0.05363984674329502
Jun 0.06056018168054504
Feb 0.06477093206951026
Nov 0.038301415487094086
Apr 0.0625
Mar 0.075
Aug 0.07453416149068323
Jul 0.04777070063694268
May 0.06876371616678859
Sep 0.06129032258064516
```

مشاهده می کنیم ماه های aug , mar دارای بیشترین نرخ کلاهبرداری هستند.

این نمودار را برای فیچر driver rating نیز رسم می کنیم و نرخ کلاهبرداری را برای آن بدست می آوریم.

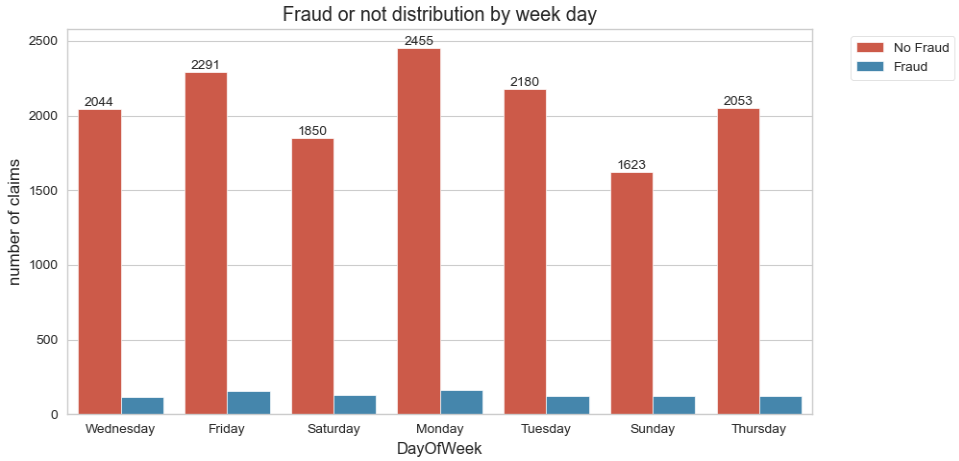



```
1 0.058823529411764705
4 0.06198892112898971
3 0.06230690010298661
2 0.05631578947368421
```

این میزان ratio نشان دهنده این است که کلاهبرداری بودن درخواست بیمه تا حدی با امتیاز راننده رابطه مستقیم دارد.

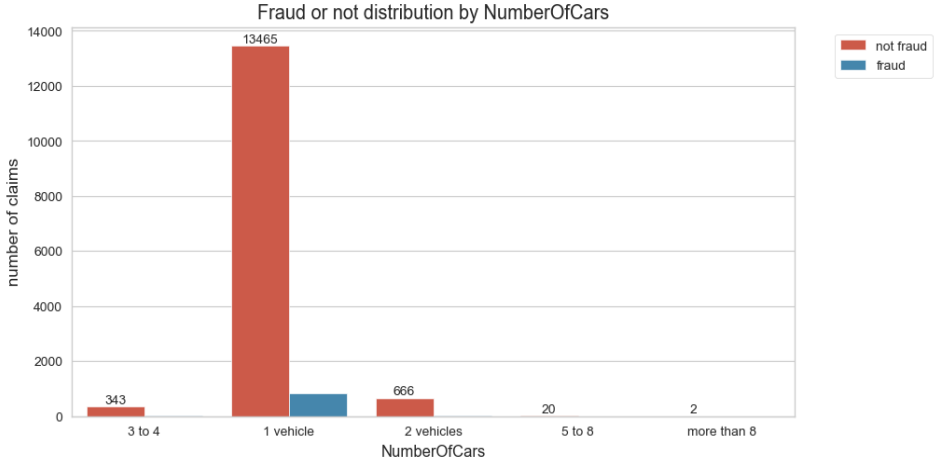
به ترتیب این پروسه را برای WeekOfDay نیز تکرار می کنیم.

```
Wednesday 0.053265400648448355
Friday 0.06298568507157465
Saturday 0.06659939455095863
Monday 0.06118546845124283
Tuesday 0.05217391304347826
Sunday 0.06991404011461318
Thursday 0.05522319374137138
```



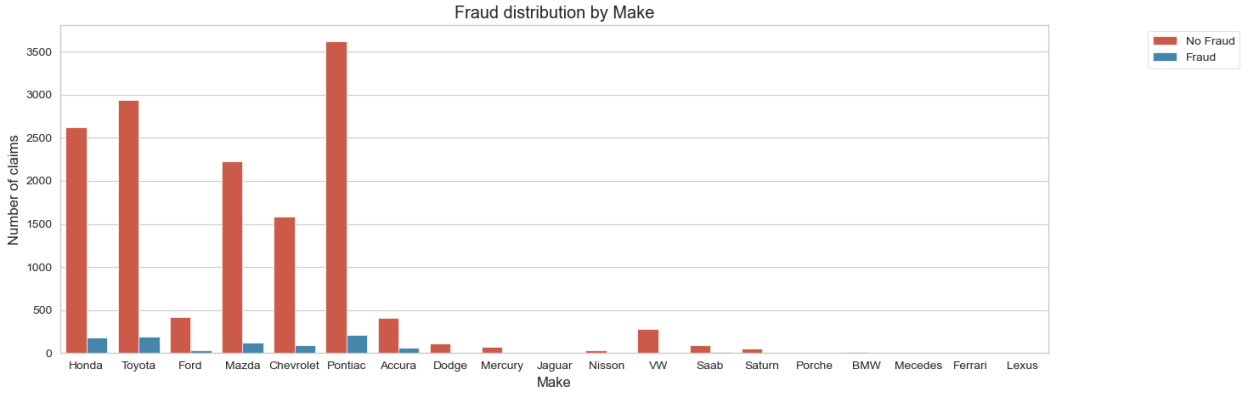
مشاهده می شود میزان بیشتر کلاهبرداری اخرهفته که شامل روزهای Sunday و Saturday می باشد ، بیشتر از باقی روزهاست.

```
3 to 4 0.07795698924731183
1 vehicle 0.059378274537198746
2 vehicles 0.06064880112834979
5 to 8 0.047619047619047616
more than 8 0.0
```



همانطور که می بینیم بیشترین توزیع درخواست ها در تعداد وسط دارایی ماشین قرار دارد اما نرخ کلاهبرداری افرادی که 3 تا 4 ماشین داشته اند بیشتر از سایرین است.

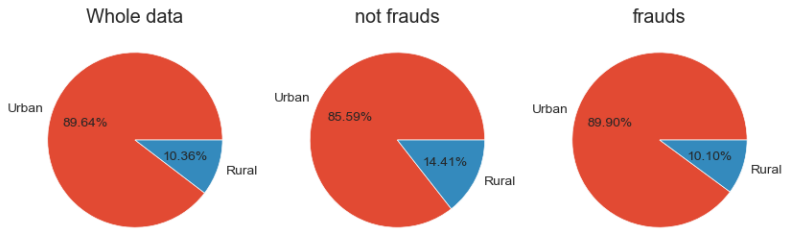
نمودار و نرخ کلاهبرداری را برای مدل ماشین ها بدست می آوریم:



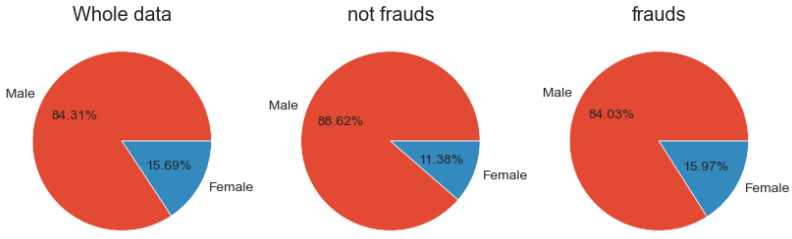
```
Honda 0.06392857142857143
Toyota 0.059596283242550466
Ford 0.07333333333333333
Mazda 0.052251486830926085
Chevrolet 0.05591909577632362
Pontiac 0.05551211884284597
Accura 0.125
Dodge 0.01834862385321101
Mercury 0.07228915662650602
Jaguar 0.0
Nisson 0.03333333333333333
VW 0.028268551236749116
Saab 0.10185185185185185
Saturn 0.10344827586206896
Porche 0.0
BMW 0.06666666666666667
Mecedes 0.25
Ferrari 0.0
Lexus 0.0
```

همانطور که مشاهده می کنیم بیشترین مدل ماشین که دارای ادعای کلاهبرداری بوده با اختلاف مربوط به مدل `saab ,` می باشد.همچنین با توجه به نمودار در می یابیم که مدل های `Saturn` `Honda, Toyota, ford, mazda,` به میزان قابل توجهی درخواست بیمه تصادف بیشتری داشته اند. این می تواند نشانی از مستعدتر بودن این `Chevrolet, potiac` مدل ها برای تصادف باشند.

Distributation by area

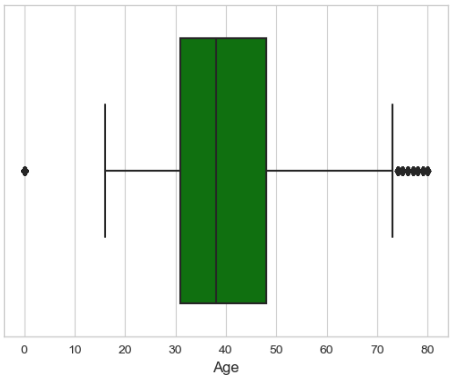


Distributation by sex

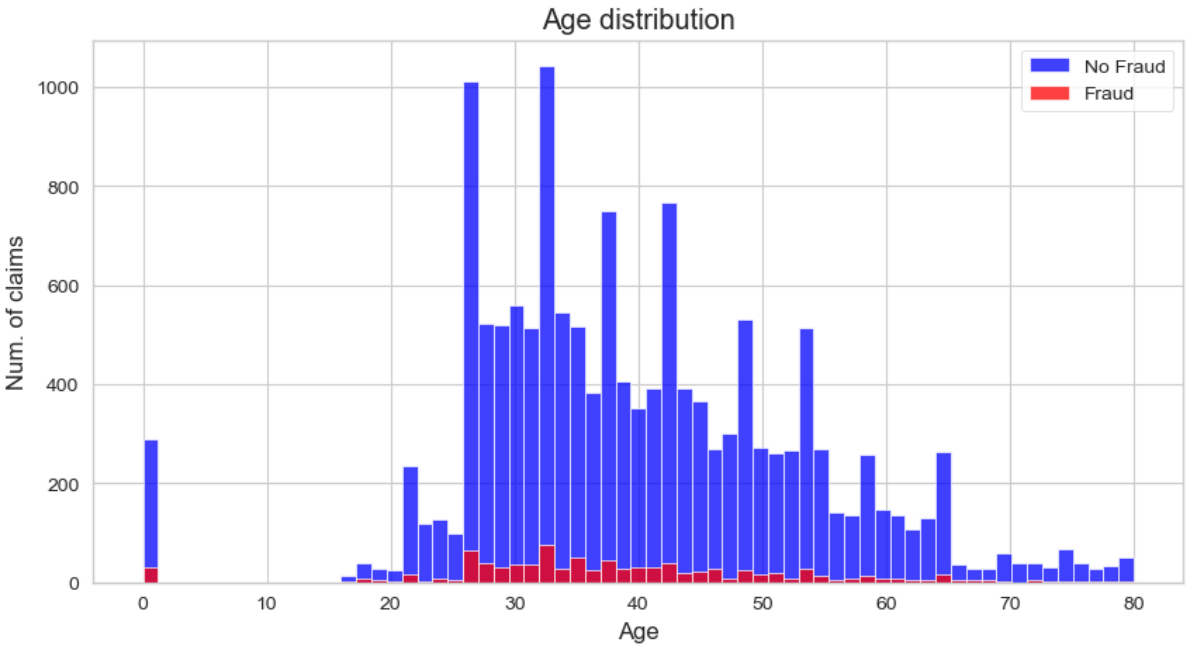


Distributation by WitnessPresent

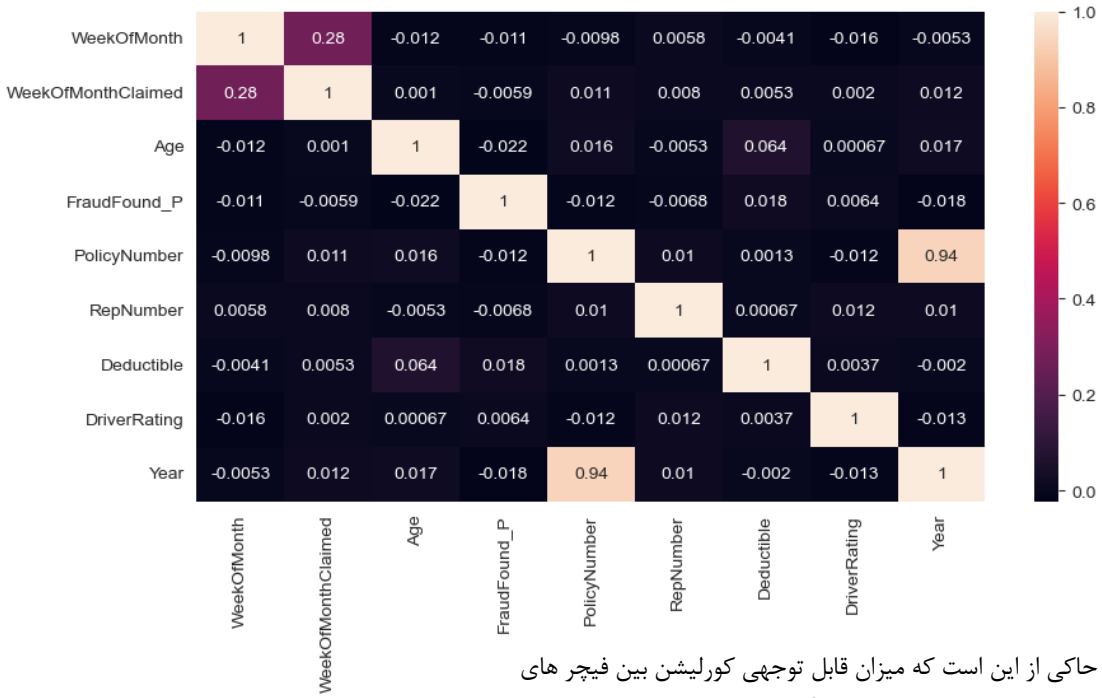
سه نمودار بالا نشان دهنده این توزیع در سه فیچر sex, witnessPresent , area می باشد. به دلیل اینکه این سه فیچر تنها مقادیر باینری اتخاذ می کنند استفاده از نمودار دایره ای انتخاب بهتری برای آنان است.
با مقایسه درخواست ها متوجه میشویم نرخ کلاهبرداری در نقاط شهری ، در جنس زنان و در عدم حضور شاهد بیشتر از حالت دیگر است.
جال پس از مصورسازی دیتا برای یافتن داده های پرتی که در age شناسایی کردیم از boxplot استفاده می کنیم.



نمودار نشان دهنده این است که دارای تعدادی داده پرت در نقطه صفر و بزرگ تر از 72 می باشیم اما به دلیل پیوسته بودن مقادیر بالا 72 از حذف آن ها صرف نظر می کنیم. نمدار histogram فیچر age هم نشانی بر تایید این امر می باشد که در زیر مصور شده است:



این نمودار تاییدیه بر حذف عنصر پرت age = 0 از دیتاست می باشد.
چیزی که در بخش پیش از مدل سازی باقی مانده بررسی این است که آیا فیپر های ما با هم correlation دارند. یا خیر و برای آزمایش این مورد دیتا را مصور سازی می کنیم:



این نمودار حاکی از این است که میزان قابل توجهی کورلیشن بین فیچر های weekOfMonth , weekOfMonthClaimed- Year , PolicyNumber و deduction, age وجود دارد که نیاز است به طور اتومات توسط مدل ها یا با اعمال متد فیچر سلکشنی اصلاح شوند.

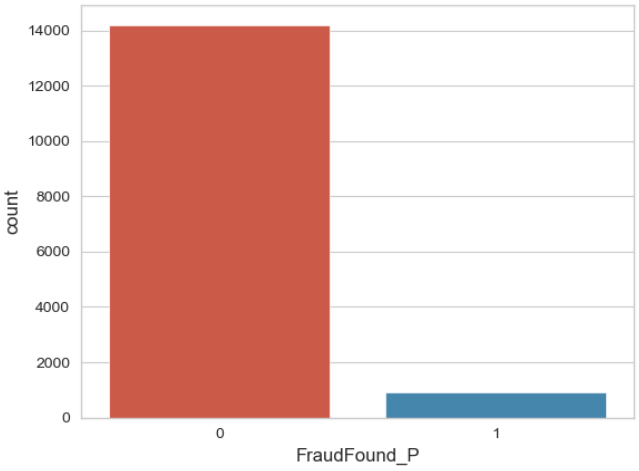
حال برای آماده سازی دیتا قبل از ترین مدل ها نیاز است داده های غیرعددی که پیش تر بیان شد به طور عددی کد گذاری شوند. این فیچر ها به صورت زیر می باشند:

```
Index(['Month', 'DayOfWeek', 'Make', 'AccidentArea', 'DayOfWeekClaimed',
       'MonthClaimed', 'Sex', 'MaritalStatus', 'Fault', 'PolicyType',
       'VehicleCategory', 'VehiclePrice', 'Days_Policy_Accident',
       'Days_Policy_Claim', 'PastNumberOfClaims', 'AgeOfVehicle',
       'AgeOfPolicyHolder', 'PoliceReportFiled', 'WitnessPresent',
'AgentType',
       'NumberOfSuppliments', 'AddressChange_Claim', 'NumberOfCars',
       'BasePolicy'],
                                                      dtype='object')
```

نیاز است که تمامی حالت های این فیچر ها به اولویت ترتیبی کدگذاری شوند. پس از طی کردن مسیر فهمیدن دیتا و حذف نویز از آن نیاز است اصلاحاتی مانند بالانس کردن ، اسکیل کردن و فیچر سلکشن بر روی دیتا اعمال شود.

با چک کردن وضعیت ایم بالانسی دیتا شروع می کنیم:

```
0    14208
1      892
Name: FraudFound_P, dtype: int64
```



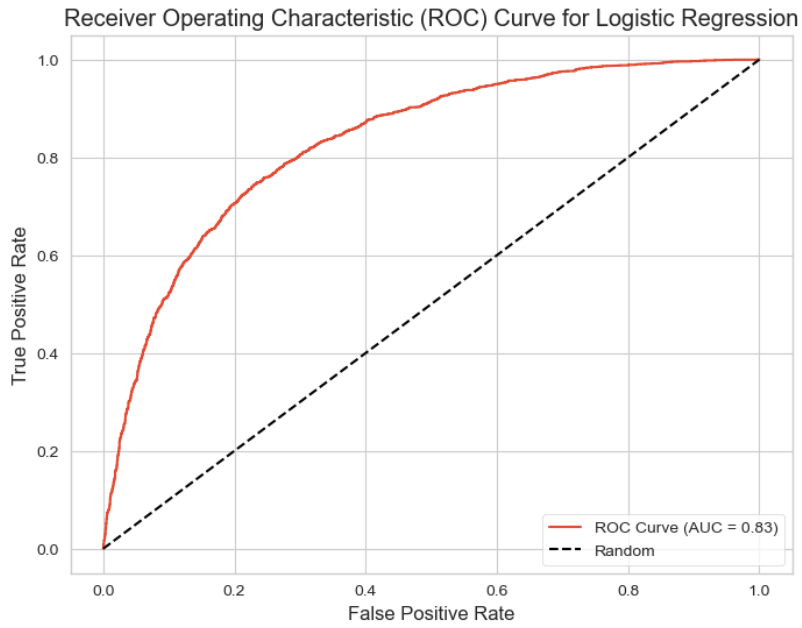
ارقام و نمودار مقابل به ایم بالانس بودن دیتا اشاره دارند.
از جمله روش هایی که برای حل این مشکل می توان به کار برد : Over sampling, under sampling , SMOTE می باشد
که روش اورسمپلینگ به دلیل زیاد کردن دیتا بر اساس بخش هایی از آن ممکن است دچار اورفیت و روش اندرسمپلینگ به دلیل کاهش حجم دیتا ممکن است دچار اندرفیت شود پس در ابتدا روشی که حدس می زنیم عملکرد بهتری نشان دهد یعنی SMOTE را اعمال می کنیم و سپس خروجی را با دو تکنیک دیگر هم آزمایش می کنیم تا به مقایسه آن ها بپردازیم.
پیش از اعمال مدل باید آن را اسکیل کنیم و فیچر های نامطلوب را کنار بگذاریم برای اسکیل کردن از روش min max استفاده می کنیم و برای فیچر سلکشن select_kbest را استفاده می کنیم که 10 فیچر مرتبط تر را برگزینند.
حال در بخش مدل سازی نیاز است که مدل های logistic regression , svm, decision tree, random forest, KNN, naïve bayes را برروی دیتاست پیاده سازی کنیم. خروجی را برحسب سه معیار : 1- accuracy score برای هر مدل و 2- stratified cross validation 3- classification report برای هر مدل تخمین می زنیم.
Classification report شامل موارد زیر می باشد:
precision-1 : TP / (TP + FP)
Recall-2 : TP / (TP + FN)
F1-score-3 : 2 * (Precision * Recall) / (Precision + Recall)
Support-4

خروجی برای logistic regression :

```
logistic regression accuracy : 0.7533427163969036
Cross-validation scores: [0.80031668 0.80010558 0.80714411 0.8023931
0.80151329]
Mean accuracy: 0.8022945515861906
[[2137  738]
 [ 664 2145]]
              precision    recall  f1-score   support

           0       0.76      0.74      0.75      2875
           1       0.74      0.76      0.75      2809

    accuracy                           0.75      5684
   macro avg       0.75      0.75      0.75      5684
weighted avg       0.75      0.75      0.75      5684
```

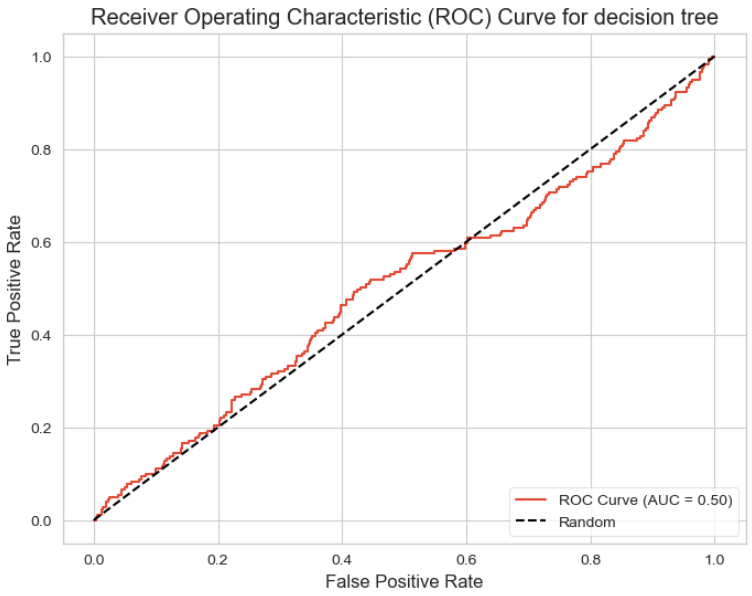


خروجی برای SVM :

```
SVM Acuuracy : 0.9400662251655629
Cross-validation scores: [0.51055595 0.51205349 0.50413514 0.51486891
0.52736231]
Mean accuracy: 0.5137951590376029
[[2839    0]
 [ 181    0]]
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      2839
           1       0.00      0.00      0.00       181

    accuracy                           0.94      3020
   macro avg       0.47      0.50      0.48      3020
weighted avg       0.88      0.94      0.91      3020
```

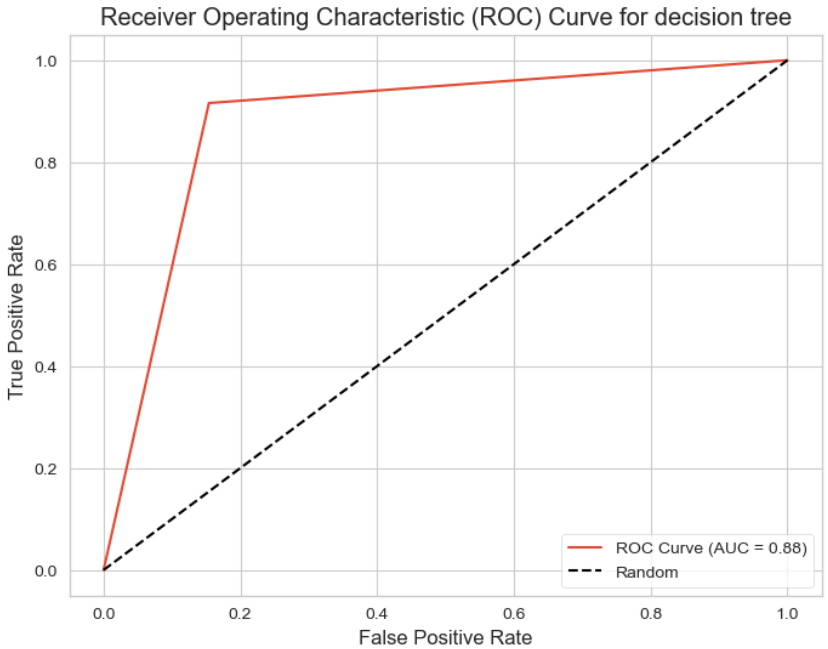


خروجی برای decision tree:

```
Desicion Tree Accuracy: 0.8805418719211823
Cross-validation scores: [0.92241379 0.92064051 0.92574345 0.9211684
0.91342601]
Mean accuracy: 0.9206784299210593
[[2432  443]
 [ 236 2573]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88      2875
           1       0.85      0.92      0.88      2809

    accuracy                           0.88      5684
   macro avg       0.88      0.88      0.88      5684
weighted avg       0.88      0.88      0.88      5684
```

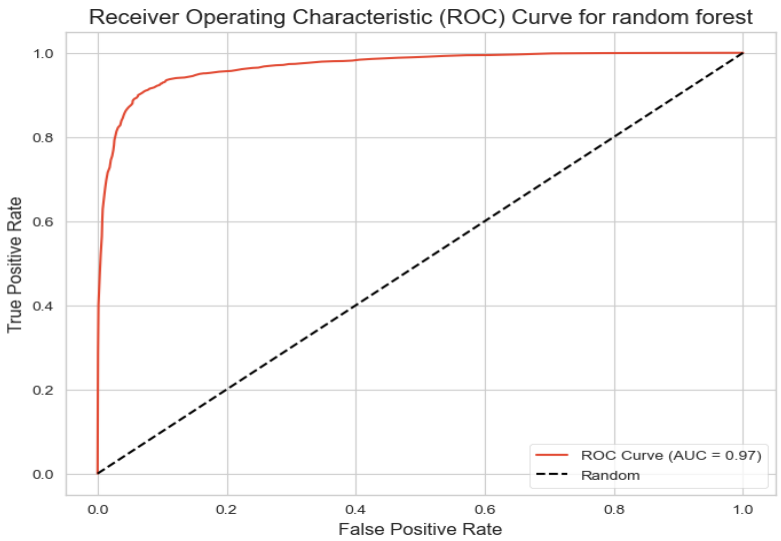


خروجی برای random forest:

```
Random Forest Accuracy: 0.9139690358902182
Cross-validation scores: [0.95935961 0.95882456 0.96040824 0.95548126
0.95460144]
Mean accuracy: 0.9577350216573672
[[2585  290]
 [ 199 2610]]
              precision    recall  f1-score   support

           0       0.93      0.90      0.91      2875
           1       0.90      0.93      0.91      2809

    accuracy                           0.91      5684
   macro avg       0.91      0.91      0.91      5684
weighted avg       0.91      0.91      0.91      5684
```

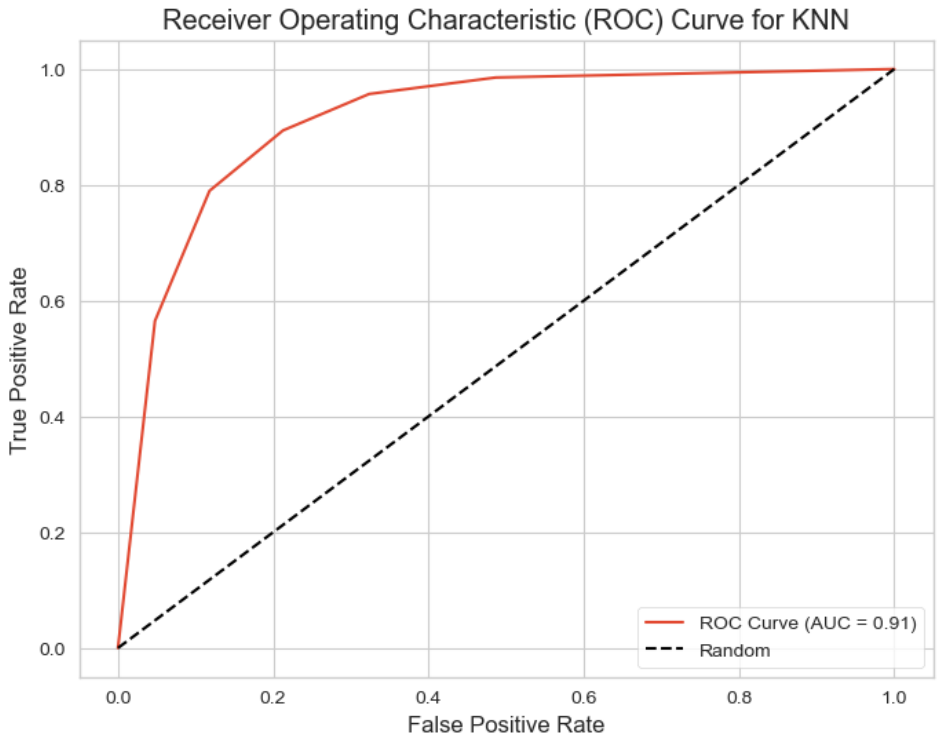


خروجی برای KNN :

```
KNN Accuracy: 0.8400774102744546
Cross-validation scores: [0.82125264 0.83283477 0.82086926 0.82298082
0.83054725]
Mean accuracy: 0.8256969469421437
[[2264  611]
 [ 298 2511]]
              precision    recall  f1-score   support

           0       0.88      0.79      0.83      2875
           1       0.80      0.89      0.85      2809

    accuracy                           0.84      5684
   macro avg       0.84      0.84      0.84      5684
weighted avg       0.84      0.84      0.84      5684
```

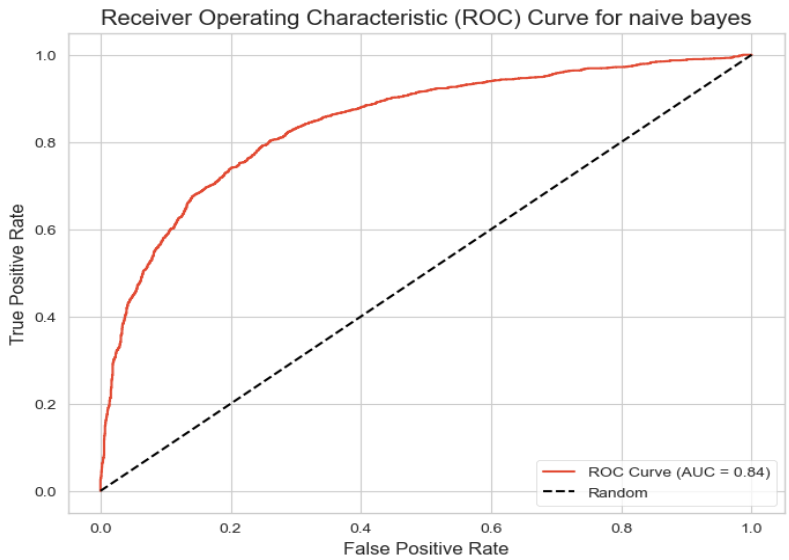


خروجی برای naïve bayes :

```
Naive Bayes Accuracy: 0.7536945812807881
Cross-validation scores: [0.76724138 0.75910611 0.7754707  0.76244941
0.75998592]
Mean accuracy: 0.7648507041569836
[[1869 1006]
 [ 394 2415]]
              precision    recall  f1-score   support

           0       0.83      0.65      0.73      2875
           1       0.71      0.86      0.78      2809

    accuracy                           0.75      5684
   macro avg       0.77      0.75      0.75      5684
weighted avg       0.77      0.75      0.75      5684
```



تا اینجا بهترین عملکرد مربوط به رندوم فورست بوده است . حال متد over sampling را برای بالانس کردن امتحان می کنیم و خروجی را گزارش می کنیم:

```
logistic regression accuracy : 0.6857615894039735
Cross-validation scores: [0.72185576 0.73548813 0.72229551 0.74268749
0.73784913]
Mean accuracy: 0.73203520389699
[[1933  906]
 [  43  138]]
              precision    recall  f1-score   support

           0       0.98      0.68      0.80      2839
           1       0.13      0.76      0.23       181

    accuracy                           0.69      3020
   macro avg       0.56      0.72      0.51      3020
weighted avg       0.93      0.69      0.77      3020


Desicion Tree Accuracy: 0.9231788079470199
Cross-validation scores: [0.97955145 0.97955145 0.98043096 0.97668793
0.97954695]
 Mean accuracy: 0.9791537482357142
[[2741   98]
 [ 134   47]]
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      2839
           1       0.32      0.26      0.29       181

    accuracy                           0.92      3020
   macro avg       0.64      0.61      0.62      3020
weighted avg       0.92      0.92      0.92      3020



Random Forest Accuracy: 0.9427152317880795
Cross-validation scores: [0.99912049 0.99824099 0.99890062 0.9982406
0.99868045]
Mean accuracy: 0.998636628014373
[[2836    3]
 [ 170   11]]
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      2839
           1       0.79      0.06      0.11       181
```

```
    accuracy                           0.94      3020
   macro avg       0.86      0.53      0.54      3020
weighted avg       0.93      0.94      0.92      3020


KNN Accuracy: 0.8125827814569536
Cross-validation scores: [0.91248901 0.91600704 0.9173263  0.91400924
0.91466901]
Mean accuracy: 0.9149001177770183
[[2407  432]
 [ 134   47]]
              precision    recall  f1-score   support

           0       0.95      0.85      0.89      2839
           1       0.10      0.26      0.14       181

    accuracy                           0.81      3020
   macro avg       0.52      0.55      0.52      3020
weighted avg       0.90      0.81      0.85      3020

Naive Bayes Accuracy: 0.6738410596026491
Cross-validation scores: [0.70140721 0.69942832 0.70294635 0.69870244
0.70815923]
Mean accuracy: 0.7021287098358414
[[1903  936]
 [  49  132]]
              precision    recall  f1-score   support

           0       0.97      0.67      0.79      2839
           1       0.12      0.73      0.21       181

    accuracy                           0.67      3020
   macro avg       0.55      0.70      0.50      3020
weighted avg       0.92      0.67      0.76      3020
```

حال خروجی را برای مدل های اعمال شده پس از under sampling گزارش می کنیم:

```
logistic regression accuracy : 0.6682119205298013
Cross-validation scores: [0.70877193 0.71578947 0.68309859 0.6971831
0.71478873]
Mean accuracy: 0.7039263652087966
[[1878  961]
 [  41  140]]
              precision    recall  f1-score   support

           0       0.98      0.66      0.79      2839
```

```
           1       0.13      0.77      0.22       181

    accuracy                           0.67      3020
   macro avg       0.55      0.72      0.50      3020
weighted avg       0.93      0.67      0.76      3020

Desicion Tree Accuracy: 0.7685430463576159
Cross-validation scores: [0.70175439 0.78947368 0.6971831  0.75704225
0.76408451]
Mean accuracy: 0.7419075858660735
[[2190  649]
 [  50  131]]
              precision    recall  f1-score   support

           0       0.98      0.77      0.86      2839
           1       0.17      0.72      0.27       181
    accuracy                           0.77      3020
   macro avg       0.57      0.75      0.57      3020
weighted avg       0.93      0.77      0.83      3020

Random Forest Accuracy: 0.7254966887417219
Cross-validation scores: [0.80350877 0.7754386  0.78169014 0.79225352
0.80985915]
Mean accuracy: 0.7925500370644922
[[2030  809]
 [  20  161]]
              precision    recall  f1-score   support

           0       0.99      0.72      0.83      2839
           1       0.17      0.89      0.28       181

    accuracy                           0.73      3020
   macro avg       0.58      0.80      0.56      3020
weighted avg       0.94      0.73      0.80      3020

KNN Accuracy: 0.5350993377483444
Cross-validation scores: [0.50877193 0.54035088 0.47887324 0.48943662
0.51056338]
Mean accuracy: 0.5055992092908327
[[1524 1315]
 [  89   92]]
              precision    recall  f1-score   support

           0       0.94      0.54      0.68      2839
           1       0.07      0.51      0.12       181

    accuracy                           0.54      3020
   macro avg       0.51      0.52      0.40      3020
weighted avg       0.89      0.54      0.65      3020
```

```
Naive Bayes Accuracy: 0.6751655629139073
Cross-validation scores: [0.68070175 0.69473684 0.65140845 0.68661972
0.67253521]
Mean accuracy: 0.6772003953545835
[[1915  924]
 [  57  124]]
              precision    recall  f1-score   support

           0       0.97      0.67      0.80      2839
           1       0.12      0.69      0.20       181

    accuracy                           0.68      3020
   macro avg       0.54      0.68      0.50      3020
weighted avg       0.92      0.68      0.76      3020
```

	Logistic regression	svm	Decision tree	Random forest	KNN	BayesNaïve
SMOTE	0.8022	0.5137	0.9206	0.9577	0.8256	0.7648
Over sampling	0.7320	0.5094	0.9791	0.9986	0.9149	0.7021
Under sampling	0.7039	0.4003	0.7425	0.7915	0.5055	0.6772

تحلیل خروجی ها و جدول:

مشاهده کردیم که در هر سه روش بهترین عملکرد مربوط به رندوم فورست بوده و بدترین عملکرد مربوط به svm و بهترین عملکرد با بهترین متد مربوط به oversample کردن با رندوم فورست که دقت 99.86 درصد دارد. و به ترتیب اورسمپلینگ – SMOTE و سپس اندرفیت عملکرد خوبی در بالانس سازی داشته اند.

حال برای بهبود عملکرد دو مدلی که بهترین و بدترین خروجی را داشتند اقدام به بهبود رندوم فورست و svm می کنیم در ابتدا برای بهبود عملکرد رندوم فورست تعداد درخت های آن را به 500 افزایش می دهیم. در حالت دیفالت کتابخانه scikit-learn از 100 درخت استفاده می کند.

```
Cross-validation scores: [0.99890062 0.99890062 0.99890062 0.99780075
0.99934022]
Mean accuracy: 0.9987685638070392
[[2837    2]
 [ 174    7]]
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      2839
           1       0.78      0.04      0.07       181

    accuracy                           0.94      3020
   macro avg       0.86      0.52      0.52      3020
weighted avg       0.93      0.94      0.92      3020
```

مشاهده می کنیم مقدار 1 ده هزارم عملکرد مدل بهبود یافته است.

سپس هایپر پارامتر درخت را با استفاده از tune gridsearch می کنیم:

```
Random Forest Accuracy: 0.9417218543046357
```

مشاهده می شود چون از اکورسی اسکور برای گزارش استفاده کرده ایم مقدار کمی ئز این مقیاس گزارش دهی مدل بهتر شده.

و سپس با استفاده از متد ensamble از adaboost استفاده می کنیم:

```
Cross-validation scores: [0.99890062 0.99934037 0.99912049 0.99868045
0.99934022]
Mean accuracy: 0.9990764301087498
[[2838    1]
 [ 180    1]]
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      2839
           1       0.50      0.01      0.01       181

    accuracy                           0.94      3020
   macro avg       0.72      0.50      0.49      3020
weighted avg       0.91      0.94      0.91      3020
```

بدین شکل مشاهده تاثیر چشمگیر در بهبود عملکرد رندوم فورست می باشیم .

حال برای بهبود مدل svm ابتدا از تون کردن هایپرپارامتر توسط گریدسرچ استفاده می کنیم. دقت کنید که چون ران تایم اجرایی svm زیاد است از اکورسی اسکور برای گزارش استفاده می کنیم.

```
Accuracy: 0.2867549668874172
```

از بین مولفه های تعریفی ما برای گرید سرچ که راجب کرنل poly, rbf و عمق کم بود تاثیر چندادن مثبتی از اعمال این گرید سرچ بر روی دل نگرفتیم . حال دستی کرنل دیگری را امتحان می کنیم که نتیجه را بررسی کنیم.

```
Accuracy: 0.48841059602649006
```

نتیجه مقابل پس از اعمال کرنل سیگموید بر روی مدل بدست آمده که حاکی از مطلوب تر بودن آن می باشد.هرچند که بین مدل هایی که تا اینجا امتحان کردیم بدترین عملکرد مربوط به svm بوده.

نتیجه :

در این پژوهش با دیتای بشدت غیر بالانسی مواجه بودیم که پس از اعمال سه متد بالانس سازی به ترتیب عملکرد بهتر را در over sampling سپس SMOTE و سپس under sampling شاهد بودیم. و از بین کلسیفایر های مختلف رندوم فورست با 99.8 درصد بهترین عملکرد را داشته که پس از اعمال روش انسمبل ادابوست بر روی آن این مقدار به 99.9 افزایش یافته.بعد از آن دسیژن تری ها و بعد آن KNN عملکرد بهتر مدل ها بوده اند. از تفاوت زیادی که در برخی مدل ها بین اکورسی اسکور و کراس ولیدیشن شاهد بودیم میتوان حدس زد که اکورسی اسکور خیلی معیار مناسبی نمی باشد. بدترین عملکرد هم کلسیفایر svm با به تقریب 50 درصد درستی داشته که بسیار نزدیک به رندومنس می باشد که این خلاف انتظار هم نیست با توجه به این که این کلسیفایر برای دیتاهایی با حجم زیاد عملکرد مناسبی ندارد.

	Logistic regression	svm	Decision tree	Random forest	KNN	BayesNaïve
SMOTE	0.8022	0.5137	0.9206	0.9577	0.8256	0.7648
Over sampling	0.7320	0.5094	0.9791	0.9990	0.9149	0.7021
Under sampling	0.7039	0.4800	0.7425	0.7915	0.5055	0.6772

جدول بالا حاصل تغییر مقایر بهبود یافته و نهایی الگوریتم ها می باشد.