Kaggle Competition: 대출 위험도 예측 모델 개발

은행에서 대출 대상자를 예측하는 알고리즘

실제 은행 데이터를 사용하여 대출을 받으려는 사람이 상환을 할 수 있을지 없을지를 예측

- 문제 해결
  - 대출 대상자를 예측하는 알고리즘 개발
- 목표 변수 : SeriousDlqin2yrs
  - 최근 2년 동안 90일 이상 연체한 적이 있는지 여부
  - 값: 1 (연체한 적 있음), 0 (연체한 적 없음)
- 알고리즘 결과
  - 일정 기간(2년) 내에 채무 불이행 여부
- 평가 지표
  - roc_auc 점수

데이터 읽기 및 탐색

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(1)

lender_data = pd.read_csv('cs-training.csv', index_col=0)
lender_data.head()
```

Out[5]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90I
1	1	0.766127	45	2	0.802982	9120.0	13	
2	0	0.957151	40	0	0.121876	2600.0	4	
3	0	0.658180	38	1	0.085113	3042.0	2	
4	0	0.233810	30	0	0.036050	3300.0	5	
5	0	0.907239	49	1	0.024926	63588.0	7	

In [2]:
```
lender_data.tail()
```

Out[2]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfT
149996	0	0.040674	74	0	0.225131	2100.0	4	
149997	0	0.299745	44	0	0.716562	5584.0	4	
149998	0	0.246044	58	0	3870.000000	NaN	18	
149999	0	0.000000	30	0	0.000000	5716.0	4	
150000	0	0.850283	64	0	0.249908	8158.0	8	

In [3]:
```
print(lender_data.shape)
lender_data.info()
```

```
(150000, 11)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150000 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                      150000 non-null int64
RevolvingUtilizationOfUnsecuredLines  150000 non-null float64
age                                   150000 non-null int64
NumberOfTime30-59DaysPastDueNotWorse  150000 non-null int64
DebtRatio                             150000 non-null float64
MonthlyIncome                         120269 non-null float64
NumberOfOpenCreditLinesAndLoans       150000 non-null int64
NumberOfTimes90DaysLlate              150000 non-null int64
NumberRealEstateLoansOrLines          150000 non-null int64
NumberOfTime60-89DaysPastDueNotWorse  150000 non-null int64
NumberOfDependents                    146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 13.7 MB
```

In [4]:
```
lender_data.describe()
```

Out[4]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans
count	150000.000000	150000.000000	150000.000000	150000.000000	150000.000000	1.202690e+05	150000.000000
mean	0.066840	6.048438	52.295207	0.421033	353.005076	6.670221e+03	8.452760
std	0.249746	249.755371	14.771866	4.192781	2037.818523	1.438467e+04	5.145951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000
25%	0.000000	0.029867	41.000000	0.000000	0.175074	3.400000e+03	5.000000
50%	0.000000	0.154181	52.000000	0.000000	0.366508	5.400000e+03	8.000000
75%	0.000000	0.559046	63.000000	0.000000	0.868254	8.249000e+03	11.000000
max	1.000000	50708.000000	109.000000	98.000000	329664.000000	3.008750e+06	58.000000

In [47]:
```
lender_data.corr()
```

Out[47]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpen
SeriousDlqin2yrs	1.000000	-0.001802	-0.115386	0.125587	-0.007602	-0.019746	
RevolvingUtilizationOfUnsecuredLines	-0.001802	1.000000	-0.005898	-0.001314	0.003961	0.007124	
age	-0.115386	-0.005898	1.000000	-0.062995	0.024188	0.037717	
NumberOfTime30-59DaysPastDueNotWorse	0.125587	-0.001314	-0.062995	1.000000	-0.006542	-0.010217	
DebtRatio	-0.007602	0.003961	0.024188	-0.006542	1.000000	-0.028712	
MonthlyIncome	-0.019746	0.007124	0.037717	-0.010217	-0.028712	1.000000	
NumberOfOpenCreditLinesAndLoans	-0.029669	-0.011281	0.147705	-0.055312	0.049565	0.091455	
NumberOfTimes90DaysLate	0.117175	-0.001061	-0.061005	0.983603	-0.008320	-0.012743	
NumberRealEstateLoansOrLines	-0.007038	0.006235	0.033150	-0.030565	0.120046	0.124959	
NumberOfTime60-89DaysPastDueNotWorse	0.102261	-0.001048	-0.057159	0.987005	-0.007533	-0.011116	
NumberOfDependents	0.046048	0.001557	-0.213303	-0.002680	-0.040673	0.062647	

In [ ]:
```
#추가로, 뭐가 중요한 영향력있는 컬럼인지도 파악?
```

In [ ]:
```

```

데이터 전처리

In [6]:
```
lender_data1 = lender_data
lender_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150000 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                      150000 non-null int64
RevolvingUtilizationOfUnsecuredLines  150000 non-null float64
age                                   150000 non-null int64
NumberOfTime30-59DaysPastDueNotWorse  150000 non-null int64
DebtRatio                             150000 non-null float64
MonthlyIncome                         120269 non-null float64
NumberOfOpenCreditLinesAndLoans       150000 non-null int64
NumberOfTimes90DaysLate               150000 non-null int64
NumberRealEstateLoansOrLines          150000 non-null int64
NumberOfTime60-89DaysPastDueNotWorse  150000 non-null int64
NumberOfDependents                    146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 13.7 MB
```

In [44]:
```
# MonthlyIncome 결측치 처리

print(lender_data1['MonthlyIncome'].max()) #income 최대값
print(lender_data1['MonthlyIncome'].min()) #income 최소값
print(lender_data1['MonthlyIncome'].mean()) #income 평균
print(lender_data1['MonthlyIncome'].median()) #income 중앙값
```

```
3008750.0
0.0
6670.221237392844
5400.0
```

In [45]:
```
lender_data1.sort_values(by='MonthlyIncome', ascending=False).head()
```

Out[45]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTin
73764	0	0.007328	52	0	0.001470	3008750.0	10	
137141	0	0.000000	68	0	0.002776	1794060.0	15	
111366	0	0.163856	44	0	0.004013	1560100.0	12	
50641	0	0.469025	44	1	0.004537	1072500.0	9	
122544	0	0.041599	55	0	0.000147	835040.0	8	

In [46]:
```
# MonthlyIncome 평균 및 중앙값 조회
lender_data1.groupby('SeriousDlqin2yrs')['MonthlyIncome'].mean()
```

Out[46]:
```
SeriousDlqin2yrs
0    6747.837774
1    5630.826493
Name: MonthlyIncome, dtype: float64
```

In [47]:
```
lender_data1.groupby('SeriousDlqin2yrs')['MonthlyIncome'].median()
```

Out[47]:
```
SeriousDlqin2yrs
0    5466.0
1    4500.0
Name: MonthlyIncome, dtype: float64
```

mean(평균)으로 결측치 대체

In [7]:
```
# 연체한적 있는지 여부인 0, 1 로 gruopby하여 dataframe의 형태를 유지하며 0,1의 mean 값으로 이루어진 새 df을 만들고
me = lender_data1.groupby('SeriousDlqin2yrs')['MonthlyIncome'].transform(np.mean)
me
```

Out[7]:
```
1          5630.826493
2          6747.837774
3          6747.837774
4          6747.837774
5          6747.837774
6          6747.837774
7          6747.837774
8          6747.837774
9          6747.837774
10         6747.837774
11         6747.837774
12         6747.837774
13         6747.837774
14         5630.826493
15         6747.837774
16         6747.837774
17         6747.837774
18         6747.837774
19         6747.837774
20         6747.837774
21         6747.837774
22         5630.826493
23         6747.837774
24         6747.837774
25         6747.837774
26         5630.826493
27         6747.837774
28         6747.837774
29         6747.837774
30         6747.837774
              ...
149971     6747.837774
149972     6747.837774
149973     6747.837774
149974     6747.837774
149975     6747.837774
149976     6747.837774
149977     6747.837774
149978     6747.837774
149979     6747.837774
149980     5630.826493
149981     6747.837774
149982     6747.837774
149983     6747.837774
149984     6747.837774
149985     6747.837774
149986     6747.837774
149987     6747.837774
149988     6747.837774
149989     6747.837774
149990     6747.837774
149991     6747.837774
149992     6747.837774
149993     6747.837774
149994     6747.837774
149995     6747.837774
149996     6747.837774
149997     6747.837774
149998     6747.837774
149999     6747.837774
150000     6747.837774
Name: MonthlyIncome, Length: 150000, dtype: float64
```

In [8]:
```
#결측치(nan) 대체/변환
lender_data1['MonthlyIncome'] = lender_data1['MonthlyIncome'].fillna(me)
lender_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150000 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                      150000 non-null int64
RevolvingUtilizationOfUnsecuredLines  150000 non-null float64
age                                   150000 non-null int64
NumberOfTime30-59DaysPastDueNotWorse  150000 non-null int64
DebtRatio                             150000 non-null float64
MonthlyIncome                         150000 non-null float64
NumberOfOpenCreditLinesAndLoans       150000 non-null int64
NumberOfTimes90DaysLate               150000 non-null int64
NumberRealEstateLoansOrLines          150000 non-null int64
NumberOfTime60-89DaysPastDueNotWorse  150000 non-null int64
NumberOfDependents                    146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 13.7 MB
```

In [50]:
```
#NumberOfDependents 결측치 처리

print(lender_data1['NumberOfDependents'].max()) #income 최대값
print(lender_data1['NumberOfDependents'].min()) #income 최소값
print(lender_data1['NumberOfDependents'].mean()) #income 평균
print(lender_data1['NumberOfDependents'].median()) #income 중앙값
```

```
20.0
0.0
0.7572222678605657
0.0
```

In [9]:
```
# NumberOfDependents 결측치 제거 -- 결측치인 행 수가 적기에 그냥 제거

lender_data1 = lender_data1.dropna()
lender_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146076 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                      146076 non-null int64
RevolvingUtilizationOfUnsecuredLines  146076 non-null float64
age                                   146076 non-null int64
NumberOfTime30-59DaysPastDueNotWorse  146076 non-null int64
DebtRatio                             146076 non-null float64
MonthlyIncome                         146076 non-null float64
NumberOfOpenCreditLinesAndLoans       146076 non-null int64
NumberOfTimes90DaysLate               146076 non-null int64
NumberRealEstateLoansOrLines          146076 non-null int64
NumberOfTime60-89DaysPastDueNotWorse  146076 non-null int64
NumberOfDependents                    146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 13.4 MB
```

In [ ]:
```
```

머신러닝 -- 예측,검증 및 평가 수행

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import Binarizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, recall_score, precision_score, f1_score, accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, roc_auc_score
```

In [10]:
```
# input, output (X, y) 나누기
y = lender_data1['SeriousDlqin2yrs']
X = lender_data1.drop(columns=['SeriousDlqin2yrs'])
```

In [58]:
```
X.head()
```

Out[58]:

	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	Number
1	0.766127	45	2	0.802982	9120.0	13	0	
2	0.957151	40	0	0.121876	2600.0	4	0	
3	0.658180	38	1	0.085113	3042.0	2	1	
4	0.233810	30	0	0.036050	3300.0	5	0	
5	0.907239	49	1	0.024926	63588.0	7	0	

In [59]:
```
y.head()
```

Out[59]:
```
1    1
2    0
3    0
4    0
5    0
Name: SeriousDlqin2yrs, dtype: int64
```

In [60]:
```
np.unique(y, return_counts=True) # target에서 1이 총 몇 명인지 확인
```

Out[60]:
```
(array([0, 1], dtype=int64), array([136229,   9847], dtype=int64))
```

In [11]:
```
# 훈련데이터, 테스트데이터 분리
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[11]:
```
((109557, 10), (36519, 10), (109557,), (36519,))
```

In [62]:
```
np.unique(y_train, return_counts=True)
```

Out[62]:
```
(array([0, 1], dtype=int64), array([102170,   7387], dtype=int64))
```

In [ ]:
```

```

In [68]:
```
#DecisionTreeClassifier() -- 결정트리 모델
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
pred_train = tree.predict(X_train)
pred_test = tree.predict(X_test)
```

In [74]:
```
print(confusion_matrix(y_train, pred_train))
print(confusion_matrix(y_test, pred_test))
```

```
[[102169      1]
 [     0   7387]]
[[32350  1709]
 [ 1518   942]]
```

In [69]:
```
# accuracy_score()로 위의 결정트리 모델 평가
print(accuracy_score(y_train, pred_train))
print(accuracy_score(y_test, pred_test))
```

```
0.9999908723312979
0.9116350392946138
```

In [76]:
```
# Accuracy, Recall, Precision, F1-Score 동시 조회
print(classification_report(y_test, pred_test, target_names=['0', '1']))
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.95     34059
           1       0.36      0.38      0.37      2460

    accuracy                           0.91     36519
   macro avg       0.66      0.67      0.66     36519
weighted avg       0.91      0.91      0.91     36519
```

In [ ]:
```

```

In [77]:
```
# y 실제값과 예측값을 받아서 각 지표를 출력하는 함수
def print_metrics(y, pred, title=None):
    acc = accuracy_score(y, pred)
    recall = recall_score(y, pred)
    precision = precision_score(y, pred)
    f1 = f1_score(y, pred)
    if title:
        print(title)
    print(f'정확도:{acc}, 재현율:{recall}, 정밀도:{precision}, f1점수:{f1}')
```

In [80]:
```
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
pred_tree_train = tree.predict(X_train)
pred_tree_test = tree.predict(X_test)
print_metrics(y_train, pred_tree_train, '결정나무 - train')
print_metrics(y_test, pred_tree_test, '결정나무 - test')
```

```
결정나무 - train
정확도:0.9999908723312979, 재현율:1.0, 정밀도:0.9998646453708717, f1점수:0.9999323181049069
결정나무 - test
정확도:0.9113612092335497, 재현율:0.3853658536585366, 정밀도:0.3546576879910213, f1점수:0.36937463471654003
```

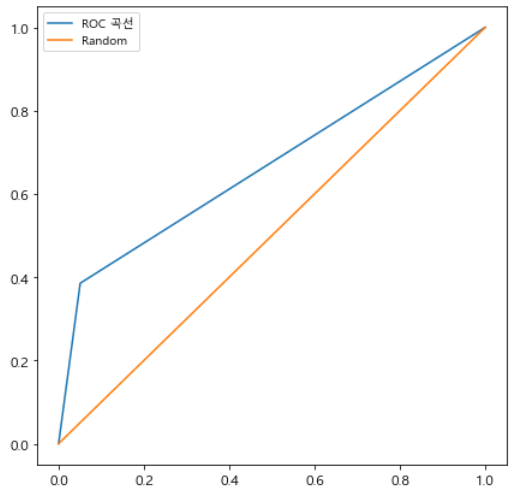
In [ ]:
```
# 여기선 실제 Positive 데이터를 Negative로 잘못 판단하면 업무상 큰 영향이 있는 경우이기에 재현율이 중요
# FN(False Negative: Negative라고 예측 했지만 실제는 Positive)를 낮추는데 초점
```

In [88]:
```
# ROC 곡선 및 AUC 확인

pred_proba_tree = tree.predict_proba(X_test)
pred_proba_tree.shape
```

Out[88]:
```
(36519, 2)
```

In [97]:
```
positive_proba_tree = pred_proba_tree[:, 1]
```

In [100]:
```
# ROC 곡선
fprs, tprs, th = roc_curve(y_test, positive_proba_tree)

plt.figure(figsize=(7,7))
plt.plot(fprs, tprs, label='ROC 곡선')
plt.plot([0,1],[0,1], label='Random')
plt.legend()
plt.show()
```



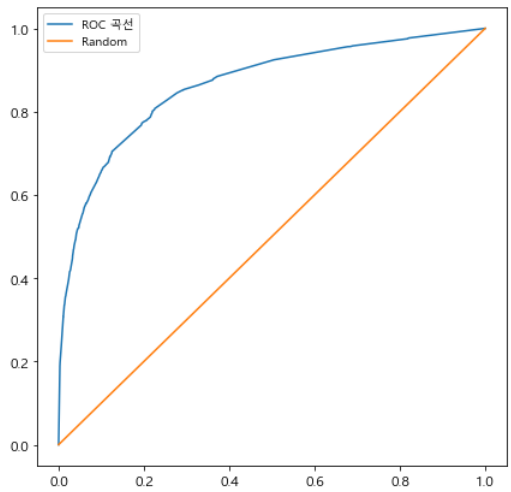
In [102]:
```
# AUC 조회
auc = roc_auc_score(y_test, pred_proba_tree[:, 1])
auc
```

Out[102]:
```
0.6673592238432733
```

In [181]:
```
# DecisionTree 복잡도 제어
y = lender_data1['SeriousDlqin2yrs']
X = lender_data1.drop(columns=['SeriousDlqin2yrs'])

X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

tree = DecisionTreeClassifier(max_depth = 10)
tree.fit(X_train, y_train)
pred_tree_train = tree.predict(X_train)
pred_tree_test = tree.predict(X_test)
print_metrics(y_train, pred_tree_train, '결정나무 - train')
print_metrics(y_test, pred_tree_test, '결정나무 - test')
```

```
결정나무 - train
정확도:0.9513586534863131, 재현율:0.38477143629970245, 정밀도:0.7848275862068965, f1점수:0.5163807968055176
결정나무 - test
정확도:0.9432624113475178, 재현율:0.32001630656339175, 정밀도:0.6602186711522288, f1점수:0.43108182317408017
```

In [182]:
```
pred_proba_tree = tree.predict_proba(X_test)
positive_proba_tree = pred_proba_tree[:, 1]

fprs, tprs, th = roc_curve(y_test, positive_proba_tree)

plt.figure(figsize=(7,7))
plt.plot(fprs, tprs, label='ROC 곡선')
plt.plot([0,1],[0,1], label='Random')
plt.legend()
plt.show()
```



In [183]:
```
auc = roc_auc_score(y_test, pred_proba_tree[:, 1])
auc
```

Out[183]:
```
0.8644041174335836
```

In [ ]:

이번엔 median(중앙값)으로 결측치 대체 및 결정트리 실행

In [105]:
```
lender_data2 = lender_data
```

In [106]:
```
lender_data2.groupby('SeriousDlqin2yrs')['MonthlyIncome'].median()
```

Out[106]:
```
SeriousDlqin2yrs
0    6690.0
1    5240.0
Name: MonthlyIncome, dtype: float64
```

In [107]:
```
med = lender_data2.groupby('SeriousDlqin2yrs')['MonthlyIncome'].transform(np.median)
med
```

Out[107]:
```
1         5240.0
2         6690.0
3         6690.0
4         6690.0
5         6690.0
6         6690.0
7         6690.0
8         6690.0
9         6690.0
10        6690.0
11        6690.0
12        6690.0
13        6690.0
14        5240.0
15        6690.0
16        6690.0
17        6690.0
18        6690.0
19        6690.0
20        6690.0
21        6690.0
22        5240.0
23        6690.0
24        6690.0
25        6690.0
26        5240.0
27        6690.0
28        6690.0
29        6690.0
30        6690.0
           ...
149971    6690.0
149972    6690.0
149973    6690.0
149974    6690.0
149975    6690.0
149976    6690.0
149977    6690.0
149978    6690.0
149979    6690.0
149980    5240.0
149981    6690.0
149982    6690.0
149983    6690.0
149984    6690.0
149985    6690.0
149986    6690.0
149987    6690.0
149988    6690.0
149989    6690.0
149990    6690.0
149991    6690.0
149992    6690.0
149993    6690.0
149994    6690.0
149995    6690.0
149996    6690.0
149997    6690.0
149998    6690.0
149999    6690.0
150000    6690.0
Name: MonthlyIncome, Length: 150000, dtype: float64
```

In [108]:
```
lender_data2['MonthlyIncome'] = lender_data2['MonthlyIncome'].fillna(med)
lender_data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150000 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                      150000 non-null int64
RevolvingUtilizationOfUnsecuredLines  150000 non-null float64
age                                   150000 non-null int64
NumberOfTime30-59DaysPastDueNotWorse  150000 non-null int64
DebtRatio                             150000 non-null float64
MonthlyIncome                         150000 non-null float64
NumberOfOpenCreditLinesAndLoans       150000 non-null int64
NumberOfTimes90DaysLate               150000 non-null int64
NumberRealEstateLoansOrLines          150000 non-null int64
NumberOfTime60-89DaysPastDueNotWorse  150000 non-null int64
NumberOfDependents                    146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 13.7 MB
```

In [109]:
```
lender_data2 = lender_data2.dropna()
lender_data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146076 entries, 1 to 150000
Data columns (total 11 columns):
SeriousDlqin2yrs                      146076 non-null int64
RevolvingUtilizationOfUnsecuredLines  146076 non-null float64
age                                   146076 non-null int64
NumberOfTime30-59DaysPastDueNotWorse  146076 non-null int64
DebtRatio                             146076 non-null float64
MonthlyIncome                         146076 non-null float64
NumberOfOpenCreditLinesAndLoans       146076 non-null int64
NumberOfTimes90DaysLate               146076 non-null int64
NumberRealEstateLoansOrLines          146076 non-null int64
NumberOfTime60-89DaysPastDueNotWorse  146076 non-null int64
NumberOfDependents                    146076 non-null float64
dtypes: float64(4), int64(7)
memory usage: 13.4 MB
```

In [110]:
```
# input, output (X, y) 나누기
y = lender_data2['SeriousDlqin2yrs']
X = lender_data2.drop(columns=['SeriousDlqin2yrs'])
```

In [111]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

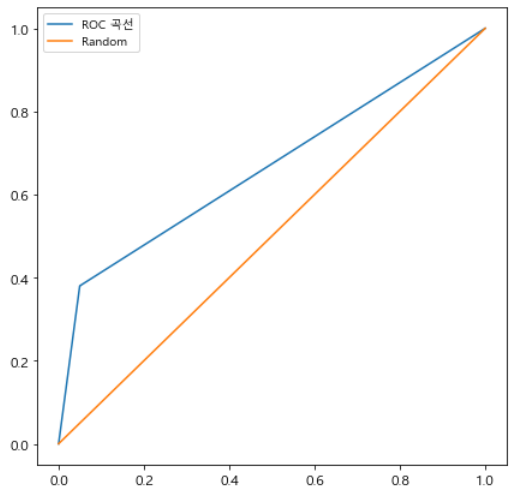
Out[111]:
```
((109557, 10), (36519, 10), (109557,), (36519,))
```

In [112]:
```
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
pred_tree_train = tree.predict(X_train)
pred_tree_test = tree.predict(X_test)
print_metrics(y_train, pred_tree_train, '결정나무 - train')
print_metrics(y_test, pred_tree_test, '결정나무 - test')
```

```
결정나무 - train
정확도:0.9999817446625957, 재현율:0.9997293273785357, 정밀도:1.0, f1점수:0.9998646453708717
결정나무 - test
정확도:0.9120731673923164, 재현율:0.37957689178193654, 정밀도:0.35624284077892326, f1점수:0.36753988575930674
```

In [116]:
```
pred_proba_tree = tree.predict_proba(X_test)
positive_proba_tree = pred_proba_tree[:, 1]
```

In [117]:
```
fprs, tprs, th = roc_curve(y_test, positive_proba_tree)

plt.figure(figsize=(7,7))
plt.plot(fprs, tprs, label='ROC 곡선')
plt.plot([0,1],[0,1], label='Random')
plt.legend()
plt.show()
```



In [118]:
```
auc = roc_auc_score(y_test, pred_proba_tree[:, 1])
auc
```

Out[118]:
```
0.6652138773949547
```

In [ ]:
```
# mean(평균)으로 결측치를 대체했을 때가 결과가 조금 더 좋다
# mean(평균)으로 결측치 제거한 것을 데이터셋으로 사용
```

In [ ]:
```

```

다른 분류 모델들 사용

In [121]:
```
y = lender_data1['SeriousDlqin2yrs']
X = lender_data1.drop(columns=['SeriousDlqin2yrs'])

X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[121]:
```
((109557, 10), (36519, 10), (109557,), (36519,))
```

K-최근접 이웃 (K-Nearest Neighbors, K-NN)

In [195]:
```
# 기존 데이터로 예측
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

pred_knn_train = knn.predict(X_train)
pred_knn_test = knn.predict(X_test)

print_metrics(y_train, pred_knn_train, 'KNN - train')
print_metrics(y_test, pred_knn_test, 'KNN - test')
```

```
KNN - train
정확도:0.9502907162481631, 재현율:0.30267784690289423, 정밀도:0.8852848101265823, f1점수:0.4511187260632937
KNN - test
정확도:0.9381691722117254, 재현율:0.18222584590297594, 정밀도:0.6394849785407726, f1점수:0.2836294416243655
```

In [197]:
```
pred_proba_knn = knn.predict_proba(X_test)
positive_proba_knn = pred_proba_knn[:, 1]
auc = roc_auc_score(y_test, positive_proba_knn)
auc
```

Out[197]:
```
0.6317673153542933
```

In [198]:
```
# Standard Scaling (표준화)

std_scaler = StandardScaler()
std_scaler.fit(X_train)
X_train_std_scaled = std_scaler.transform(X_train)
X_test_std_scaled = std_scaler.transform(X_test)
```

In [200]:
```
# Standard Sclaing된 데이터로 K-최근접 이웃 (K-Nearest Neighbors, K-NN) 사용

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_std_scaled, y_train)

pred_knn_train = knn.predict(X_train_std_scaled)
pred_knn_test = knn.predict(X_test_std_scaled)

print_metrics(y_train, pred_knn_train, 'KNN - train')
print_metrics(y_test, pred_knn_test, 'KNN - test')

pred_proba_knn = knn.predict_proba(X_test_std_scaled)
positive_proba_knn = pred_proba_knn[:, 1]
auc = roc_auc_score(y_test, positive_proba_knn)
auc
```

```
KNN - train
정확도:0.9489398212802468, 재현율:0.3490668109277793, 정밀도:0.7676977989292088, f1점수:0.4799181851989587
KNN - test
정확도:0.9272981187874805, 재현율:0.1842641663269466, 정밀도:0.40867992766726946, f1점수:0.2540039336892385
```

Out[200]:
```
0.6678754023657442
```

In [201]:
```
# MinMaxScaling

mm_scaler = MinMaxScaler()
mm_scaler.fit(X_train)

X_train_mm_scaled = mm_scaler.transform(X_train)
X_test_mm_scaled = mm_scaler.transform(X_test)
```

In [202]:
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_mm_scaled, y_train)

pred_knn_train = knn.predict(X_train_mm_scaled)
pred_knn_test = knn.predict(X_test_mm_scaled)

print_metrics(y_train, pred_knn_train, 'KNN - train')
print_metrics(y_test, pred_knn_test, 'KNN - test')

pred_proba_knn = knn.predict_proba(X_test_mm_scaled)
positive_proba_knn = pred_proba_knn[:, 1]
auc = roc_auc_score(y_test, positive_proba_knn)
auc
```

```
KNN - train
정확도:0.9479357777230117, 재현율:0.3149851230727617, 정밀도:0.7847035040431267, f1점수:0.4495271183169272
KNN - test
정확도:0.9280922259645664, 재현율:0.14757439869547492, 정밀도:0.4035674470457079, f1점수:0.21611940298507462
```

Out[202]:
```
0.6493558976868217
```

In [ ]:
```
# MinMaxScaling 보다 Standard Scaling로 전처리 한 것이 결과가 더 좋다
```

Support Vector Machine(SVM)

In [ ]:
```
# 너무 오래걸려 보류
```

In [ ]:
```
#기존 데이터로 예측
svc = SVC(C=0.1, gamma=0.1, probability=True) #Support Vector Classification(SVC) 객체 생성
svc.fit(X_train, y_train) #학습
pred_svc_train = svc.predict(X_train) #예측
pred_svc_test = svc.predict(X_test)

print_metrics(y_train, pred_svc_train, 'SVC - train')
print_metrics(y_test, pred_svc_test, 'SVC - test')

pred_proba_svc = svc.predict_proba(X_test)
positive_proba_svc = pred_proba_svc[:, 1]
auc = roc_auc_score(y_test, positive_proba_svc)
auc
```

In [ ]:
```
param = {
    'kernel':['linear','rbf'],
    'C':[0.01,0.1,1,10],
    'gamma':[0.01,0.1,1,10]
}

svc = SVC()
from sklearn.model_selection import GridSearchCV
g_search = GridSearchCV(svc
                        , param
                        , scoring=['accuracy','precision','recall','f1','roc_auc']
                        , cv=3
                        , n_jobs=-1)
```

In [ ]:
```
# Standard Scaling
std_scaler = StandardScaler()
std_scaler.fit(X_train)
X_train_std_scaled = std_scaler.transform(X_train)
X_test_std_scaled = std_scaler.transform(X_test)
```

In [ ]:
```
# MinMaxScaling
mm_scaler = MinMaxScaler()
mm_scaler.fit(X_train)

X_train_mm_scaled = mm_scaler.transform(X_train)
X_test_mm_scaled = mm_scaler.transform(X_test)
```

결정트리 GridSearchCV로 최적의 파라미터 찾기

In [ ]:
```
#최적의 파라미터 for DecisionTreeCalssifier()
```

In [8]:
```
from sklearn.model_selection import GridSearchCV
param={
    'max_depth':range(1,11),
    'min_samples_leaf':range(1,51)
}

g_search = GridSearchCV(DecisionTreeClassifier(),
                        param,
                        cv=5,
                        n_jobs=-1,
                        scoring=['accuracy','precision','recall','f1','roc_auc'],
                        refit = 'accuracy')
```

In [9]:
```
g_search.fit(X, y)
```

Out[9]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
                                              splitter='best'),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': range(1, 11),
                         'min_samples_leaf': range(1, 51)},
             pre_dispatch='2*n_jobs', refit='accuracy',
             return_train_score=False,
             scoring=['accuracy', 'precision', 'recall', 'f1', 'roc_auc'],
             verbose=0)
```

In [10]:
```
g_search.best_params_
```

Out[10]:
```
{'max_depth': 9, 'min_samples_leaf': 37}
```

In [14]:
```
result_gridsearch = pd.DataFrame(g_search.cv_results_)
result_gridsearch.head()
```

Out[14]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_leaf	params	split0_test_accuracy	split1_test_accuracy
0	0.156214	0.000002	0.078105	0.000001	1	1	{'max_depth': 1, 'min_samples_leaf': 1}	0.932571	0.932571
1	0.159336	0.006247	0.093320	0.009788	1	2	{'max_depth': 1, 'min_samples_leaf': 2}	0.932571	0.932571
2	0.164732	0.008450	0.096341	0.008263	1	3	{'max_depth': 1, 'min_samples_leaf': 3}	0.932571	0.932571
3	0.164287	0.008091	0.093434	0.009213	1	4	{'max_depth': 1, 'min_samples_leaf': 4}	0.932571	0.932571
4	0.164968	0.014438	0.085406	0.013778	1	5	{'max_depth': 1, 'min_samples_leaf': 5}	0.932571	0.932571

5 rows × 47 columns

In [15]:
```
best_tree_model = g_search.best_estimator_
best_tree_model
```

Out[15]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=9,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=37, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [16]:
```
impt_features = best_tree_model.feature_importances_
impt_features
```

Out[16]:
```
array([0.11404679, 0.00742521, 0.07513269, 0.13402033, 0.22577304,
       0.01068378, 0.37428079, 0.00254442, 0.05459854, 0.00149442])
```

In [17]:
```
pd.Series(impt_features, index = X.columns).sort_values()
```

Out[17]:
```
NumberOfDependents                      0.001494
NumberRealEstateLoansOrLines            0.002544
age                                     0.007425
NumberOfOpenCreditLinesAndLoans         0.010684
NumberOfTime60-89DaysPastDueNotWorse    0.054599
NumberOfTime30-59DaysPastDueNotWorse    0.075133
RevolvingUtilizationOfUnsecuredLines    0.114047
DebtRatio                               0.134020
MonthlyIncome                           0.225773
NumberOfTimes90DaysLate                 0.374281
dtype: float64
```

In [10]:
```
#최적의 파라미터 적용
tree = DecisionTreeClassifier(max_depth=9, min_samples_leaf=37)
tree.fit(X_train, y_train)
pred_tree_train = tree.predict(X_train)
pred_tree_test = tree.predict(X_test)

print_metrics(y_train, pred_tree_train, '결정나무 - train')
print_metrics(y_test, pred_tree_test, '결정나무 - test')

pred_proba_tree = tree.predict_proba(X_test)
positive_proba_tree = pred_proba_tree[:, 1]
auc = roc_auc_score(y_test, positive_proba_tree)
auc
```

```
결정나무 - train
정확도:0.9455260731856476, 재현율:0.30036679798940363, 정밀도:0.7299438758666227, f1점수:0.42560153994225214
결정나무 - test
정확도:0.9429064322681344, 재현율:0.29927594529364443, 정밀도:0.6844526218951242, f1점수:0.4164567590260286
```

Out[10]:
```
0.8831485939573248
```

Random Forest (랜덤포레스트)

In [6]:
```
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of n_estimators wi
ll change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[6]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [11]:
```
pred_rf_train = rf_clf.predict(X_train)
pred_rf_test = rf_clf.predict(X_test)

print_metrics(y_train, pred_rf_train, '랜덤포레스트 - train')
print_metrics(y_test, pred_rf_test, '랜덤포레스트 - test')

pred_proba_rf = rf_clf.predict_proba(X_test)
positive_proba_rf = pred_proba_rf[:, 1]
auc = roc_auc_score(y_test, positive_proba_rf)
auc
```

```
랜덤포레스트 - train
정확도:0.9920954389039496, 재현율:0.8839831544627089, 정밀도:0.9981592268752876, f1점수:0.9376080691642652
랜덤포레스트 - test
정확도:0.9429064322681344, 재현율:0.3065164923572003, 정밀도:0.678539626001781, f1점수:0.42227763923524525
```

Out[11]:
```
0.8235710966633374
```

In [16]:
```
# 랜덤포레스트 파라미터 변경
rf_clf = RandomForestClassifier(n_estimators=1000,
                                max_depth=10,
                                max_features=10)
rf_clf.fit(X_train, y_train)

pred_rf_train = rf_clf.predict(X_train)
pred_rf_test = rf_clf.predict(X_test)

print_metrics(y_train, pred_rf_train, '랜덤포레스트 - train')
print_metrics(y_test, pred_rf_test, '랜덤포레스트 - test')

pred_proba_rf = rf_clf.predict_proba(X_test)
positive_proba_rf = pred_proba_rf[:, 1]
auc = roc_auc_score(y_test, positive_proba_rf)
auc

# auc score는 이게 현재까지 최고
```

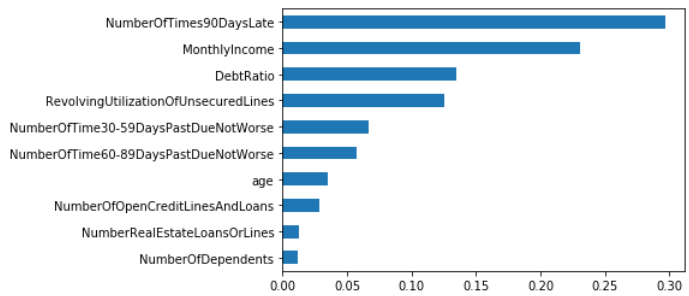
```
랜덤포레스트 - train
정확도:0.95489106127404, 재현율:0.3670696916179867, 정밀도:0.9051926298157454, f1점수:0.5223274695534507
랜덤포레스트 - test
정확도:0.9464388400558613, 재현율:0.31174577634754624, 정밀도:0.7598039215686274, f1점수:0.4420992584141472
```

Out[16]:
```
0.8975110204309532
```

In [20]:
```
# 특성 중요도
impt_features = pd.Series(rf_clf.feature_importances_, index=X.columns).sort_values()
impt_features
```

Out[20]:
```
NumberOfDependents                      0.011879
NumberRealEstateLoansOrLines            0.012520
NumberOfOpenCreditLinesAndLoans         0.028288
age                                     0.035271
NumberOfTime60-89DaysPastDueNotWorse    0.057906
NumberOfTime30-59DaysPastDueNotWorse    0.066688
RevolvingUtilizationOfUnsecuredLines    0.125181
DebtRatio                               0.134518
MonthlyIncome                           0.230620
NumberOfTimes90DaysLate                 0.297128
dtype: float64
```

In [22]:
```
impt_features.plot(kind='barh');
```



In [29]:
```
#최적의 파라미터는?
param = {
    'max_depth':range(3,11),
    'max_features':[6,7,8,9,10]
}

g_search = GridSearchCV(RandomForestClassifier(),
                        param,
                        cv = 3,
                        n_jobs = -1,
                        scoring=['accuracy','recall','roc_auc'],
                        refit = 'roc_auc')

g_search.fit(X_train, y_train)
```

```
C:\Users\Playdata\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators
will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[29]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': range(3, 11),
                         'max_features': [6, 7, 8, 9, 10]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring=['accuracy', 'recall', 'roc_auc'], verbose=0)
```

In [30]:
```
g_search.best_params_
```

Out[30]:
```
{'max_depth': 9, 'max_features': 6}
```

In [31]:
```
df = pd.DataFrame(g_search.cv_results_)
df.sort_values('rank_test_roc_auc').head()
```

Out[31]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_features	params	split0_test_accuracy	split1_test_accuracy	split2_t
30	2.160943	0.065452	0.203080	0.000002	9	6	{'max_depth': 9, 'max_features': 6}	0.943237	0.943290	
38	3.360138	0.025256	0.202792	0.000941	10	9	{'max_depth': 10, 'max_features': 9}	0.943757	0.942523	
35	2.360245	0.063612	0.208285	0.007363	10	6	{'max_depth': 10, 'max_features': 6}	0.942798	0.942989	
36	2.686515	0.025516	0.213491	0.007364	10	7	{'max_depth': 10, 'max_features': 7}	0.942579	0.944084	
37	3.034220	0.037468	0.207210	0.003219	10	8	{'max_depth': 10, 'max_features': 8}	0.943428	0.943125	

5 rows × 25 columns

In [32]:
```
best_rf_model = g_search.best_estimator_
impt_features = best_rf_model.feature_importances_
pd.Series(impt_features, index=X.columns).sort_values()
```

Out[32]:
```
NumberOfDependents                     0.009764
NumberRealEstateLoansOrLines           0.012693
NumberOfOpenCreditLinesAndLoans        0.022887
age                                    0.026900
NumberOfTime30-59DaysPastDueNotWorse   0.069536
DebtRatio                              0.107591
NumberOfTime60-89DaysPastDueNotWorse   0.108499
RevolvingUtilizationOfUnsecuredLines   0.128784
MonthlyIncome                          0.222493
NumberOfTimes90DaysLate                0.290854
dtype: float64
```

In [39]:
```
# stratify=y 적용
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[39]:
```
((109557, 10), (36519, 10), (109557,), (36519,))
```

In [41]:
```
rf_clf = RandomForestClassifier(n_estimators=1000,
                                max_depth=5,
                                max_features=10)
rf_clf.fit(X_train, y_train)

pred_rf_train = rf_clf.predict(X_train)
pred_rf_test = rf_clf.predict(X_test)

print_metrics(y_train, pred_rf_train, '랜덤포레스트 - train')
print_metrics(y_test, pred_rf_test, '랜덤포레스트 - test')

pred_proba_rf = rf_clf.predict_proba(X_test)
positive_proba_rf = pred_proba_rf[:, 1]
auc = roc_auc_score(y_test, positive_proba_rf)
auc
```

```
랜덤포레스트 - train
정확도:0.9402502806758126, 재현율:0.2043331076506432, 정밀도:0.692519504359798, f1점수:0.31555834378920955
랜덤포레스트 - test
정확도:0.9397026205536844, 재현율:0.19536961819658813, 정밀도:0.6851851851851852, f1점수:0.30404551201011376
```

Out[41]:
```
0.8707098044428647
```

In [34]:
```
#최적의 파라미터 적용
rf_clf = RandomForestClassifier(n_estimators=1000,
                                max_depth=9,
                                max_features=6)
rf_clf.fit(X_train, y_train)

pred_rf_train = rf_clf.predict(X_train)
pred_rf_test = rf_clf.predict(X_test)

print_metrics(y_train, pred_rf_train, '랜덤포레스트 - train')
print_metrics(y_test, pred_rf_test, '랜덤포레스트 - test')

pred_proba_rf = rf_clf.predict_proba(X_test)
positive_proba_rf = pred_proba_rf[:, 1]
auc = roc_auc_score(y_test, positive_proba_rf)
auc

# auc score는 오히려 감소
```

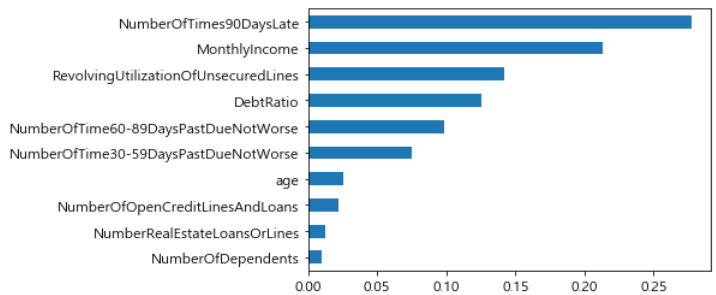
```
랜덤포레스트 - train
정확도:0.9519245689458455, 재현율:0.3364017869229728, 정밀도:0.8719298245614036, f1점수:0.48549379701084305
랜덤포레스트 - test
정확도:0.9444672636161998, 재현율:0.2711382113821138, 정밀도:0.7394678492239468, f1점수:0.3967876264128494
```

Out[34]:
```
0.8883422704789895
```

In [35]:
```
impt_features = pd.Series(rf_clf.feature_importances_, index=X.columns).sort_values()
impt_features
```

Out[35]:
```
NumberOfDependents                    0.009597
NumberRealEstateLoansOrLines          0.012118
NumberOfOpenCreditLinesAndLoans       0.021731
age                                   0.025840
NumberOfTime30-59DaysPastDueNotWorse  0.074968
NumberOfTime60-89DaysPastDueNotWorse  0.098654
DebtRatio                             0.125296
RevolvingUtilizationOfUnsecuredLines  0.141734
MonthlyIncome                         0.212881
NumberOfTimes90DaysLate               0.277181
dtype: float64
```

In [36]:
```
impt_features.plot(kind='barh');
```



GradientBoosting

In [3]:
```
gb_clf = GradientBoostingClassifier()
gb_clf.fit(X_train, y_train)

pred_gb_train = gb_clf.predict(X_train)
pred_gb_test = gb_clf.predict(X_test)

print_metrics(y_train, pred_gb_train, 'GradientBoosting - train')
print_metrics(y_test, pred_gb_test, 'GradientBoosting - test')

pred_proba_gb = gb_clf.predict_proba(X_test)
positive_proba_gb = pred_proba_gb[:, 1]
auc = roc_auc_score(y_test, positive_proba_gb)
auc
```

```
GradientBoosting - train
정확도:0.9469865001779896, 재현율:0.3105455529985109, 정밀도:0.7623795280824194, f1점수:0.4413235859946133
GradientBoosting - test
정확도:0.9454256688299242, 재현율:0.3016260162601626, 정밀도:0.7295968534906588, f1점수:0.42680471670980735
```

Out[3]:
```
0.8922045424761479
```

In [14]:
```
#GridSearch 사용해서 최적의 파라미터 찾기
param = {
    'max_depth':[1,2,3,4,5],
    'learning_rate':[0.01,0.1,0.5,1]
}
g_search = GridSearchCV(GradientBoostingClassifier(),
                        param_grid = param,
                        cv = 3,
                        n_jobs = -1,
                        scoring=['accuracy','precision','recall','f1','roc_auc'],
                        refit = 'roc_auc')

g_search.fit(X_train, y_train)
```

Out[14]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=GradientBoostingClassifier(criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=None,
                                                  presort='auto',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.1, 0.5, 1],
                         'max_depth': [1, 2, 3, 4, 5]},
             pre_dispatch='2*n_jobs', refit='roc_auc', return_train_score=False,
             scoring=['accuracy', 'precision', 'recall', 'f1', 'roc_auc'],
             verbose=0)
```

In [15]:
```
g_search.best_params_
```

Out[15]:
```
{'learning_rate': 0.1, 'max_depth': 5}
```

In [19]:
```
df = pd.DataFrame(g_search.cv_results_)
df.sort_values('rank_test_roc_auc').head()
```

Out[19]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_max_depth	params	split0_test_accuracy	split1_test_accuracy	split2_t
9	29.397986	0.776805	0.756371	0.082245	0.1	5	{'learning_rate': 0.1, 'max_depth': 5}	0.944578	0.944823	
8	20.877644	0.354163	0.584438	0.014860	0.1	4	{'learning_rate': 0.1, 'max_depth': 4}	0.945181	0.945179	
7	15.212464	0.129945	0.473004	0.003083	0.1	3	{'learning_rate': 0.1, 'max_depth': 3}	0.946002	0.944495	
11	10.254196	0.389029	0.371708	0.022385	0.5	2	{'learning_rate': 0.5, 'max_depth': 2}	0.944797	0.944330	
12	14.493571	0.145581	0.419916	0.010926	0.5	3	{'learning_rate': 0.5, 'max_depth': 3}	0.943209	0.944549	

5 rows × 37 columns

In [20]:
```
best_gb_model = g_search.best_estimator_
impt_features = best_gb_model.feature_importances_
pd.Series(impt_features, index=X.columns).sort_values()
```

Out[20]:
```
NumberOfDependents                      0.005344
NumberRealEstateLoansOrLines            0.008137
NumberOfOpenCreditLinesAndLoans         0.014170
age                                     0.017326
NumberOfTime60-89DaysPastDueNotWorse    0.057385
NumberOfTime30-59DaysPastDueNotWorse    0.082401
DebtRatio                               0.118518
RevolvingUtilizationOfUnsecuredLines    0.144997
MonthlyIncome                           0.215839
NumberOfTimes90DaysLate                 0.335884
dtype: float64
```

In [21]:
```
#최적의 파라미터 적용
gb_clf = GradientBoostingClassifier(learning_rate=0.1, max_depth=5)
gb_clf.fit(X_train, y_train)

pred_gb_train = gb_clf.predict(X_train)
pred_gb_test = gb_clf.predict(X_test)

print_metrics(y_train, pred_gb_train, 'GradientBoosting - train')
print_metrics(y_test, pred_gb_test, 'GradientBoosting - test')

pred_proba_gb = gb_clf.predict_proba(X_test)
positive_proba_gb = pred_proba_gb[:, 1]
auc = roc_auc_score(y_test, positive_proba_gb)
auc
```

```
GradientBoosting - train
정확도:0.9507836103580785, 재현율:0.343982672262082, 정밀도:0.8231292517006803, f1점수:0.4852014512125262
GradientBoosting - test
정확도:0.9450970727566472, 재현율:0.29715447154471547, 정밀도:0.7259185700099305, f1점수:0.42169022209402945
```

Out[21]:
```
0.8937882003897111
```

XGBoost(Extra Gradient Boost)

In [11]:
```
xgb = XGBClassifier()
xgb.fit(X_train, y_train)

pred_xgb_train = xgb.predict(X_train)
pred_xgb_test = xgb.predict(X_test)

print_metrics(y_train, pred_xgb_train, 'XGBoost - train')
print_metrics(y_test, pred_xgb_test, 'XGBoost - test')

pred_proba_xgb = xgb.predict_proba(X_test)
positive_proba_xgb = pred_proba_xgb[:, 1]
auc = roc_auc_score(y_test, positive_proba_xgb)
auc
```

```
XGBoost - train
정확도:0.9467948191352447, 재현율:0.3044537701367267, 정밀도:0.7649659863945578, f1점수:0.4355572770407669
XGBoost - test
정확도:0.9455899668665626, 재현율:0.29552845528455285, 정밀도:0.7410805300713558, f1점수:0.42255158384190644
```

Out[11]:
```
0.8921715234945004
```

In [ ]:
```
#최적의 파라미터는?
param = {
    'max_depth':[2,3,4],
    'learning_rate':[0.01,0.1,0.2],
    'n_estimators':[400,500,600]
}
g_search = GridSearchCV(XGBClassifier(),
                        param_grid = param,
                        cv = 3,
                        n_jobs = -1)

g_search.fit(X_train, y_train)
```

In [ ]:
```

```

예측 결과

가장 높게 얻은 auc score:

- 0.8975110204309532