File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
```

In [2]:
```
df = pd.read_csv("P39-Financial-Data.csv")
df.head()
```

Out[2]:

| | entry_id | age | pay_schedule | home_owner | income | months_employed | years_employed | current_address_year | personal_account_m | personal_account_y |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7629673 | 40 | bi-weekly | 1 | 3135 | 0 | 3 | 3 | 6 | 2 |
| 1 | 3560428 | 61 | weekly | 0 | 3180 | 0 | 6 | 3 | 2 | 7 |
| 2 | 6934997 | 23 | weekly | 0 | 1540 | 6 | 0 | 0 | 7 | 1 |
| 3 | 5682812 | 40 | bi-weekly | 0 | 5230 | 0 | 6 | 1 | 2 | 7 |
| 4 | 5335819 | 33 | semi-monthly | 0 | 3590 | 0 | 5 | 2 | 2 | 8 |

5 rows × 21 columns

In [3]:
```
df.isna().sum()
```

Out[3]:
```
entry_id                0
age                     0
pay_schedule            0
home_owner              0
income                  0
months_employed         0
years_employed          0
current_address_year    0
personal_account_m      0
personal_account_y      0
has_debt                0
amount_requested        0
risk_score              0
risk_score_2            0
risk_score_3            0
risk_score_4            0
risk_score_5            0
ext_quality_score       0
ext_quality_score_2     0
inquiries_last_month    0
e_signed                0
dtype: int64
```

## Feature Engineering 特征工程

In [4]:
```
df = df.drop(columns="months_employed")
df["personal_account_mounth"] = df["personal_account_m"] + df["personal_account_y"] * 12
df = df.drop(labels=["personal_account_m", "personal_account_y"], axis=1)
```

## Data Preprocessing

In [5]:
```
df = df.drop(columns="entry_id")
```

### *One Hot Encoder (before tts)*

In [6]:
```
df = pd.get_dummies(df, drop_first=True)
```

In [7]:
```
X = df.drop(columns="e_signed").to_numpy()
X
```

Out[7]:
```
array([[4.000e+01, 1.000e+00, 3.135e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [6.100e+01, 0.000e+00, 3.180e+03, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [2.300e+01, 0.000e+00, 1.540e+03, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       ...,
       [4.600e+01, 0.000e+00, 2.685e+03, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [4.200e+01, 0.000e+00, 2.515e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [2.900e+01, 1.000e+00, 2.665e+03, ..., 0.000e+00, 0.000e+00,
        1.000e+00]])
```

In [8]:
```
y = df["e_signed"].to_numpy()
y
```

Out[8]:
```
array([1, 0, 0, ..., 0, 1, 1], dtype=int64)
```

### *tts*

In [9]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

### *Feature Scaling*

In [10]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

## Model Training

### *Logistic Regression*

In [11]:
```
from sklearn.linear_model import LogisticRegression

clf_lr = LogisticRegression(random_state = 0, penalty="l2")
clf_lr.fit(X_train, y_train)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf_lr.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.54      0.39      0.45      1654
           1       0.58      0.71      0.63      1928

    accuracy                           0.56      3582
   macro avg       0.56      0.55      0.54      3582
weighted avg       0.56      0.56      0.55      3582

0.5622557230597431
[[ 653 1001]
 [ 567 1361]]
```

### *cv for lr*

In [12]:
```
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator=clf_lr, X=X_train, y=y_train, cv=10, verbose=1)
print(accuracies)
print(f"Mean: {accuracies.mean() * 100} %")
print(f"Std: {accuracies.std() * 100} %")
```

```
[0.57431961 0.5645499  0.57501745 0.57920447 0.58129798 0.56734124
 0.57402235 0.58449721 0.59776536 0.57332402]
Mean: 57.71339573578887 %
Std: 0.890141672731735 %
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.1s finished
```

### *Support Vector Machine*

In [13]:
```
%%time
from sklearn.svm import SVC

clf_svc = SVC(random_state = 0, kernel="linear")
clf_svc.fit(X_train, y_train)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf_svc.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.55      0.37      0.44      1654
           1       0.58      0.74      0.65      1928

    accuracy                           0.57      3582
   macro avg       0.56      0.55      0.55      3582
weighted avg       0.56      0.57      0.55      3582

0.568676716917923
[[ 619 1035]
 [ 510 1418]]
Wall time: 14.9 s
```

### *cv for svc*

In [14]:
```
%%time
```

```
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator=clf_svc, X=X_train, y=y_train, cv=10, verbose=1)
print(accuracies)
print(f"Mean: {accuracies.mean() * 100} %")
print(f"Std: {accuracies.std() * 100} %")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[0.58339149 0.56594557 0.57571528 0.57920447 0.58478716 0.57501745
 0.56913408 0.58310056 0.59706704 0.58030726]
Mean: 57.936703481776334 %
Std: 0.8311541039792876 %
Wall time: 2min 2s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:  2.0min finished
```

### *Random Forest*

In [15]:

```
%%time
from sklearn.ensemble import RandomForestClassifier

clf_rf = RandomForestClassifier(random_state = 0, n_estimators = 100, criterion = 'entropy')
clf_rf.fit(X_train, y_train)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

predictions = clf_rf.predict(X_test)
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.60      0.56      0.58      1654
           1       0.65      0.68      0.66      1928

    accuracy                           0.63      3582
   macro avg       0.62      0.62      0.62      3582
weighted avg       0.63      0.63      0.63      3582

0.6267448352875489
[[ 931  723]
 [ 614 1314]]
Wall time: 4.61 s
```

### *cv for rf*

In [16]:

```
%%time
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator=clf_rf, X=X_train, y=y_train, cv=10, verbose=1)
print(accuracies)
print(f"Mean: {accuracies.mean() * 100} %")
print(f"Std: {accuracies.std() * 100} %")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[0.63014655 0.6350314  0.61898116 0.619679   0.62595953 0.62177251
 0.65782123 0.62849162 0.62988827 0.64315642]
Mean: 63.10927674488415 %
Std: 1.129048353801639 %
Wall time: 43.4 s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   43.3s finished
```

### *GridSearchCV for Random Forest*

In [23]:

```
%%time
# Round 1: Entropy

from sklearn.model_selection import GridSearchCV

parameters = {"max_depth": [3, None],
              "max_features": [1, 5, 10],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 5, 10],
              "bootstrap": [True, False],
              "criterion": ["entropy"]}

grid_search = GridSearchCV(estimator=clf_rf, param_grid=parameters, scoring="accuracy", cv=10, n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
print(grid_search.best_params_)
print(grid_search.best_score_) # precision
print(grid_search.best_index_)
```

```
Fitting 10 folds for each of 108 candidates, totalling 1080 fits
RandomForestClassifier(criterion='entropy', max_features=10, min_samples_leaf=5,
                       random_state=0)
{'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_features': 10, 'min_samples_leaf': 5, 'min_samples_spli
t': 2}
0.6362573438541638
48
Wall time: 24min 48s
```

In [24]:

```
%%time
```

```
# Round 2: Entropy

from sklearn.model_selection import GridSearchCV

parameters = {"max_depth": [None],
              "max_features": [3, 5, 7],
              'min_samples_split': [8, 10, 12],
              'min_samples_leaf': [1, 2, 3],
              "bootstrap": [True],
              "criterion": ["entropy"]}

grid_search = GridSearchCV(estimator=clf_rf, param_grid=parameters, scoring="accuracy", cv=10, verbose=1)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
print(grid_search.best_params_)
print(grid_search.best_score_) # precision
print(grid_search.best_index_)
```

```
Fitting 10 folds for each of 27 candidates, totalling 270 fits
RandomForestClassifier(criterion='entropy', max_features=7, min_samples_leaf=2,
                       min_samples_split=10, random_state=0)
{'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_features': 7, 'min_samples_leaf': 2, 'min_samples_spli
t': 10}
0.6381423801299769
22
Wall time: 23min 38s
```

In [25]:

```
%%time
# Round 1: Gini

from sklearn.model_selection import GridSearchCV

parameters = {"max_depth": [3, None],
              "max_features": [1, 5, 10],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 5, 10],
              "bootstrap": [True, False],
              "criterion": ["gini"]}

grid_search = GridSearchCV(estimator=clf_rf, param_grid=parameters, scoring="accuracy", cv=10, verbose=1)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
print(grid_search.best_params_)
print(grid_search.best_score_) # precision
print(grid_search.best_index_)
```

```
Fitting 10 folds for each of 108 candidates, totalling 1080 fits
RandomForestClassifier(max_features=5, min_samples_split=10, random_state=0)
{'bootstrap': True, 'criterion': 'gini', 'max_depth': None, 'max_features': 5, 'min_samples_leaf': 1, 'min_samples_split': 1
0}
0.6368143949287932
38
Wall time: 49min 33s
```

In [26]:

```
%%time
# Round 2: Gini

from sklearn.model_selection import GridSearchCV

parameters = {"max_depth": [None],
              "max_features": [8, 10, 12],
              'min_samples_split': [2, 3, 4],
              'min_samples_leaf': [8, 10, 12],
              "bootstrap": [True],
              "criterion": ["gini"]}

grid_search = GridSearchCV(estimator=clf_rf, param_grid=parameters, scoring="accuracy", cv=10, verbose=1)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
print(grid_search.best_params_)
print(grid_search.best_score_) # precision
print(grid_search.best_index_)
```

```
Fitting 10 folds for each of 27 candidates, totalling 270 fits
RandomForestClassifier(max_features=12, min_samples_leaf=8, random_state=0)
{'bootstrap': True, 'criterion': 'gini', 'max_depth': None, 'max_features': 12, 'min_samples_leaf': 8, 'min_samples_split':
2}
0.6355596046111802
18
Wall time: 25min 22s
```