Credit Risk Prediction

Definition

Credit risk is the probability of a financial loss resulting from a borrower's failure to repay a loan. Essentially, credit risk refers to the risk that a lender may not receive the owed principal and interest, which results in an interruption of cash flows and increased costs for collection. Lenders can mitigate credit risk by **analyzing factors about a borrower's creditworthiness**, such as their current debt load and income.

Credit risks are calculated based on the borrower's overall ability to repay a loan according to its original terms. To assess credit risk on a consumer loan, lenders often look at the five Cs of credit: credit history, capacity to repay, capital, the loan's conditions, and associated collateral

[source : Investopedia]

Import Libraries

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

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
```

```
import warnings
warnings.filterwarnings('ignore')
```

Get Dataset

In [2]:
```
lc = pd.read_csv('loan_data_2007_2014.csv')
lc.head()
```

Out[2]:

	Unnamed: 0	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	...	total_bal_il	il_
0	0	1077501	1296599	5000	5000	4975.0	36 months	10.65	162.87	B	...	NaN	N
1	1	1077430	1314167	2500	2500	2500.0	60 months	15.27	59.83	C	...	NaN	N
2	2	1077175	1313524	2400	2400	2400.0	36 months	15.96	84.33	C	...	NaN	N
3	3	1076863	1277178	10000	10000	10000.0	36 months	13.49	339.31	C	...	NaN	N
4	4	1075358	1311748	3000	3000	3000.0	60 months	12.69	67.79	B	...	NaN	N

5 rows × 75 columns

In [3]:
```
lc.shape
```

Out[3]: (466285, 75)

Explore Data

In [4]:
```
# check if there any duplicated columns
lc.duplicated().sum()
```

Out[4]: 0

In [5]:
```python
# no need to remove duplicated columns
```

In [6]:
```python
# check the missing values in our dataset
missing = lc.isnull().sum()/lc.shape[0]*100
missing = missing.sort_values(ascending=False)
missing[missing>0]
```

Out[6]:
```
inq_last_12m                  100.000000
total_bal_il                  100.000000
dti_joint                     100.000000
verification_status_joint     100.000000
annual_inc_joint              100.000000
open_acc_6m                   100.000000
open_il_6m                    100.000000
open_il_12m                   100.000000
open_il_24m                   100.000000
mths_since_rcnt_il            100.000000
il_util                       100.000000
open_rv_24m                   100.000000
total_cu_tl                   100.000000
inq_fi                        100.000000
max_bal_bc                    100.000000
all_util                      100.000000
open_rv_12m                   100.000000
mths_since_last_record         86.566585
mths_since_last_major_derog    78.773926
desc                           72.981975
mths_since_last_delinq         53.690554
next_pymnt_d                   48.728567
tot_cur_bal                    15.071469
tot_coll_amt                   15.071469
total_rev_hi_lim               15.071469
emp_title                       5.916553
emp_length                      4.505399
last_pymnt_d                    0.080637
revol_util                      0.072917
collections_12_mths_ex_med      0.031097
last_credit_pull_d              0.009007
inq_last_6mths                  0.006219
earliest_cr_line                0.006219
```

```
delinq_2yrs                         0.006219
open_acc                            0.006219
pub_rec                             0.006219
acc_now_delinq                      0.006219
total_acc                           0.006219
title                               0.004504
annual_inc                          0.000858
dtype: float64
```

In [7]:
```
# remove columns whose missing values more than 40%
lc.dropna(thresh=lc.shape[0]*0.6, axis=1, inplace=True)
lc.shape
```

Out[7]:
```
(466285, 53)
```

In [8]:
```
# divide into categorical and numerical column for more data exploration
lc_cat = lc.select_dtypes(include=['object']).columns
lc_num = lc.select_dtypes(include=['float64','int64']).columns
```

In [9]:
```
# explore categorical columns
lc[lc_cat].describe().T
```

Out[9]:

	count	unique	top	freq
term	466285	2	36 months	337953
grade	466285	7	B	136929
sub_grade	466285	35	B3	31686
emp_title	438697	205475	Teacher	5399
emp_length	445277	11	10+ years	150049
home_ownership	466285	6	MORTGAGE	235875
verification_status	466285	3	Verified	168055
issue_d	466285	91	Oct-14	38782
loan_status	466285	9	Current	224226
pymnt_plan	466285	2	n	466276

url	466285	466285	https://www.lendingclub.com/browse/loanDetail....	1
purpose	466285	14	debt_consolidation	274195
title	466264	63098	Debt consolidation	164075
zip_code	466285	888	945xx	5304
addr_state	466285	50	CA	71450
earliest_cr_line	466256	664	Oct-00	3674
initial_list_status	466285	2	f	303005
last_pymnt_d	465909	98	Jan-16	179620
last_credit_pull_d	466243	103	Jan-16	327699
application_type	466285	1	INDIVIDUAL	466285

In [10]:

```
# remove columns whose too many unique values
lc.drop(['emp_title','url','title','zip_code'], axis=1, inplace=True)

# remove columns whose only 1 unique value
lc.drop('application_type', axis=1, inplace=True)

# remove columns whose particular value that is dominant
lc.drop('pymnt_plan', axis=1, inplace=True)

# remove columns which are considered useless as predictor
lc.drop(['issue_d','earliest_cr_line','last_pymnt_d','last_credit_pull_d'], axis=1, inplace=True)

lc.shape
```

Out[10]: (466285, 43)

In [11]:

```
# explore numerical columns
lc[lc_num].describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	466285.0	2.331420e+05	1.346050e+05	0.00	1.165710e+05	2.331420e+05	3.497130e+05	4.662840e+05

id	466285.0	1.307973e+07	1.089371e+07	54734.00	3.639987e+06	1.010790e+07	2.073121e+07	3.809811e+07
member_id	466285.0	1.459766e+07	1.168237e+07	70473.00	4.379705e+06	1.194108e+07	2.300154e+07	4.086083e+07
loan_amnt	466285.0	1.431728e+04	8.286509e+03	500.00	8.000000e+03	1.200000e+04	2.000000e+04	3.500000e+04
funded_amnt	466285.0	1.429180e+04	8.274371e+03	500.00	8.000000e+03	1.200000e+04	2.000000e+04	3.500000e+04
funded_amnt_inv	466285.0	1.422233e+04	8.297638e+03	0.00	8.000000e+03	1.200000e+04	1.995000e+04	3.500000e+04
int_rate	466285.0	1.382924e+01	4.357587e+00	5.42	1.099000e+01	1.366000e+01	1.649000e+01	2.606000e+01
installment	466285.0	4.320612e+02	2.434855e+02	15.67	2.566900e+02	3.798900e+02	5.665800e+02	1.409990e+03
annual_inc	466281.0	7.327738e+04	5.496357e+04	1896.00	4.500000e+04	6.300000e+04	8.896000e+04	7.500000e+06
dti	466285.0	1.721876e+01	7.851121e+00	0.00	1.136000e+01	1.687000e+01	2.278000e+01	3.999000e+01
delinq_2yrs	466256.0	2.846784e-01	7.973651e-01	0.00	0.000000e+00	0.000000e+00	0.000000e+00	2.900000e+01
inq_last_6mths	466256.0	8.047446e-01	1.091598e+00	0.00	0.000000e+00	0.000000e+00	1.000000e+00	3.300000e+01
open_acc	466256.0	1.118707e+01	4.987526e+00	0.00	8.000000e+00	1.000000e+01	1.400000e+01	8.400000e+01
pub_rec	466256.0	1.605642e-01	5.108626e-01	0.00	0.000000e+00	0.000000e+00	0.000000e+00	6.300000e+01
revol_bal	466285.0	1.623020e+04	2.067625e+04	0.00	6.413000e+03	1.176400e+04	2.033300e+04	2.568995e+06
revol_util	465945.0	5.617695e+01	2.373263e+01	0.00	3.920000e+01	5.760000e+01	7.470000e+01	8.923000e+02
total_acc	466256.0	2.506443e+01	1.160014e+01	1.00	1.700000e+01	2.300000e+01	3.200000e+01	1.560000e+02
out_prncp	466285.0	4.410062e+03	6.355079e+03	0.00	0.000000e+00	4.414700e+02	7.341650e+03	3.216038e+04
out_prncp_inv	466285.0	4.408452e+03	6.353198e+03	0.00	0.000000e+00	4.413800e+02	7.338390e+03	3.216038e+04
total_pymnt	466285.0	1.154069e+04	8.265627e+03	0.00	5.552125e+03	9.419251e+03	1.530816e+04	5.777758e+04
total_pymnt_inv	466285.0	1.146989e+04	8.254158e+03	0.00	5.499250e+03	9.355430e+03	1.523131e+04	5.777758e+04
total_rec_prncp	466285.0	8.866015e+03	7.031688e+03	0.00	3.708560e+03	6.817760e+03	1.200000e+04	3.500003e+04
total_rec_int	466285.0	2.588677e+03	2.483810e+03	0.00	9.572800e+02	1.818880e+03	3.304530e+03	2.420562e+04
total_rec_late_fee	466285.0	6.501292e-01	5.265730e+00	0.00	0.000000e+00	0.000000e+00	0.000000e+00	3.586800e+02
recoveries	466285.0	8.534421e+01	5.522161e+02	0.00	0.000000e+00	0.000000e+00	0.000000e+00	3.352027e+04
collection_recovery_fee	466285.0	8.961534e+00	8.549144e+01	0.00	0.000000e+00	0.000000e+00	0.000000e+00	7.002190e+03

last_pymnt_amnt	466285.0	3.123914e+03	5.554737e+03	0.00	3.126200e+02	5.459600e+02	3.187510e+03	3.623444e+04
collections_12_mths_ex_med	466140.0	9.085253e-03	1.086484e-01	0.00	0.000000e+00	0.000000e+00	0.000000e+00	2.000000e+01
policy_code	466285.0	1.000000e+00	0.000000e+00	1.00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
acc_now_delinq	466256.0	4.002093e-03	6.863680e-02	0.00	0.000000e+00	0.000000e+00	0.000000e+00	5.000000e+00
tot_coll_amt	396009.0	1.919135e+02	1.463021e+04	0.00	0.000000e+00	0.000000e+00	0.000000e+00	9.152545e+06
tot_cur_bal	396009.0	1.388017e+05	1.521147e+05	0.00	2.861800e+04	8.153900e+04	2.089530e+05	8.000078e+06
total_rev_hi_lim	396009.0	3.037909e+04	3.724713e+04	0.00	1.350000e+04	2.280000e+04	3.790000e+04	9.999999e+06

In [12]:
```
# remove identity columns
lc.drop(['Unnamed: 0','id','member_id'], axis=1, inplace=True)
lc.shape
```

Out[12]:
```
(466285, 40)
```

Define Target Label

In [13]:
```
lc['loan_status'].value_counts(normalize=True)
```

Out[13]:
```
loan_status
Current                                                0.480878
Fully Paid                                             0.396193
Charged Off                                            0.091092
Late (31-120 days)                                     0.014798
In Grace Period                                        0.006747
Does not meet the credit policy. Status:Fully Paid     0.004263
Late (16-30 days)                                      0.002612
Default                                                0.001784
Does not meet the credit policy. Status:Charged Off    0.001632
Name: proportion, dtype: float64
```

In [14]:
```
# assume charged off, Late (31-120 days), Late (16-30 days), Default, and Does not meet the credit policy. Status:C
# as BAD BORROWERS
lc['loan_status'] = np.where(lc.loc[:,'loan_status'].isin
                               (['Charged Off','Default','Late (31-120 days)','Late (16-30 days)',
```

```
                                'Does not meet the credit policy. Status:Charged Off'
                                ]),1,0)

lc['loan_status'].value_counts(normalize=True)
```

Out[14]:
```
loan_status
0    0.888081
1    0.111919
Name: proportion, dtype: float64
```

In [15]:
```
# okay, target label has been defined!
```

In [16]:
```
lc['loan_status'].head()
```

Out[16]:
```
0    0
1    1
2    0
3    0
4    0
Name: loan_status, dtype: int32
```

Split Data into Training and Testing Set

In [17]:
```
X = lc.drop('loan_status', axis=1)
y = lc['loan_status']

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

In [18]:
```
X_train.shape, X_test.shape
```

Out[18]:
```
((373028, 39), (93257, 39))
```

In [19]:
```
y_train.value_counts(normalize=True)
```

Out[19]:
```
loan_status
0    0.888081
1    0.111919
Name: proportion, dtype: float64
```

In [20]:
```
y_test.value_counts(normalize=True)
```

Out[20]:
```
loan_status
0    0.888083
1    0.111917
Name: proportion, dtype: float64
```

Clean the Data

In [21]:
```
cat = X_train.select_dtypes(include=['object']).columns
num = X_train.select_dtypes(include=['float64','int64']).columns
```

In [22]:
```
for col in cat:
    print(col, ' -----------> ', X_train[col].nunique())
    print(X_train[col].unique())
    print()
```

```
term  ----------->  2
[' 36 months' ' 60 months']

grade  ----------->  7
['C' 'D' 'B' 'A' 'F' 'E' 'G']

sub_grade  ----------->  35
['C1' 'C4' 'D4' 'B3' 'B2' 'D2' 'D3' 'D1' 'C2' 'C3' 'A1' 'A5' 'C5' 'B1'
 'B4' 'D5' 'F3' 'A4' 'B5' 'E2' 'A3' 'A2' 'E4' 'E1' 'E5' 'E3' 'F2' 'F5'
 'G1' 'G5' 'F1' 'G2' 'F4' 'G3' 'G4']

emp_length  ----------->  11
['9 years' '2 years' '10+ years' '8 years' '7 years' '< 1 year' '1 year'
 '3 years' '6 years' '4 years' '5 years' nan]

home_ownership  ----------->  6
['OWN' 'MORTGAGE' 'RENT' 'OTHER' 'NONE' 'ANY']

verification_status  ----------->  3
['Verified' 'Not Verified' 'Source Verified']
```

```
purpose  ----------->  14
['debt_consolidation' 'other' 'house' 'credit_card' 'medical'
 'major_purchase' 'car' 'home_improvement' 'vacation' 'small_business'
 'wedding' 'moving' 'educational' 'renewable_energy']

addr_state  ----------->  50
['FL' 'KY' 'NY' 'NJ' 'OH' 'AZ' 'IL' 'CA' 'WA' 'MN' 'SC' 'WI' 'OR' 'MD'
 'TN' 'PA' 'GA' 'TX' 'VA' 'AL' 'CO' 'MA' 'WY' 'OK' 'CT' 'NV' 'AR' 'NM'
 'NH' 'IN' 'KS' 'MS' 'MI' 'UT' 'NC' 'HI' 'DE' 'MO' 'MT' 'LA' 'SD' 'WV'
 'RI' 'DC' 'AK' 'VT' 'ME' 'NE' 'ID' 'IA']

initial_list_status  ----------->  2
['f' 'w']
```

In [23]:

```python
# clean column which consist number and change it's data type
X_train['term'] = pd.to_numeric(X_train['term'].str.replace(' months', ''))

X_train['emp_length'] = X_train['emp_length'].str.replace(' years', '')
X_train['emp_length'] = X_train['emp_length'].str.replace(' year', '')
X_train['emp_length'] = X_train['emp_length'].str.replace('+', '')
X_train['emp_length'] = X_train['emp_length'].str.replace('< 1', '0')
X_train['emp_length'].fillna(value=0, inplace=True)
X_train['emp_length'] = pd.to_numeric(X_train['emp_length'])
```

In [24]:

```python
X_train[['term', 'emp_length']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 373028 entries, 120814 to 397161
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   term        373028 non-null  int64
 1   emp_length  373028 non-null  int64
dtypes: int64(2)
memory usage: 8.5 MB
```

In [25]:

```python
X_train[cat].isnull().sum()
```

Out[25]:

```
term                  0
grade                 0
```

```
sub_grade              0
emp_length             0
home_ownership         0
verification_status    0
purpose                0
addr_state             0
initial_list_status    0
dtype: int64
```

In [26]:
```python
# conduct the same data cleaning for testing set too
```

In [27]:
```python
# clean column which consist number and change it's data type
X_test['term'] = pd.to_numeric(X_test['term'].str.replace(' months', ''))

X_test['emp_length'] = X_test['emp_length'].str.replace(' years', '')
X_test['emp_length'] = X_test['emp_length'].str.replace(' year', '')
X_test['emp_length'] = X_test['emp_length'].str.replace('+', '')
X_test['emp_length'] = X_test['emp_length'].str.replace('< 1', '0')
X_test['emp_length'].fillna(value=0, inplace=True)
X_test['emp_length'] = pd.to_numeric(X_test['emp_length'])
```

In [28]:
```python
X_test[['term', 'emp_length']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 93257 entries, 124850 to 175867
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   term        93257 non-null  int64
 1   emp_length  93257 non-null  int64
dtypes: int64(2)
memory usage: 2.1 MB
```

In [29]:
```python
X_test[cat].isnull().sum()
```

Out[29]:
```
term                   0
grade                  0
sub_grade              0
emp_length             0
home_ownership         0
```

```
verification_status    0
purpose                0
addr_state             0
initial_list_status    0
dtype: int64
```

In [30]:
```
# handle missing values in numerical column
num_miss = X_train[num].isnull().sum()[X_train[num].isnull().sum()>0]
num_miss
```

Out[30]:
```
annual_inc                        3
delinq_2yrs                      25
inq_last_6mths                   25
open_acc                         25
pub_rec                          25
revol_util                      269
total_acc                        25
collections_12_mths_ex_med      117
acc_now_delinq                   25
tot_coll_amt                  56210
tot_cur_bal                   56210
total_rev_hi_lim              56210
dtype: int64
```

In [31]:
```
X_train[num_miss.index] = X_train[num_miss.index].fillna(X_train[num_miss.index].median())
```

In [32]:
```
num_miss = X_train[num].isnull().sum()[X_train[num].isnull().sum()>0]
num_miss
```

Out[32]:
```
Series([], dtype: int64)
```

In [33]:
```
num_miss = X_test[num].isnull().sum()[X_test[num].isnull().sum()>0]
num_miss
```

Out[33]:
```
annual_inc                        1
delinq_2yrs                       4
inq_last_6mths                    4
open_acc                          4
pub_rec                           4
revol_util                       71
```

```
total_acc                          4
collections_12_mths_ex_med        28
acc_now_delinq                     4
tot_coll_amt                   14066
tot_cur_bal                    14066
total_rev_hi_lim               14066
dtype: int64
```

In [34]:
```python
X_test[num_miss.index] = X_test[num_miss.index].fillna(X_test[num_miss.index].median())
```

In [35]:
```python
num_miss = X_test[num].isnull().sum()[X_test[num].isnull().sum()>0]
num_miss
```

Out[35]:
```
Series([], dtype: int64)
```

In [36]:
```python
cat = X_train.select_dtypes(include=['object']).columns
num = X_train.select_dtypes(include=['float64','int64']).columns
```

Feature Encoding

In [37]:
```python
encoder = LabelEncoder()
for col in cat:
    encoder.fit(X_train[col].unique())
    X_train[col] = encoder.transform(X_train[col])
    X_test[col] = encoder.transform(X_test[col])
```

Feature Scaling

In [38]:
```python
scaler = StandardScaler()
for col in num:
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.fit_transform(X_test)
```

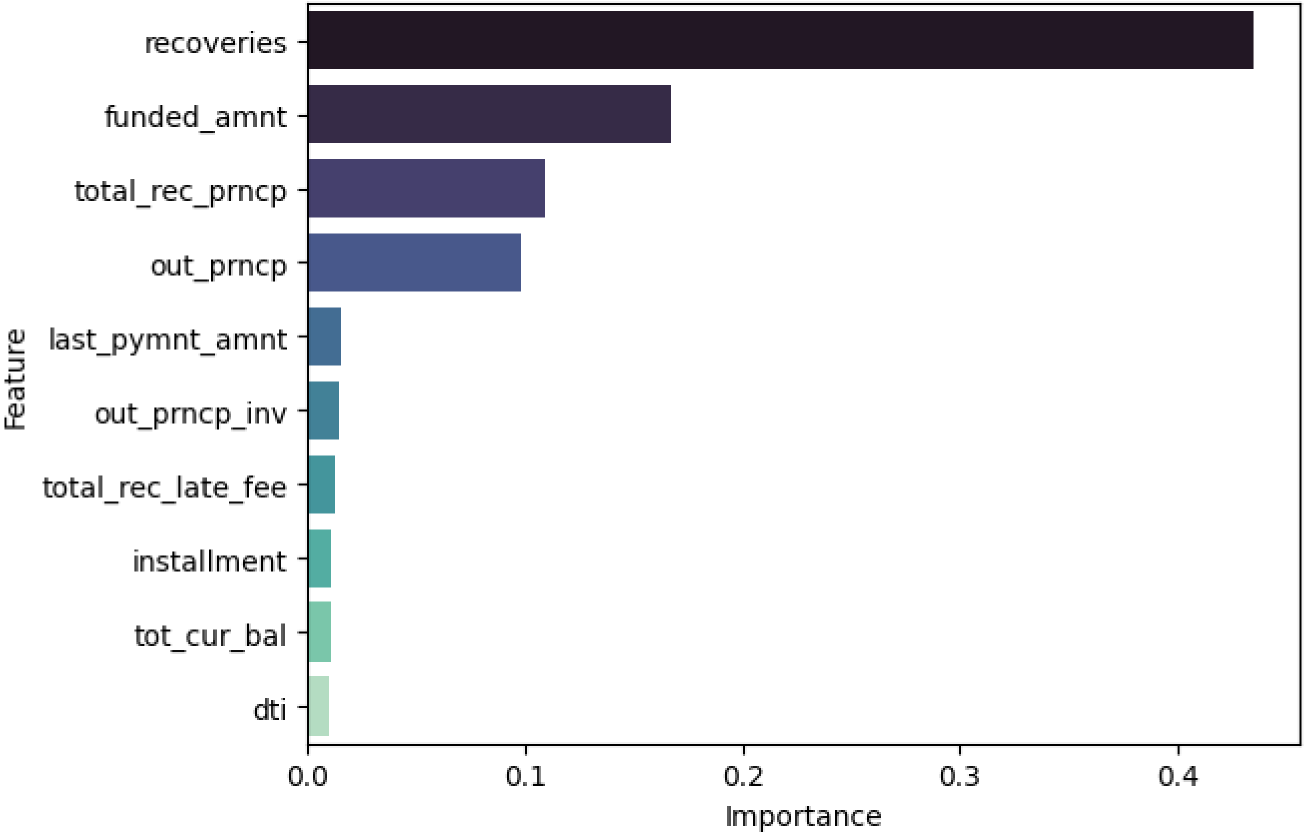
Modeling

Decision Tree Classifier

In [39]:
```
# decision tree classifier model
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
dtree_preds = dtree.predict(X_test)
print('model : DecisionTreeClassifier')
print('accuracy score : %', round(accuracy_score(y_test,dtree_preds)*100,2))
print('f1 score : %', round(f1_score(y_test,dtree_preds,average='micro')*100,2))
print('precision score : %', round(precision_score(y_test,dtree_preds,average='micro')*100,2))
print('recall score : %', round(recall_score(y_test,dtree_preds,average='micro')*100,2))
print()
```

```
model : DecisionTreeClassifier
accuracy score : % 95.22
f1 score : % 95.22
precision score : % 95.22
recall score : % 95.22
```

In [40]:
```
# feature importance of decision tree classifier model
dtree_fi = pd.DataFrame({'Feature' : X.columns, 'Importance' : dtree.feature_importances_})
dtree_fi = dtree_fi.sort_values('Importance', ascending=False)
dtree_fi = dtree_fi.head(10)
sns.barplot(data=dtree_fi, x='Importance', y='Feature', palette='mako', orient='h')
plt.show()
```



Logistic Regression

In [41]:

```
# logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
logreg_preds = logreg.predict(X_test)
print('model : LinerRegression')
print('accuracy score : %', round(accuracy_score(y_test,logreg_preds)*100,2))
print('f1 score : %', round(f1_score(y_test,logreg_preds,average='micro')*100,2))
print('precision score : %', round(precision_score(y_test,logreg_preds,average='micro')*100,2))
print('recall score : %', round(recall_score(y_test,logreg_preds,average='micro')*100,2))
print()
```

```
model : LinerRegression
accuracy score : % 97.88
f1 score : % 97.88
precision score : % 97.88
recall score : % 97.88
```

In [42]:

```
# feature importance of logistic regression model
logreg_fi = pd.DataFrame({'Feature': X.columns, 'Importance': np.abs(logreg.coef_[0])})
logreg_fi = logreg_fi.sort_values('Importance', ascending=False)
logreg_fi = logreg_fi.head(10)
sns.barplot(data=logreg_fi, x='Importance', y='Feature', palette='mako', orient='h')
plt.show()
```