Application of ML in Industries

Experiment 5 Credit Card Fraud Prediction

Dhruv Singhal || 500075346 || R177219074 || AIML B3 || Sem 5

Importing 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, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Reading Dataset

In [2]:
```
dataset=pd.read_csv('UCI_Credit_Card.csv')
dataset.head()
```

Out[2]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_A
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	0.0	6
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261.0	0.0	10
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	15
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	20

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AI
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131.0	2000.0	366

5 rows × 25 columns

In [3]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   ID                          30000 non-null  int64  
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64  
 3   EDUCATION                   30000 non-null  int64  
 4   MARRIAGE                    30000 non-null  int64  
 5   AGE                         30000 non-null  int64  
 6   PAY_0                       30000 non-null  int64  
 7   PAY_2                       30000 non-null  int64  
 8   PAY_3                       30000 non-null  int64  
 9   PAY_4                       30000 non-null  int64  
 10  PAY_5                       30000 non-null  int64  
 11  PAY_6                       30000 non-null  int64  
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
```

```
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
ID                          0
LIMIT_BAL                   0
SEX                         0
EDUCATION                   0
MARRIAGE                    0
AGE                         0
PAY_0                       0
PAY_2                       0
PAY_3                       0
PAY_4                       0
PAY_5                       0
PAY_6                       0
BILL_AMT1                   0
BILL_AMT2                   0
BILL_AMT3                   0
BILL_AMT4                   0
BILL_AMT5                   0
BILL_AMT6                   0
PAY_AMT1                    0
PAY_AMT2                    0
PAY_AMT3                    0
PAY_AMT4                    0
PAY_AMT5                    0
PAY_AMT6                    0
default.payment.next.month  0
dtype: int64
```

In [5]:
```
dataset.describe()
```

Out[5]:

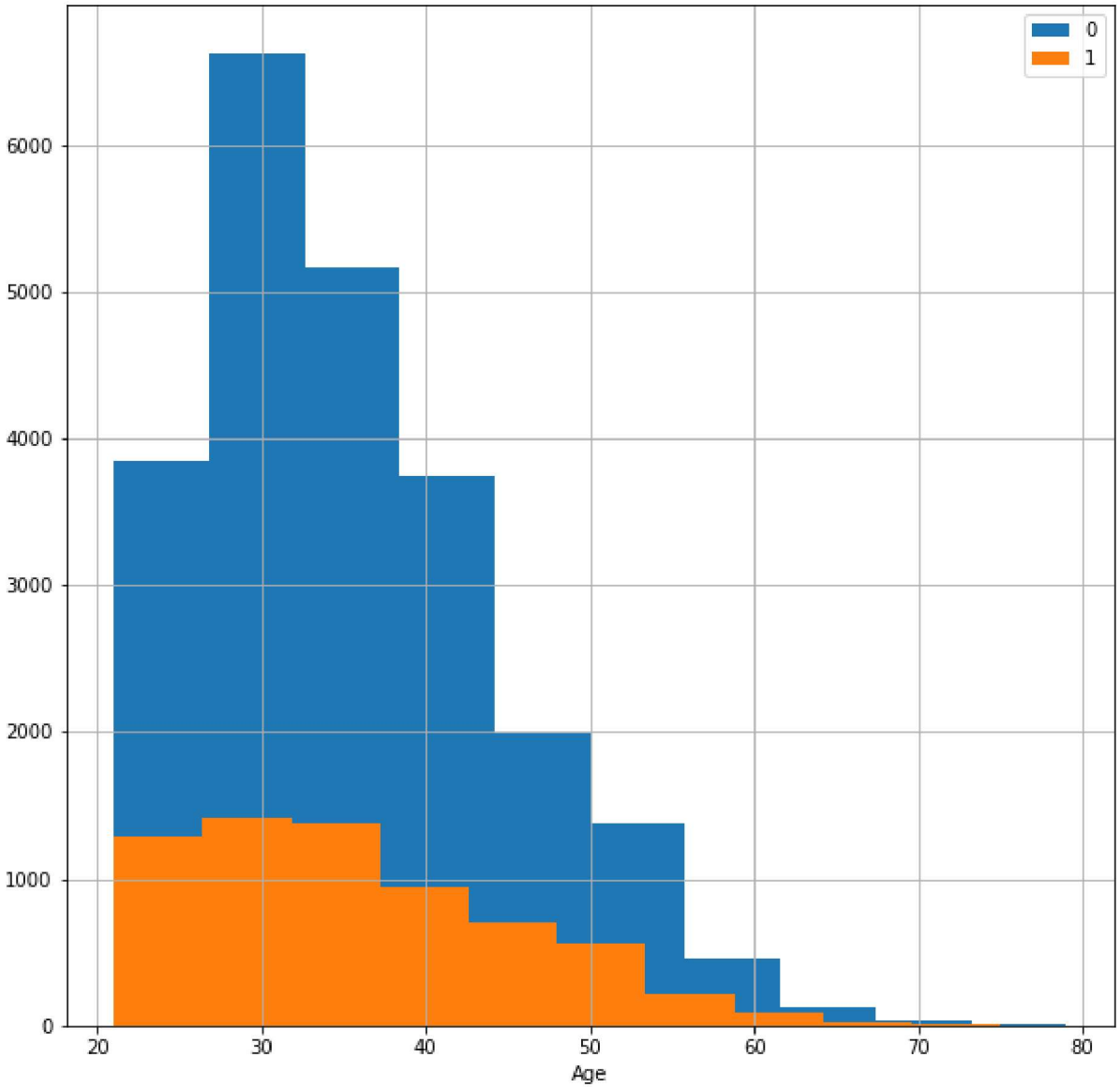
	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200	-0.220667
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868	1.169139
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	0.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	8.000000

8 rows × 25 columns

In [6]:

```
plt.figure(figsize=[10,10])
dataset.groupby('default.payment.next.month')['AGE'].hist(legend=True)
plt.xlabel('Age')
plt.show()
```

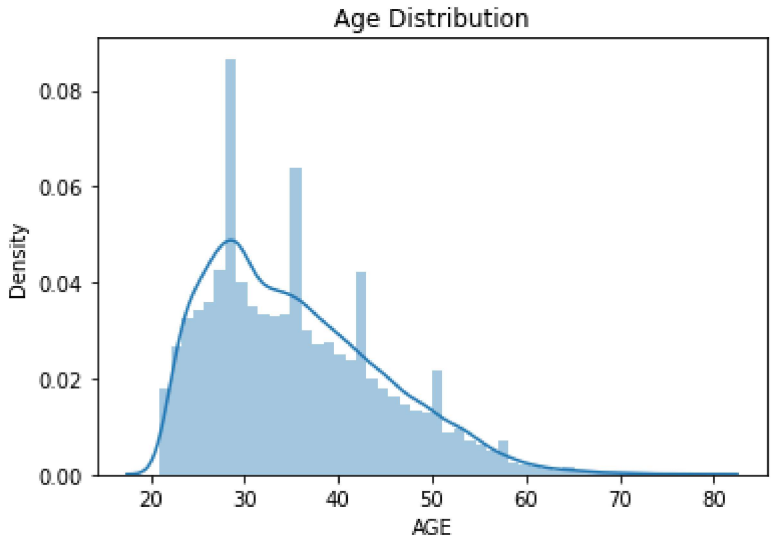


In [7]:

```
sns.distplot(dataset['AGE'])
plt.title('Age Distribution')
```

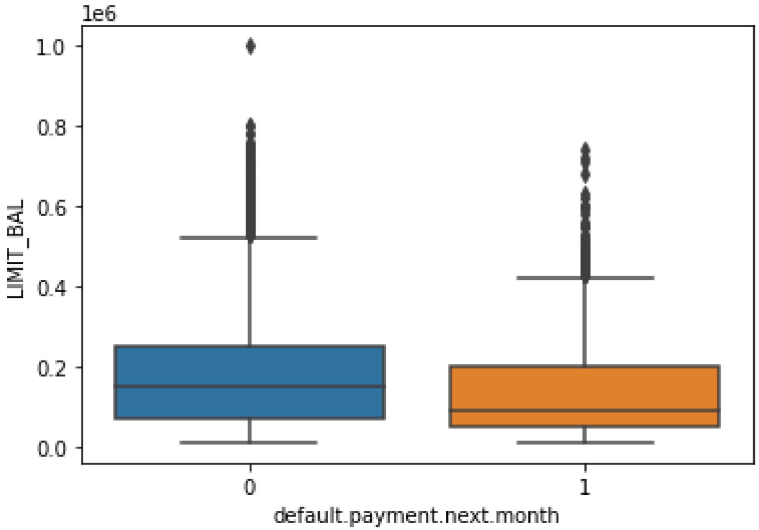
Out[7]:

```
Text(0.5, 1.0, 'Age Distribution')
```



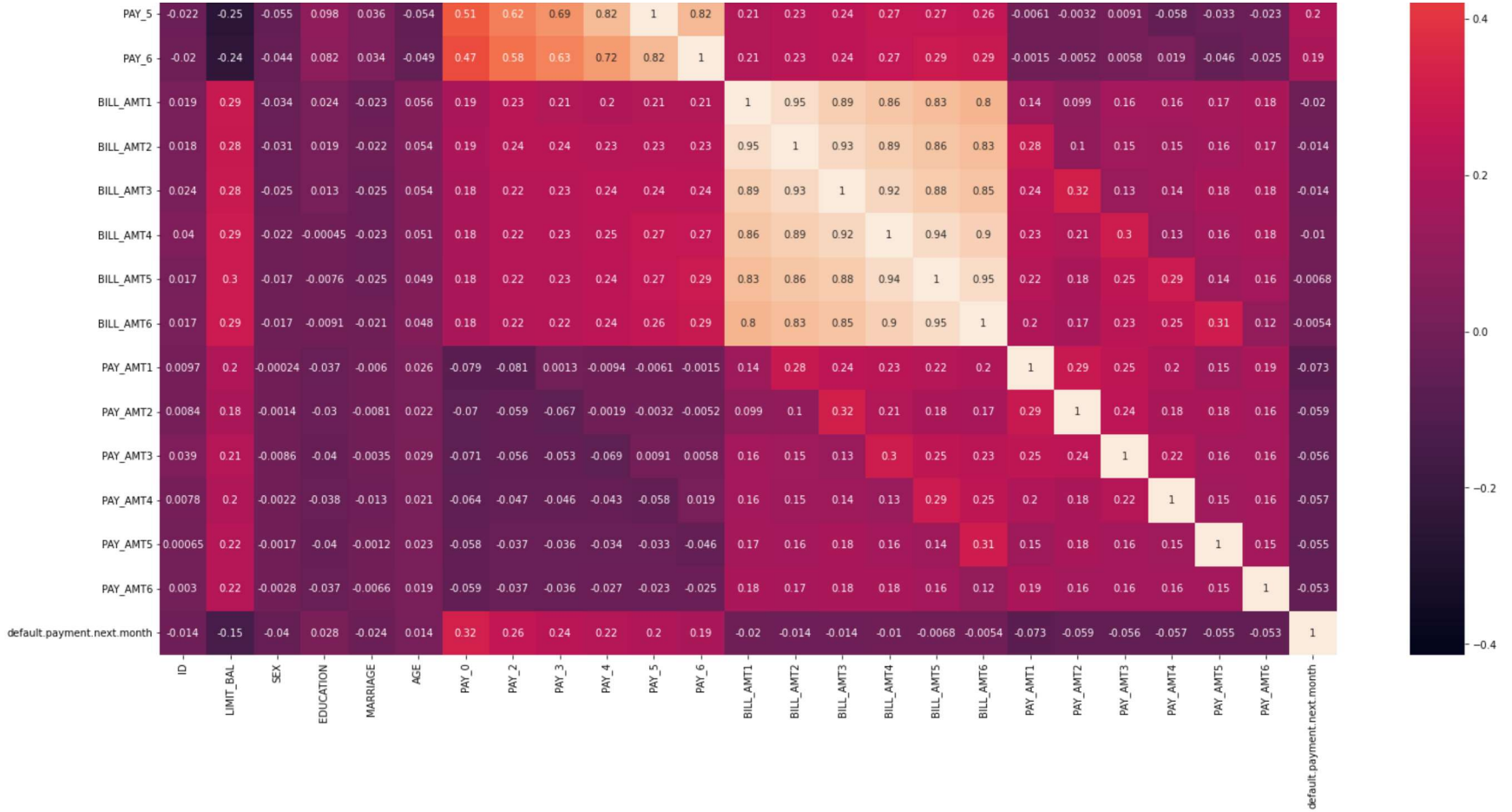
In [8]:

```
sns.boxplot('default.payment.next.month','LIMIT_BAL',data=dataset)
plt.show()
```



In [9]:

```
plt.subplots(figsize=(26,20))
corr = dataset.corr()
sns.heatmap(corr,annot=True)
plt.show()
```



In [10]:
```
x= dataset.drop(['default.payment.next.month','ID' ],1)
y = dataset['default.payment.next.month']
```

In [11]:
```
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.20,random_state=42)
```

Without Hyperparameter Tuning

In [12]:

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,classification_report,plot_roc_curve,plot_confusion_ma
rf_model = RandomForestClassifier()
rf_model.fit(X_train,Y_train)
y_pred_rf = rf_model.predict(X_test)
print("Training:",rf_model.score(X_train,Y_train))
print("Testing:",rf_model.score(X_test,Y_test))
```

```
Training: 0.999375
Testing: 0.8166666666666667
```

In [13]:

```
def res(y_valid):
    cm_log = confusion_matrix(Y_test,y_valid)
    ConfusionMatrixDisplay(cm_log).plot()
    print(classification_report(Y_test,y_valid))
```

In [ ]:

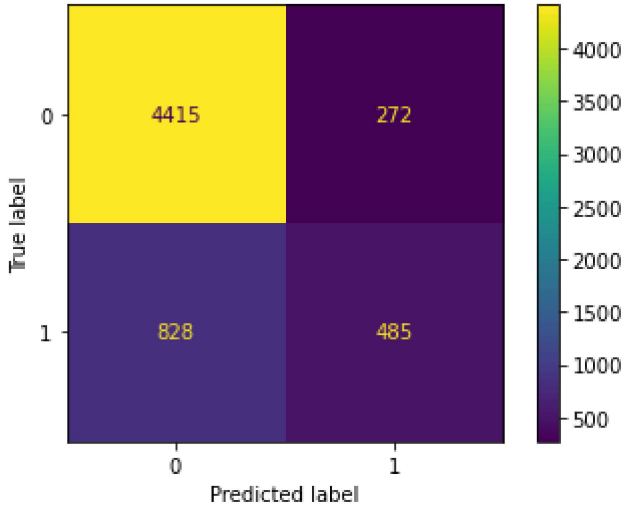
In [14]:

```
res(y_pred_rf)
```

```
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      4687
           1       0.64      0.37      0.47      1313

    accuracy                           0.82      6000
   macro avg       0.74      0.66      0.68      6000
weighted avg       0.80      0.82      0.80      6000
```



In [15]:
```
print(confusion_matrix(Y_test, y_pred_rf))
```

```
[[4415  272]
 [ 828  485]]
```

With Hyperparameter Tuning

Applying K-Fold cross validation to test performance on rf_model with Hyperparameter Tuning

K-Fold

Dividing dataset into 10 folds. The number 10 is totally arbitrary in this case.

In [16]:
```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=10, random_state=11, shuffle=True)
```

In [17]:
```
# Parameters for RF model
parameters = {"criterion" : ["gini", "entropy"],
              "class_weight" : ["balanced", "balanced_subsample", None],
              "max_features" : ["sqrt", "log2", None]
             }
```

In [18]:
```
# Constructing model
model = RandomForestClassifier(n_jobs=-1, warm_start=True)
```

In [19]:
```
from sklearn.model_selection import GridSearchCV
# Setting up grid search cross validation
searcher = GridSearchCV(estimator=model, param_grid=parameters, n_jobs=-1, cv=kfold, return_train_score=False)

# Training the model
searcher.fit(x, y)

# Printing the results
pd.set_option("display.max_columns", None)
pd.DataFrame(searcher.cv_results_).set_index("rank_test_score").sort_values(by="rank_test_score")
```

Out[19]:

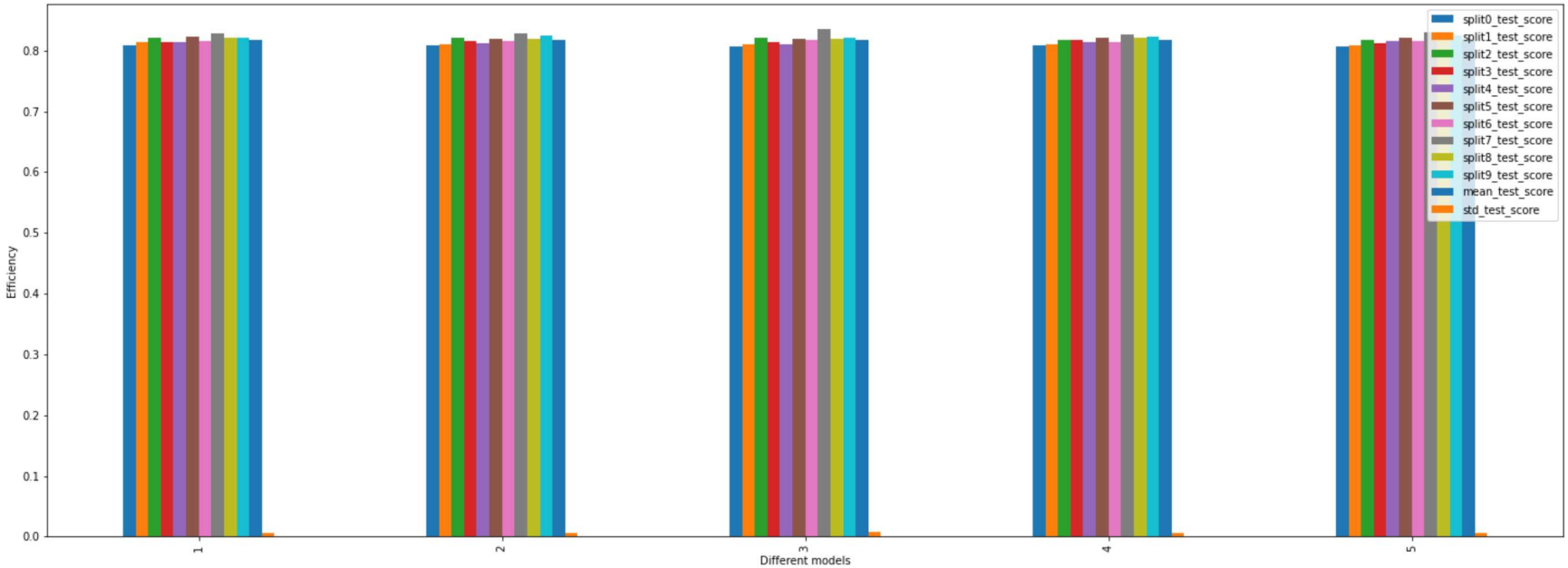
rank_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_class_weight	param_criterion	param_max_features	
1	16.731626	0.749763	3.434429	0.831129	None	gini	sqrt	{'class 'criter
2	105.414462	35.499394	1.846542	1.016091	None	entropy	None	{'class 'criteri
3	17.549488	0.725497	3.138971	0.970474	None	gini	log2	{'class 'criter
4	104.968587	12.482781	2.727023	0.665613	balanced_subsample	entropy	None	'balanc

rank_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_class_weight	param_criterion	param_max_features	
5	22.618840	1.037190	3.642943	1.563547	balanced_subsample	entropy	log2	'balanc
6	22.131058	2.287558	3.655122	1.348635	None	entropy	log2	{'class 'criteri
7	23.085424	1.574891	4.188843	0.573223	balanced	entropy	sqrt	'balan
8	15.377007	1.173935	2.501027	1.864834	balanced	gini	log2	'balan
9	104.793394	14.488110	3.533969	0.842728	balanced	entropy	None	'balan
10	26.094910	0.790070	3.758195	1.109503	balanced_subsample	entropy	sqrt	'balanc
11	18.746535	0.797454	4.271173	0.843659	balanced_subsample	gini	sqrt	'balanc
12	82.963625	7.943265	3.437119	2.029441	None	gini	None	{'class 'criter
13	17.381095	1.948746	3.479343	1.509687	balanced_subsample	gini	log2	'balanc
14	24.466813	1.420629	3.636772	1.078947	None	entropy	sqrt	{'class 'criteri
15	14.284942	1.817047	1.629520	1.881844	balanced	gini	sqrt	'balan

rank_test_score	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_class_weight	param_criterion	param_max_features	
16	20.677965	0.749326	4.375654	0.632643	balanced	entropy	log2	'balan
17	71.745545	6.712729	3.565217	0.905008	balanced_subsample	gini	None	'balanc
18	69.872097	5.887195	4.143349	0.689454	balanced	gini	None	'balan

In [20]:
```
pd.DataFrame(searcher.cv_results_).sort_values(by="rank_test_score").set_index("rank_test_score").drop(["mean_fit_time", "
```

Out[20]:
```
<AxesSubplot:xlabel='Different models', ylabel='Efficiency'>
```


split0_test_score
split1_test_score
split2_test_score
split3_test_score
split4_test_score
split5_test_score
split6_test_score
split7_test_score
split8_test_score
split9_test_score
mean_test_score
std_test_score
Efficiency
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1
2
3
4
5
Different models

In [ ]:

In [ ]: