loan-classification

June 21, 2024

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

[3]:
```
df = pd.read_csv('loan.csv')
df
```

[3]:
```
    age  gender    occupation education_level marital_status  income  \
0    32    Male      Engineer      Bachelor's        Married   85000
1    45  Female       Teacher        Master's         Single   62000
2    28    Male       Student     High School         Single   25000
3    51  Female       Manager      Bachelor's        Married  105000
4    36    Male    Accountant      Bachelor's        Married   75000
..  …       …             …             …              …      …
56   39    Male     Architect        Master's        Married  100000
57   25  Female  Receptionist     High School         Single   32000
58   43    Male        Banker      Bachelor's        Married   95000
59   30  Female        Writer        Master's         Single   55000
60   38    Male          Chef    Associate's        Married   65000

    credit_score loan_status
0            720    Approved
1            680    Approved
2            590      Denied
3            780    Approved
4            710    Approved
..           …          …
56           770    Approved
57           570      Denied
58           760    Approved
59           650    Approved
60           700    Approved

[61 rows x 8 columns]
```

Data cleaning

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61 entries, 0 to 60
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   age              61 non-null     int64
 1   gender           61 non-null     object
 2   occupation       61 non-null     object
 3   education_level  61 non-null     object
 4   marital_status   61 non-null     object
 5   income           61 non-null     int64
 6   credit_score     61 non-null     int64
 7   loan_status      61 non-null     object
dtypes: int64(3), object(5)
memory usage: 3.9+ KB
```

[8]:
```
df.duplicated().sum()
```

[8]:
```
0
```

EDA

[11]:
```
df['education_level'].unique()
```

[11]:
```
array(["Bachelor's", "Master's", 'High School', "Associate's", 'Doctoral'],
      dtype=object)
```

[13]:
```
df['marital_status'].unique()
```

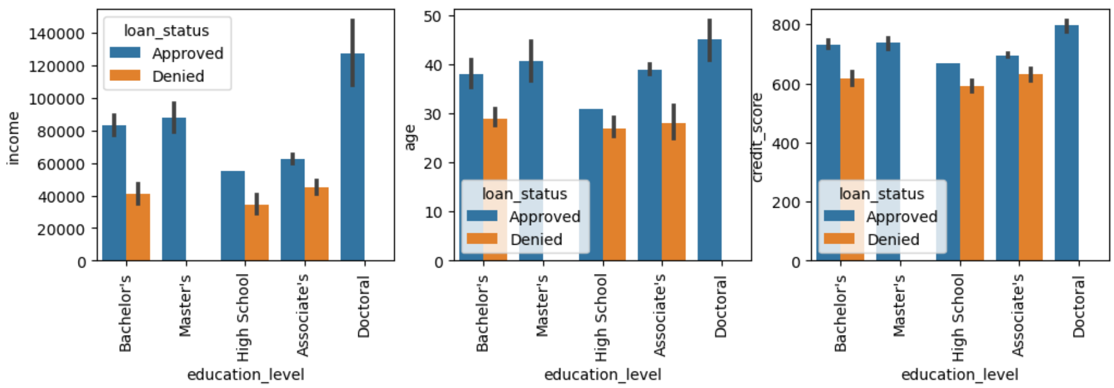
[13]:
```
array(['Married', 'Single'], dtype=object)
```

[15]:
```
df['loan_status'].value_counts()
```

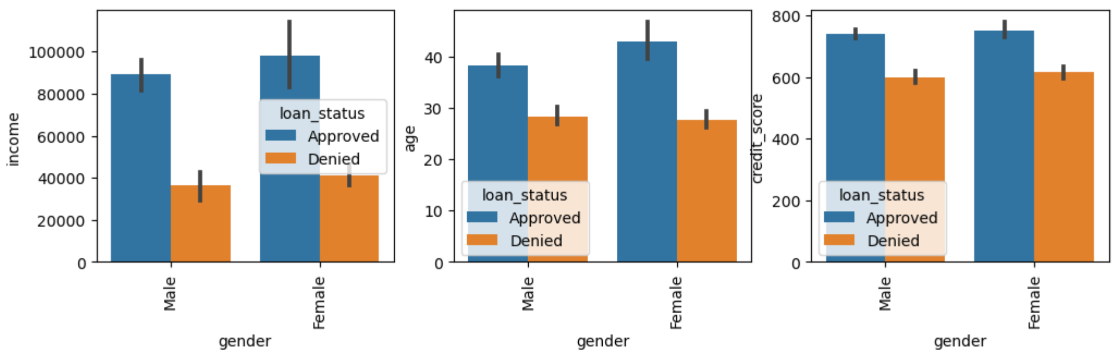
[15]:
```
loan_status
Approved    45
Denied      16
Name: count, dtype: int64
```

[17]:
```
plt.figure(figsize=(12,3))
plt.subplot(131)
sns.barplot(df,x='education_level',y='income',hue='loan_status')
plt.xticks(rotation='vertical')
plt.subplot(132)
sns.barplot(df,x='education_level',y='age',hue='loan_status')
plt.xticks(rotation='vertical')
plt.subplot(133)
```

```
sns.barplot(df,x='education_level',y='credit_score',hue='loan_status')
plt.xticks(rotation='vertical')
plt.show()
```

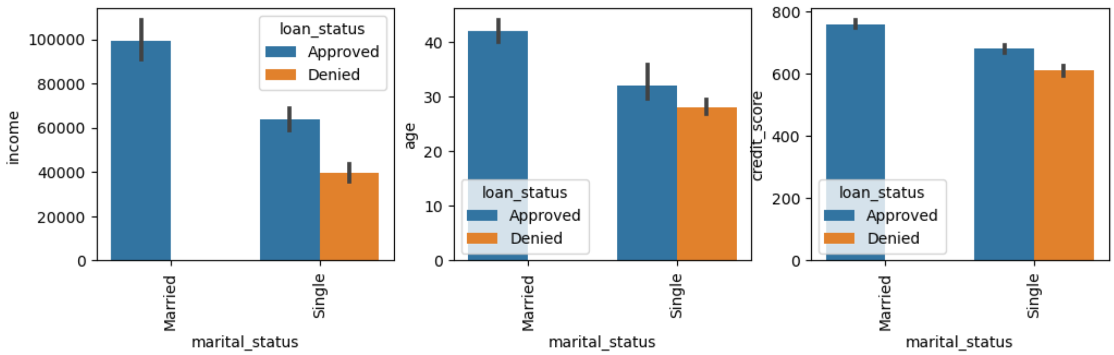


[18]:
```
plt.figure(figsize=(12,3))
plt.subplot(131)
sns.barplot(df,x='gender',y='income',hue='loan_status')
plt.xticks(rotation='vertical')
plt.subplot(132)
sns.barplot(df,x='gender',y='age',hue='loan_status')
plt.xticks(rotation='vertical')
plt.subplot(133)
sns.barplot(df,x='gender',y='credit_score',hue='loan_status')
plt.xticks(rotation='vertical')
plt.show()
```



[19]:
```
plt.figure(figsize=(12,3))
plt.subplot(131)
sns.barplot(df,x='marital_status',y='income',hue='loan_status')
plt.xticks(rotation='vertical')
```

```
plt.subplot(132)
sns.barplot(df,x='marital_status',y='age',hue='loan_status')
plt.xticks(rotation='vertical')
plt.subplot(133)
sns.barplot(df,x='marital_status',y='credit_score',hue='loan_status')
plt.xticks(rotation='vertical')
plt.show()
```



[20]:
```
# education_level ->master's & doctaral ->loan approved
#marital_status -> married ->loan approved
```

Feature Engineering

[22]:
```
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]
```

[28]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

[29]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder,OrdinalEncoder

trf = ColumnTransformer([
    ␣
↪('ohe',OneHotEncoder(drop='first',sparse_output=False),['gender','occupation','marital_stat
    ('oe',OrdinalEncoder(categories=[["High School","Bachelor's", "Master's",␣
↪"Associate's", "Doctoral"]]),['education_level'])
],remainder='passthrough')
```

[30]:
```
X = trf.fit_transform(X)
```

[34]:
```
X.shape
```

[34]: (61, 43)

[36]:
```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X = scale.fit_transform(X)
```

Model Building & performance improvement

[39]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,VotingClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score,precision_score,classification_report
```

[40]:
```
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=2)
```

[42]:
```
X_train.shape
```

[42]: (48, 43)

[45]:
```
lr = LogisticRegression(solver='liblinear',penalty='l1',random_state=2)
dt =␣
 ↪DecisionTreeClassifier(criterion='gini',splitter='best',max_depth=5,random_state=2)
knn = KNeighborsClassifier(n_neighbors=5 , weights='distance')
svc = SVC(kernel='rbf',random_state=2)
rf = RandomForestClassifier(n_estimators=50,max_depth=4,max_samples=0.
 ↪5,bootstrap=True,random_state=2)
gbc = GradientBoostingClassifier(max_leaf_nodes=8,random_state=2)
xgb = xgb.XGBClassifier(booster='gbtree',learning_rate=0.1,random_state=2)
```

[47]:
```
def performance(clf,X_train,y_train,X_test,y_test):
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred)

    return accuracy,precision
```

[81]:
```
print('LR',performance(lr,X_train,y_train,X_test,y_test))
print('DT',performance(dt,X_train,y_train,X_test,y_test))
print('KNN',performance(knn,X_train,y_train,X_test,y_test))
```

```
print('SVC',performance(svc,X_train,y_train,X_test,y_test))
print('RF',performance(rf,X_train,y_train,X_test,y_test))
print('GBC',performance(gbc,X_train,y_train,X_test,y_test))
print('XGB',performance(xgb,X_train,y_train,X_test,y_test))
```

```
LR (0.9230769230769231, 1.0)
DT (1.0, 1.0)
KNN (0.8461538461538461, 0.0)
SVC (0.8461538461538461, 0.0)
RF (1.0, 1.0)
```

```
D:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\ProgramData\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
GBC (1.0, 1.0)
XGB (1.0, 1.0)
```

[83]:
```
from sklearn.model_selection import cross_val_score
print('LR',np.mean(cross_val_score(lr,X,y,scoring='accuracy',cv=10)))
print('DT',np.mean(cross_val_score(dt,X,y,scoring='accuracy',cv=10)))
print('KNN',np.mean(cross_val_score(knn,X,y,scoring='accuracy',cv=10)))
print('SVC',np.mean(cross_val_score(svc,X,y,scoring='accuracy',cv=10)))
print('RF',np.mean(cross_val_score(rf,X,y,scoring='accuracy',cv=10)))
print('GBC',np.mean(cross_val_score(gbc,X,y,scoring='accuracy',cv=10)))
print('XGB',np.mean(cross_val_score(xgb,X,y,scoring='accuracy',cv=10)))
```

```
LR 0.9666666666666668
DT 0.9833333333333334
KNN 0.8857142857142858
SVC 0.8857142857142858
RF 1.0
GBC 0.9833333333333334
XGB 1.0
```

[85]:
```
estimators = [('rf',rf),('dt',dt),('xgb',xgb)]
vc = VotingClassifier(estimators=estimators , voting='soft')
```

[87]:
```
vc.fit(X_train,y_train)
y_pred = vc.predict(X_test)
accuracy_score(y_test,y_pred)
```

[87]: 1.0

[89]:
```
np.mean(cross_val_score(vc,X,y,scoring='accuracy',cv=10))
```

[89]: 1.0

[91]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00         2

    accuracy                           1.00        13
   macro avg       1.00      1.00      1.00        13
weighted avg       1.00      1.00      1.00        13
```

[93]:
```
from sklearn.metrics import confusion_matrix,recall_score,precision_score
```

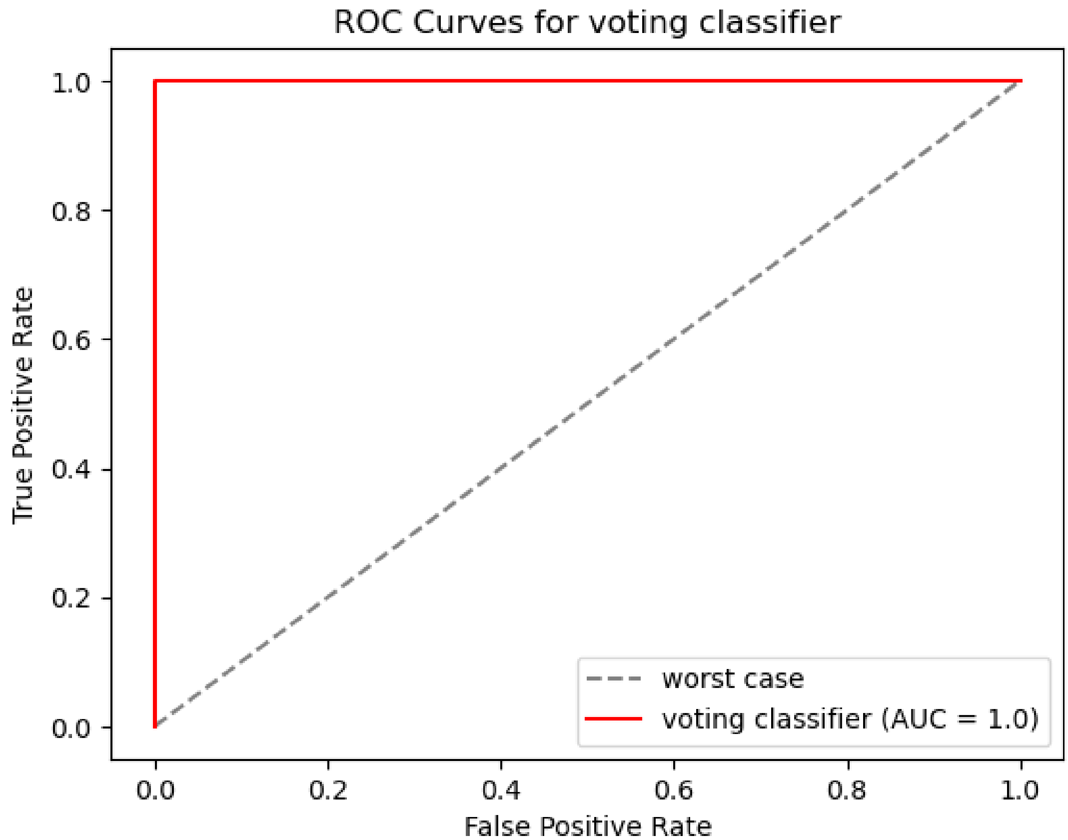
[95]:
```
print('recall_score\n',recall_score(y_test,y_pred))
print('precision_score\n',precision_score(y_test,y_pred))
print('confusion_matrix\n',confusion_matrix(y_test,y_pred))
```

```
recall_score
 1.0
precision_score
 1.0
confusion_matrix
 [[11  0]
 [ 0  2]]
```

[103]:
```
from sklearn.metrics import roc_curve,auc
```

[125]:
```
vc_prob = vc.predict_proba(X_test)[:,1]
fpr,tpr,_ = roc_curve(y_test,lr_prob)
roc_auc = auc(fpr , tpr)
```

[127]:
```
plt.plot([0,1] , [0,1] , '--' , color='grey' , label='worst case')
plt.plot(fpr , tpr , color='red' , label='voting classifier (AUC = {})'.
  ↪format(roc_auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for voting classifier')
plt.legend()
plt.show()
```


ROC Curves for voting classifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
worst case
voting classifier (AUC = 1.0)

```
[ ]:
```