# urn-prediction-logistic-regression

May 25, 2024

[33]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix,␣
↪roc_curve, precision_score, recall_score, precision_recall_curve
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

[2]:
```
df = pd.read_csv('/content/churn_prediction.csv')
```

[3]:
```
df.head()
```

[3]:
```
   customer_id  vintage  age gender  dependents     occupation    city  \
0            1     3135   66   Male         0.0  self_employed   187.0
1            2      310   35   Male         0.0  self_employed     NaN
2            4     2356   31   Male         0.0       salaried   146.0
3            5      478   90    NaN         NaN  self_employed  1020.0
4            6     2531   42   Male         2.0  self_employed  1494.0

   customer_nw_category  branch_code  days_since_last_transaction  …  \
0                     2          755                        224.0  …
1                     2         3214                         60.0  …
2                     2           41                          NaN  …
3                     2          582                        147.0  …
4                     3          388                         58.0  …

   previous_month_end_balance  average_monthly_balance_prevQ  \
0                     1458.71                        1458.71
1                     8704.66                        7799.26
2                     5815.29                        4910.17
3                     2291.91                        2084.54
```

```
4                     1401.72                         1643.31

   average_monthly_balance_prevQ2  current_month_credit  \
0                         1449.07                  0.20
1                        12419.41                  0.56
2                         2815.94                  0.61
3                         1006.54                  0.47
4                         1871.12                  0.33

   previous_month_credit  current_month_debit  previous_month_debit  \
0                   0.20                 0.20                  0.20
1                   0.56              5486.27                100.56
2                   0.61              6046.73                259.23
3                   0.47                 0.47               2143.33
4                 714.61               588.62               1538.06

   current_month_balance  previous_month_balance  churn
0                1458.71                 1458.71      0
1                6496.78                 8787.61      0
2                5006.28                 5070.14      0
3                2291.91                 1669.79      1
4                1157.15                 1677.16      1

[5 rows x 21 columns]
```

[4]:
```
df.columns
```

[4]:
```
Index(['customer_id', 'vintage', 'age', 'gender', 'dependents', 'occupation',
       'city', 'customer_nw_category', 'branch_code',
       'days_since_last_transaction', 'current_balance',
       'previous_month_end_balance', 'average_monthly_balance_prevQ',
       'average_monthly_balance_prevQ2', 'current_month_credit',
       'previous_month_credit', 'current_month_debit', 'previous_month_debit',
       'current_month_balance', 'previous_month_balance', 'churn'],
      dtype='object')
```

[5]:
```
df.describe()
```

[5]:
```
        customer_id       vintage           age    dependents          city  \
count  28382.000000  28382.000000  28382.000000  25919.000000  27579.000000
mean   15143.508667   2364.336446     48.208336      0.347236    796.109576
std     8746.454456   1610.124506     17.807163      0.997661    432.872102
min        1.000000    180.000000      1.000000      0.000000      0.000000
25%     7557.250000   1121.000000     36.000000      0.000000    409.000000
50%    15150.500000   2018.000000     46.000000      0.000000    834.000000
75%    22706.750000   3176.000000     60.000000      0.000000   1096.000000
max    30301.000000  12899.000000     90.000000     52.000000   1649.000000
```

| | customer_nw_category | branch_code | days_since_last_transaction | \ |
|---|---|---|---|---|
| count | 28382.000000 | 28382.000000 | 25159.000000 | |
| mean | 2.225530 | 925.975019 | 69.997814 | |
| std | 0.660443 | 937.799129 | 86.341098 | |
| min | 1.000000 | 1.000000 | 0.000000 | |
| 25% | 2.000000 | 176.000000 | 11.000000 | |
| 50% | 2.000000 | 572.000000 | 30.000000 | |
| 75% | 3.000000 | 1440.000000 | 95.000000 | |
| max | 3.000000 | 4782.000000 | 365.000000 | |

| | current_balance | previous_month_end_balance | \ |
|---|---|---|---|
| count | 2.838200e+04 | 2.838200e+04 | |
| mean | 7.380552e+03 | 7.495771e+03 | |
| std | 4.259871e+04 | 4.252935e+04 | |
| min | -5.503960e+03 | -3.149570e+03 | |
| 25% | 1.784470e+03 | 1.906000e+03 | |
| 50% | 3.281255e+03 | 3.379915e+03 | |
| 75% | 6.635820e+03 | 6.656535e+03 | |
| max | 5.905904e+06 | 5.740439e+06 | |

| | average_monthly_balance_prevQ | average_monthly_balance_prevQ2 | \ |
|---|---|---|---|
| count | 2.838200e+04 | 2.838200e+04 | |
| mean | 7.496780e+03 | 7.124209e+03 | |
| std | 4.172622e+04 | 4.457581e+04 | |
| min | 1.428690e+03 | -1.650610e+04 | |
| 25% | 2.180945e+03 | 1.832507e+03 | |
| 50% | 3.542865e+03 | 3.359600e+03 | |
| 75% | 6.666887e+03 | 6.517960e+03 | |
| max | 5.700290e+06 | 5.010170e+06 | |

| | current_month_credit | previous_month_credit | current_month_debit | \ |
|---|---|---|---|---|
| count | 2.838200e+04 | 2.838200e+04 | 2.838200e+04 | |
| mean | 3.433252e+03 | 3.261694e+03 | 3.658745e+03 | |
| std | 7.707145e+04 | 2.968889e+04 | 5.198542e+04 | |
| min | 1.000000e-02 | 1.000000e-02 | 1.000000e-02 | |
| 25% | 3.100000e-01 | 3.300000e-01 | 4.100000e-01 | |
| 50% | 6.100000e-01 | 6.300000e-01 | 9.193000e+01 | |
| 75% | 7.072725e+02 | 7.492350e+02 | 1.360435e+03 | |
| max | 1.226985e+07 | 2.361808e+06 | 7.637857e+06 | |

| | previous_month_debit | current_month_balance | previous_month_balance | \ |
|---|---|---|---|---|
| count | 2.838200e+04 | 2.838200e+04 | 2.838200e+04 | |
| mean | 3.339761e+03 | 7.451133e+03 | 7.495177e+03 | |
| std | 2.430111e+04 | 4.203394e+04 | 4.243198e+04 | |
| min | 1.000000e-02 | -3.374180e+03 | -5.171920e+03 | |
| 25% | 4.100000e-01 | 1.996765e+03 | 2.074407e+03 | |

```
50%          1.099600e+02          3.447995e+03          3.465235e+03
75%          1.357553e+03          6.667958e+03          6.654693e+03
max          1.414168e+06          5.778185e+06          5.720144e+06

              churn
count  28382.000000
mean       0.185329
std        0.388571
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28382 entries, 0 to 28381
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   customer_id                     28382 non-null  int64
 1   vintage                         28382 non-null  int64
 2   age                             28382 non-null  int64
 3   gender                          27857 non-null  object
 4   dependents                      25919 non-null  float64
 5   occupation                      28302 non-null  object
 6   city                            27579 non-null  float64
 7   customer_nw_category            28382 non-null  int64
 8   branch_code                     28382 non-null  int64
 9   days_since_last_transaction     25159 non-null  float64
 10  current_balance                 28382 non-null  float64
 11  previous_month_end_balance      28382 non-null  float64
 12  average_monthly_balance_prevQ   28382 non-null  float64
 13  average_monthly_balance_prevQ2  28382 non-null  float64
 14  current_month_credit            28382 non-null  float64
 15  previous_month_credit           28382 non-null  float64
 16  current_month_debit             28382 non-null  float64
 17  previous_month_debit            28382 non-null  float64
 18  current_month_balance           28382 non-null  float64
 19  previous_month_balance          28382 non-null  float64
 20  churn                           28382 non-null  int64
dtypes: float64(13), int64(6), object(2)
memory usage: 4.5+ MB
```

### 0.0.1 Handling Missing Values

[7]:
```
df.isnull().sum()
```

[7]:
```
customer_id                          0
vintage                              0
age                                  0
gender                             525
dependents                        2463
occupation                          80
city                               803
customer_nw_category                 0
branch_code                          0
days_since_last_transaction       3223
current_balance                      0
previous_month_end_balance           0
average_monthly_balance_prevQ        0
average_monthly_balance_prevQ2       0
current_month_credit                 0
previous_month_credit                0
current_month_debit                  0
previous_month_debit                 0
current_month_balance                0
previous_month_balance               0
churn                                0
dtype: int64
```

[8]:
```
df['gender'].replace({'Male':1,'Female':0,},inplace=True)
```

[9]:
```
df['gender'].value_counts()
```

[9]:
```
gender
1.0    16548
0.0    11309
Name: count, dtype: int64
```

[10]:
```
df['gender']
```

[10]:
```
0        1.0
1        1.0
2        1.0
3        NaN
4        1.0
        …
28377    0.0
28378    0.0
28379    1.0
```

```
28380    1.0
28381    1.0
Name: gender, Length: 28382, dtype: float64
```

[11]:
```
mode_gender = df['gender'].mode()[0]
```

[12]:
```
df['gender'].fillna(mode_gender,inplace=True)
```

[13]:
```
df['gender']
```

[13]:
```
0        1.0
1        1.0
2        1.0
3        1.0
4        1.0
        …
28377    0.0
28378    0.0
28379    1.0
28380    1.0
28381    1.0
Name: gender, Length: 28382, dtype: float64
```

[14]:
```
df['gender'].value_counts()
```

[14]:
```
gender
1.0    17073
0.0    11309
Name: count, dtype: int64
```

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

[15]:
```
dependents
0.0     21435
2.0      2150
1.0      1395
3.0       701
4.0       179
5.0        41
6.0         8
7.0         3
9.0         1
52.0        1
36.0        1
50.0        1
8.0         1
25.0        1
```

```
32.0        1
Name: count, dtype: int64
```

[16]:
```
df['dependents'] = df['dependents'].fillna(0)
```

[17]:
```
df['dependents'].value_counts()
```

[17]:
```
dependents
0.0     23898
2.0      2150
1.0      1395
3.0       701
4.0       179
5.0        41
6.0         8
7.0         3
9.0         1
52.0        1
36.0        1
50.0        1
8.0         1
25.0        1
32.0        1
Name: count, dtype: int64
```

[18]:
```
df['occupation'].fillna('self_employed',inplace=True)
```

[19]:
```
df['city'].mode()
```

[19]:
```
0    1020.0
Name: city, dtype: float64
```

[20]:
```
df['city'].fillna(df['city'].mode()[0],inplace=True)
```

[21]:
```
df['city'].isnull().sum()
```

[21]:
```
0
```

[22]:
```
df['days_since_last_transaction'].fillna(df['days_since_last_transaction'].
↪mode()[0],inplace=True)
```

[23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28382 entries, 0 to 28381
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype
```

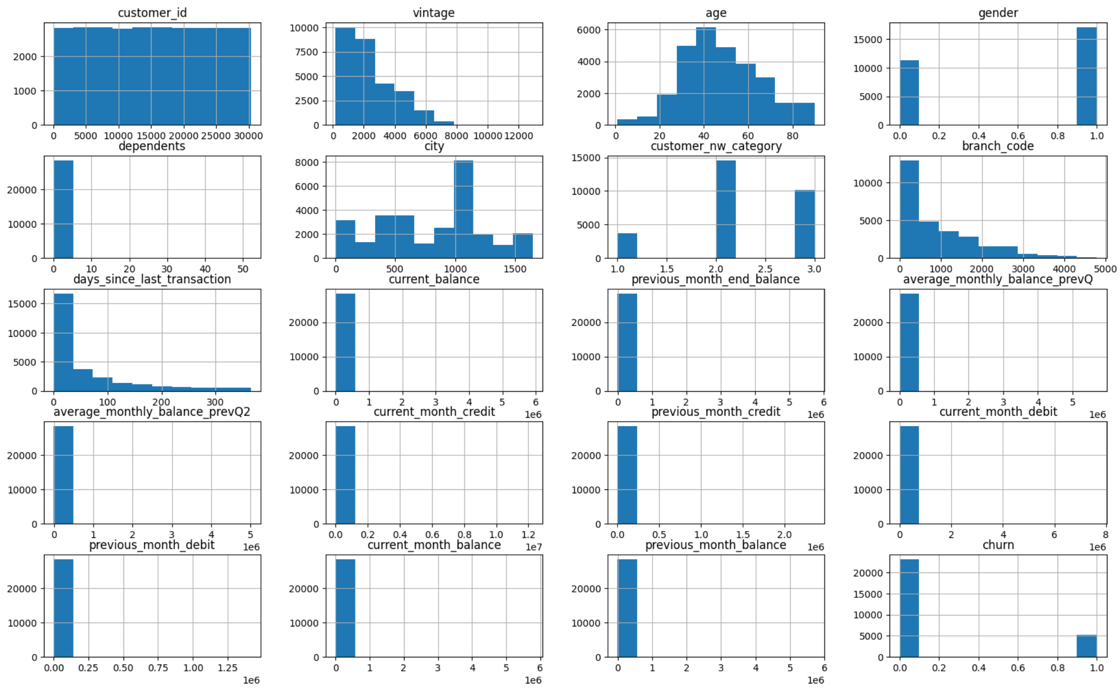
```
---  ------                          --------------  -----
 0   customer_id                     28382 non-null  int64
 1   vintage                         28382 non-null  int64
 2   age                             28382 non-null  int64
 3   gender                          28382 non-null  float64
 4   dependents                      28382 non-null  float64
 5   occupation                      28382 non-null  object
 6   city                            28382 non-null  float64
 7   customer_nw_category            28382 non-null  int64
 8   branch_code                     28382 non-null  int64
 9   days_since_last_transaction     28382 non-null  float64
 10  current_balance                 28382 non-null  float64
 11  previous_month_end_balance      28382 non-null  float64
 12  average_monthly_balance_prevQ   28382 non-null  float64
 13  average_monthly_balance_prevQ2  28382 non-null  float64
 14  current_month_credit            28382 non-null  float64
 15  previous_month_credit           28382 non-null  float64
 16  current_month_debit             28382 non-null  float64
 17  previous_month_debit            28382 non-null  float64
 18  current_month_balance           28382 non-null  float64
 19  previous_month_balance          28382 non-null  float64
 20  churn                           28382 non-null  int64
dtypes: float64(14), int64(6), object(1)
memory usage: 4.5+ MB
```

[24]:
```
df.isnull().sum()
```

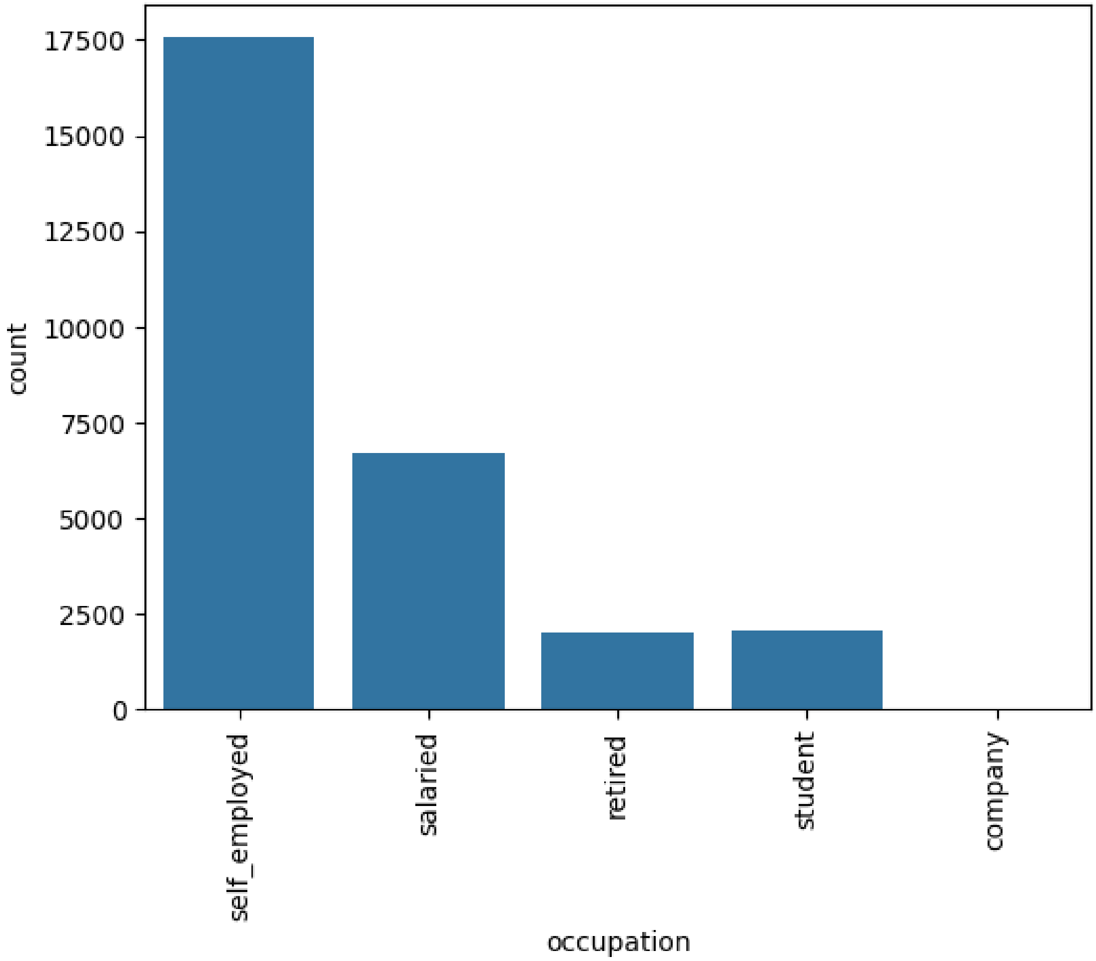
[24]:
```
customer_id                       0
vintage                           0
age                               0
gender                            0
dependents                        0
occupation                        0
city                              0
customer_nw_category              0
branch_code                       0
days_since_last_transaction       0
current_balance                   0
previous_month_end_balance        0
average_monthly_balance_prevQ     0
average_monthly_balance_prevQ2    0
current_month_credit              0
previous_month_credit             0
current_month_debit               0
previous_month_debit              0
current_month_balance             0
previous_month_balance            0
```

```
churn                                 0
dtype: int64
```

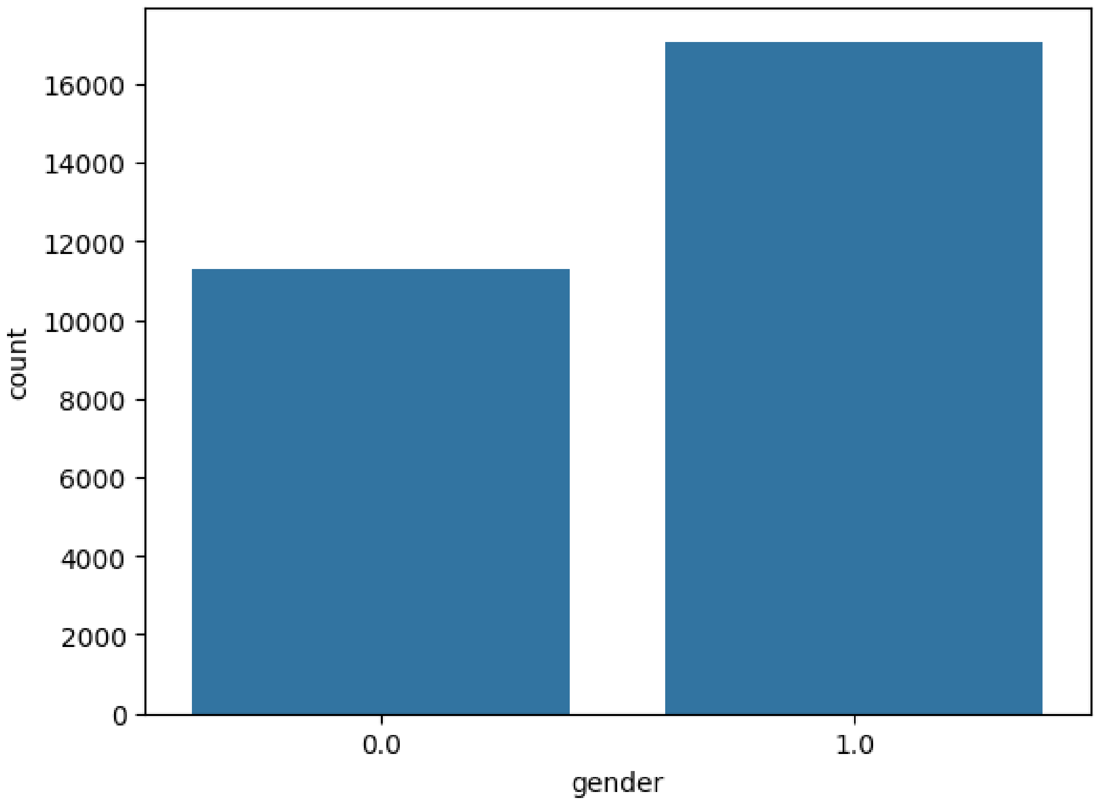
[25]:
```python
df.hist(figsize=(20, 12))
plt.show()
```



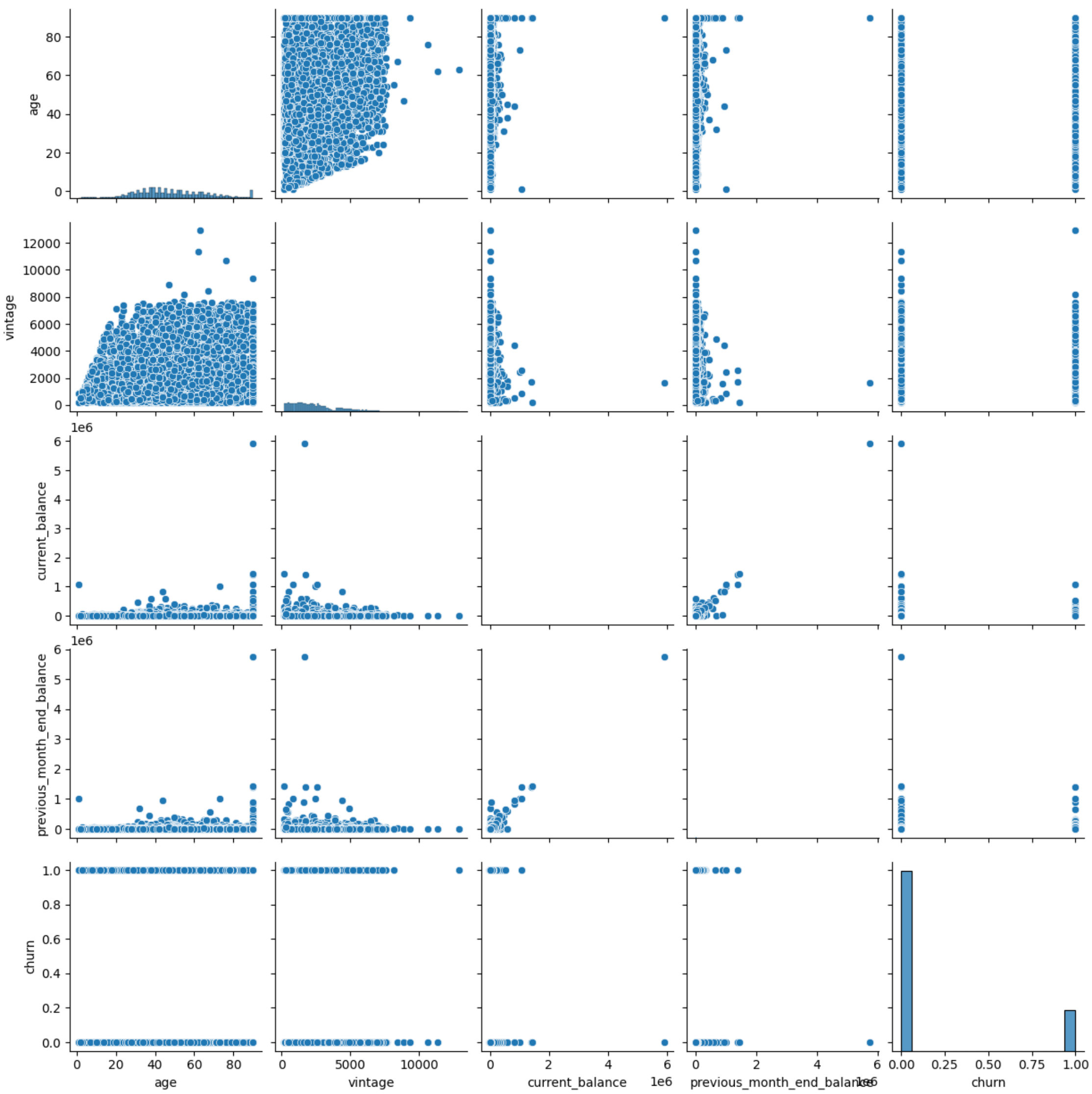

[26]:
```python
sns.countplot(data=df, x='occupation')
plt.xticks(rotation=90)
plt.show()
```



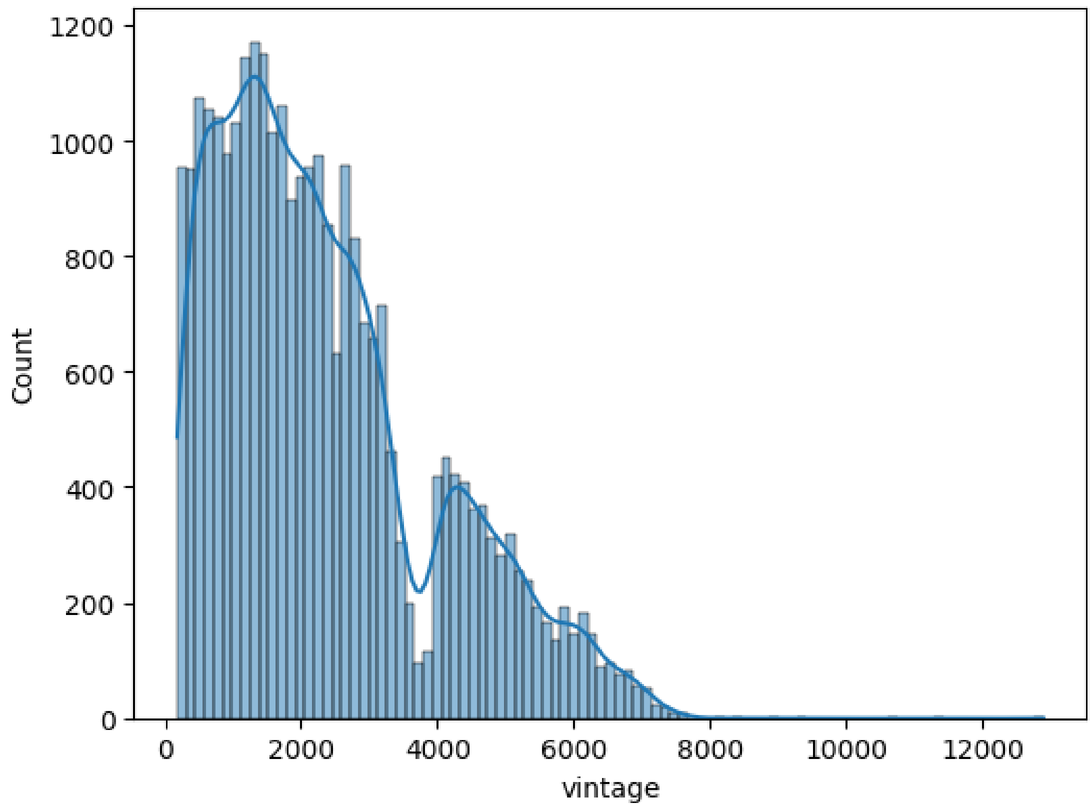

[27]:
```
sns.countplot(data=df, x='gender')
plt.show()
```



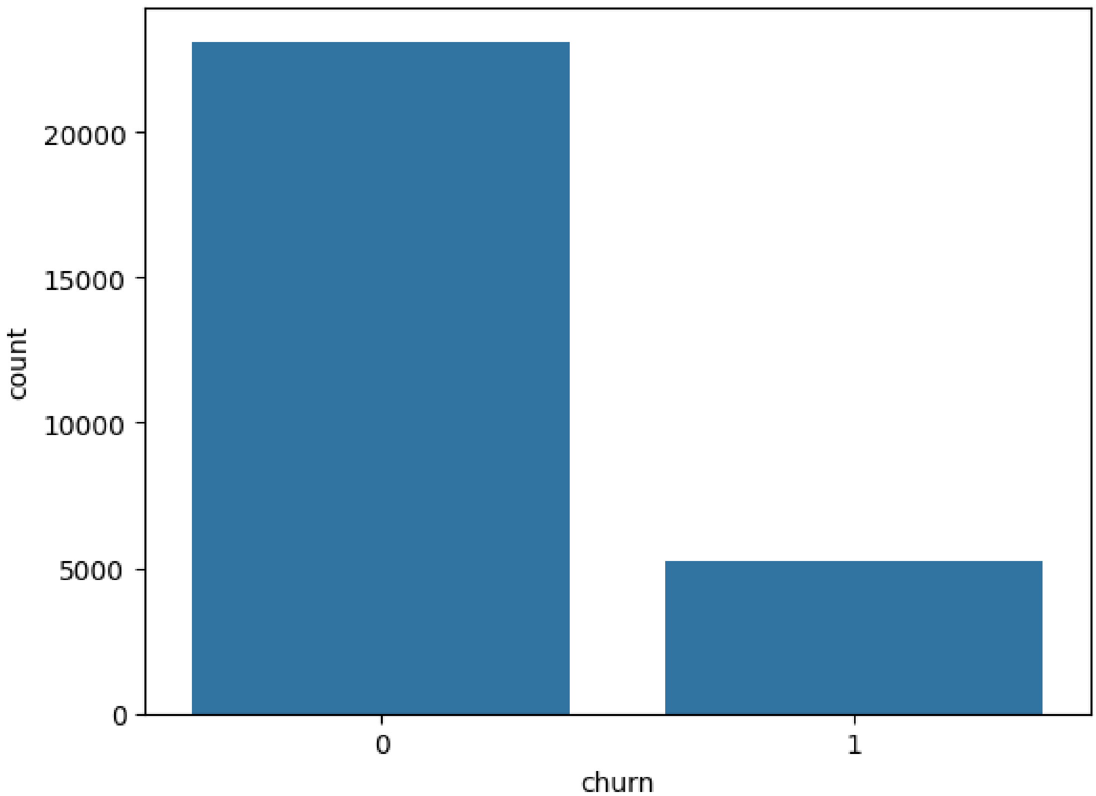

[28]:
```
sns.pairplot(df[['age', 'vintage', 'current_balance',␣
 ↪'previous_month_end_balance', 'churn']])
plt.show()
```



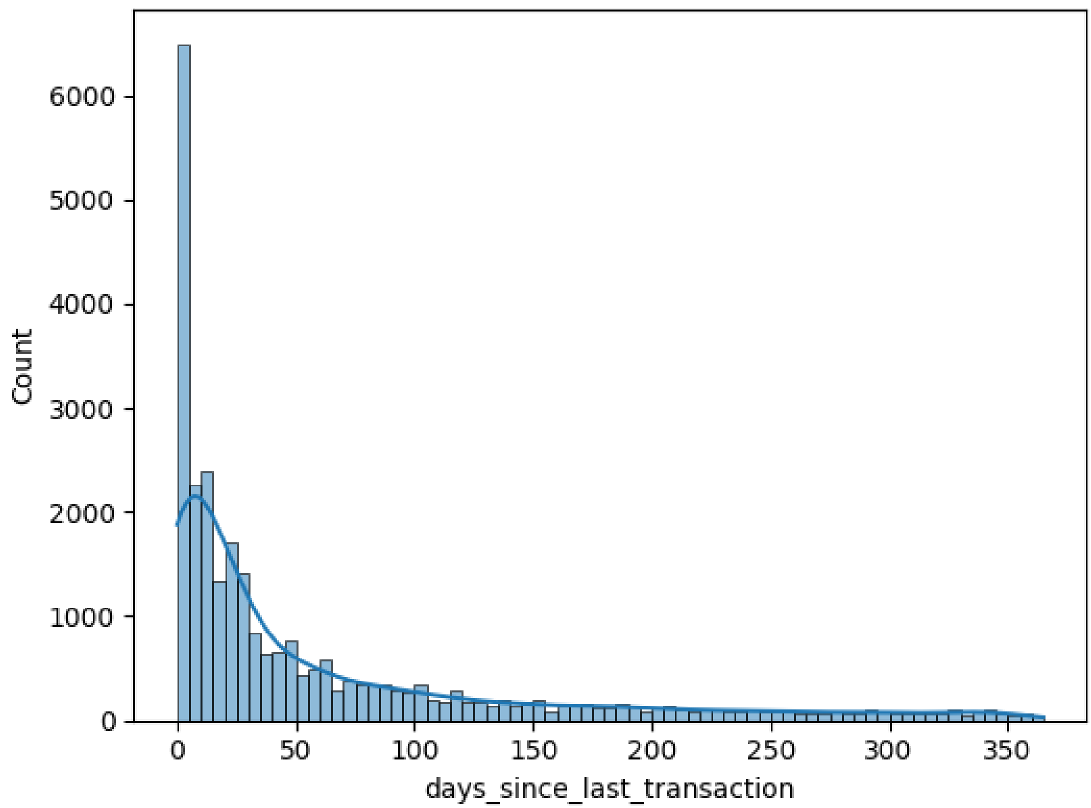

[29]:
```
sns.histplot(df['vintage'], kde=True)
plt.show()
```




[30]:
```
sns.countplot(data=df, x='churn')
plt.show()
```




[31]:
```
sns.histplot(df['days_since_last_transaction'], kde=True)
plt.show()
```




[32]:
```
df = pd.concat([df,pd.get_dummies(df['occupation'],prefix =␣
↪str('occupation'),prefix_sep='_')],axis = 1)
```

[34]:
```
num_cols = ['customer_nw_category', 'current_balance',
            'previous_month_end_balance', 'average_monthly_balance_prevQ2',␣
↪'average_monthly_balance_prevQ',
            'current_month_credit','previous_month_credit',␣
↪'current_month_debit',
            'previous_month_debit','current_month_balance',␣
↪'previous_month_balance']
for i in num_cols:
    df[i] = np.log(df[i] + 17000)

std = StandardScaler()
scaled = std.fit_transform(df[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)
```

[35]:
```
df_df_og = df.copy()
df = df.drop(columns = num_cols,axis = 1)
df = df.merge(scaled,left_index=True,right_index=True,how = "left")
```

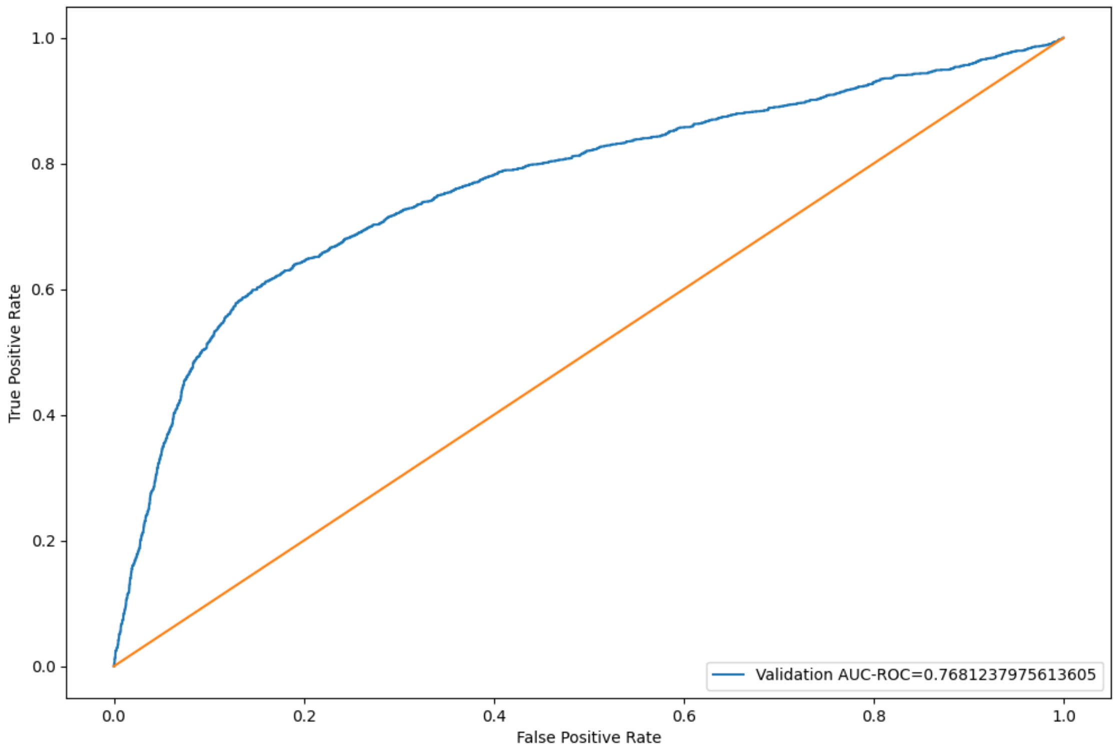
[36]:
```
y_all = df.churn
df = df.drop(['churn','customer_id','occupation'],axis = 1)
```

[37]:
```
baseline_cols = ['current_month_debit',␣
↪'previous_month_debit','current_balance','previous_month_end_balance','vintage'
                 ,'occupation_retired',␣
↪'occupation_salaried','occupation_self_employed', 'occupation_student']
```

[39]:
```
df_baseline = df[baseline_cols]
xtrain, xtest, ytrain, ytest = train_test_split(df_baseline,y_all,test_size=1/
↪3, random_state=11, stratify = y_all)
```

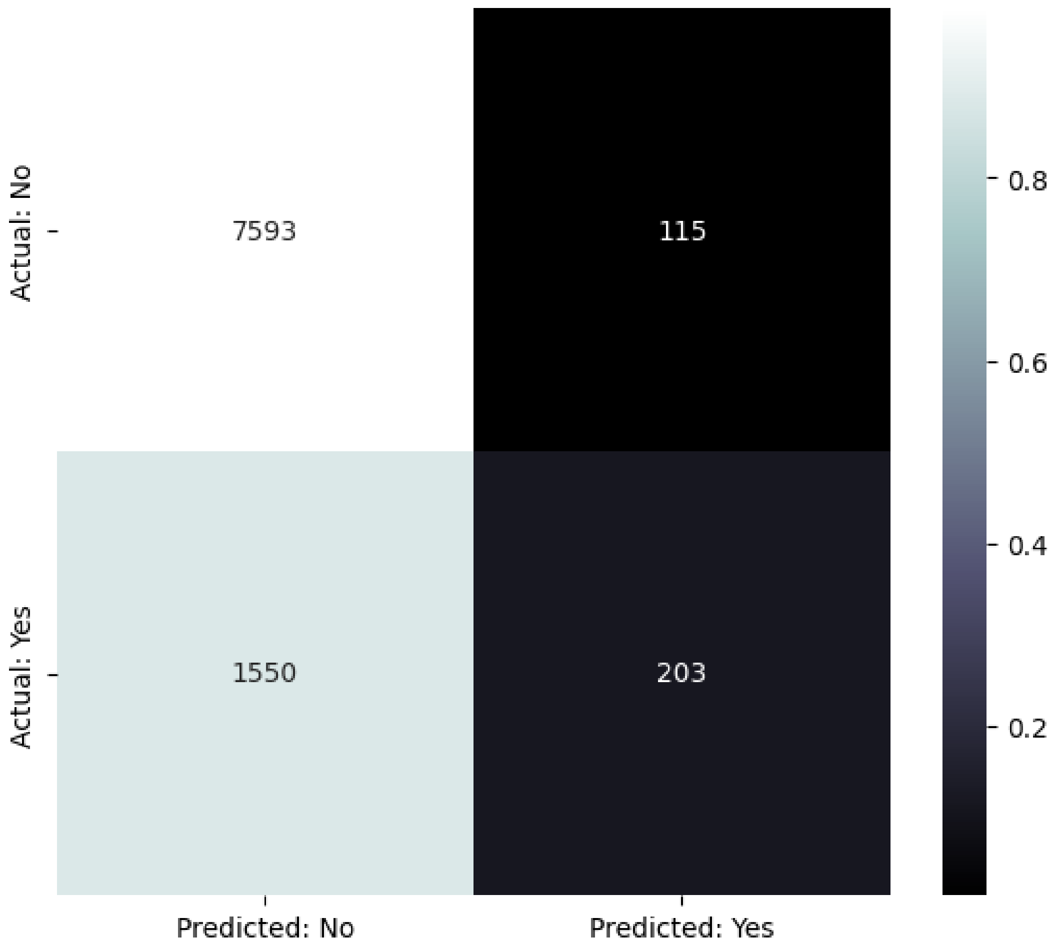
[40]:
```
model = LogisticRegression()
model.fit(xtrain,ytrain)
pred = model.predict_proba(xtest)[:,1]
```

[41]:
```
from sklearn.metrics import roc_curve
fpr, tpr, _ = roc_curve(ytest,pred)
auc = roc_auc_score(ytest, pred)
plt.figure(figsize=(12,8))
plt.plot(fpr,tpr,label="Validation AUC-ROC="+str(auc))
x = np.linspace(0, 1, 1000)
plt.plot(x, x, linestyle='-')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```




[42]:
```
pred_val = model.predict(xtest)
```

[43]:
```
label_preds = pred_val

cm = confusion_matrix(ytest,label_preds)


def plot_confusion_matrix(cm, normalized=True, cmap='bone'):
    plt.figure(figsize=[7, 6])
    norm_cm = cm
    if normalized:
        norm_cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        sns.heatmap(norm_cm, annot=cm, fmt='g', xticklabels=['Predicted:␣
↪No','Predicted: Yes'], yticklabels=['Actual: No','Actual: Yes'], cmap=cmap)

plot_confusion_matrix(cm, ['No', 'Yes'])
```




[44]:
```
recall_score(ytest,pred_val)
```

[44]: 0.11580148317170565

[45]:
```
def cv_score(ml_model, rstate = 12, thres = 0.5, cols = df.columns):
    i = 1
    cv_scores = []
    df1 = df.copy()
    df1 = df[cols]

    # 5 Fold cross validation stratified on the basis of target
    kf = StratifiedKFold(n_splits=5,random_state=rstate,shuffle=True)
    for df_index,test_index in kf.split(df1,y_all):
        print('\n{} of kfold {}'.format(i,kf.n_splits))
        xtr,xvl = df1.loc[df_index],df1.loc[test_index]
        ytr,yvl = y_all.loc[df_index],y_all.loc[test_index]
```

```
        # Define model for fitting on the training set for each fold
        model = ml_model
        model.fit(xtr, ytr)
        pred_probs = model.predict_proba(xvl)
        pp = []

        # Use threshold to define the classes based on probability values
        for j in pred_probs[:,1]:
            if j>thres:
                pp.append(1)
            else:
                pp.append(0)

        # Calculate scores for each fold and print
        pred_val = pp
        roc_score = roc_auc_score(yvl,pred_probs[:,1])
        recall = recall_score(yvl,pred_val)
        precision = precision_score(yvl,pred_val)
        sufix = ""
        msg = ""
        msg += "ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.
↪4f} ".format(roc_score, recall,precision)
        print("{}".format(msg))

         # Save scores
        cv_scores.append(roc_score)
        i+=1
    return cv_scores
```

[46]:
```
baseline_scores = cv_score(LogisticRegression(), cols = baseline_cols)
```

```
1 of kfold 5
ROC AUC Score: 0.7644836090843695, Recall Score: 0.0751, Precision Score: 0.5766

2 of kfold 5
ROC AUC Score: 0.7805820573425136, Recall Score: 0.0722, Precision Score: 0.6552

3 of kfold 5
ROC AUC Score: 0.755325784138303, Recall Score: 0.1350, Precision Score: 0.6455

4 of kfold 5
ROC AUC Score: 0.7582064809820148, Recall Score: 0.1169, Precision Score: 0.6508

5 of kfold 5
ROC AUC Score: 0.7622577114607865, Recall Score: 0.1112, Precision Score: 0.5821
```

[47]:
```
all_feat_scores = cv_score(LogisticRegression())
```

```
1 of kfold 5
ROC AUC Score: 0.7445534888500668, Recall Score: 0.1274, Precision Score: 0.7016

2 of kfold 5
ROC AUC Score: 0.7296795807214058, Recall Score: 0.1188, Precision Score: 0.6281

3 of kfold 5
ROC AUC Score: 0.7184804935729603, Recall Score: 0.1131, Precision Score: 0.5667

4 of kfold 5
ROC AUC Score: 0.758577951701817, Recall Score: 0.2015, Precision Score: 0.6795

5 of kfold 5
ROC AUC Score: 0.7115271866407042, Recall Score: 0.0970, Precision Score: 0.4928
```

[48]:
```
from sklearn.feature_selection import RFE
import matplotlib.pyplot as plt

model = LogisticRegression()
rfe = RFE(estimator=model, n_features_to_select=1, step=1)
rfe.fit(df, y_all)
```

[48]:
```
RFE(estimator=LogisticRegression(), n_features_to_select=1)
```

[49]:
```
ranking_df = pd.DataFrame()
ranking_df['Feature_name'] = df.columns
ranking_df['Rank'] = rfe.ranking_
```

[50]:
```
ranked = ranking_df.sort_values(by=['Rank'])
```

[51]:
```
rfe_top_10_scores = cv_score(LogisticRegression(), cols =␣
↪ranked['Feature_name'][:10].values)
```

```
1 of kfold 5
ROC AUC Score: 0.7999459459459459, Recall Score: 0.2310, Precision Score: 0.7254

2 of kfold 5
ROC AUC Score: 0.8076882129277567, Recall Score: 0.2224, Precision Score: 0.7548

3 of kfold 5
ROC AUC Score: 0.7994672776849501, Recall Score: 0.2253, Precision Score: 0.7096

4 of kfold 5
ROC AUC Score: 0.7944628044127514, Recall Score: 0.2063, Precision Score: 0.7282
```

```
5 of kfold 5
ROC AUC Score: 0.7950266916205085, Recall Score: 0.1920, Precision Score: 0.6733
```

[52]:
```
cv_score(LogisticRegression(), cols = ranked['Feature_name'][:10].values,␣
↪thres=0.14)
```

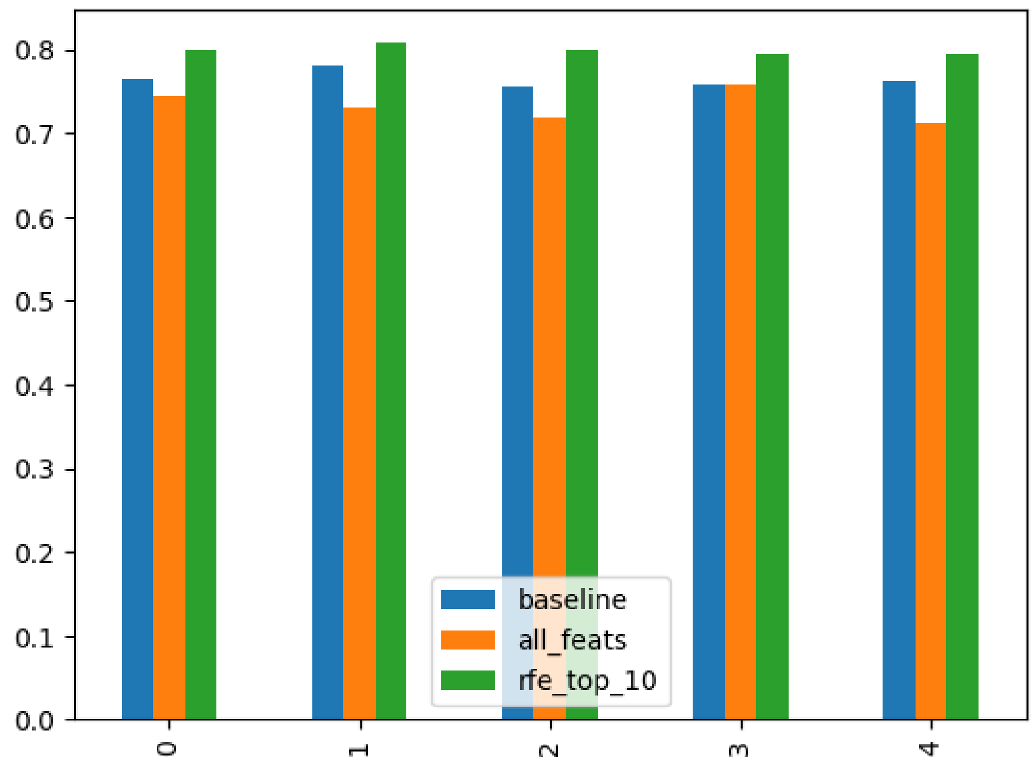
```
1 of kfold 5
ROC AUC Score: 0.7999459459459459, Recall Score: 0.8327, Precision Score: 0.2888

2 of kfold 5
ROC AUC Score: 0.8076882129277567, Recall Score: 0.8365, Precision Score: 0.2952

3 of kfold 5
ROC AUC Score: 0.7994672776849501, Recall Score: 0.8241, Precision Score: 0.2951

4 of kfold 5
ROC AUC Score: 0.7944628044127514, Recall Score: 0.8194, Precision Score: 0.2882

5 of kfold 5
ROC AUC Score: 0.7950266916205085, Recall Score: 0.8099, Precision Score: 0.2993
```

[52]:
```
[0.7999459459459459,
 0.8076882129277567,
 0.7994672776849501,
 0.7944628044127514,
 0.7950266916205085]
```

[53]:
```
results_df = pd.DataFrame({'baseline':baseline_scores, 'all_feats':␣
↪all_feat_scores, 'rfe_top_10': rfe_top_10_scores})
```

[54]:
```
results_df.plot(y=["baseline", "all_feats", "rfe_top_10"], kind="bar")
```

[54]:
```
<Axes: >
```




### 0.0.2 Here, we can see that the model based on RFE is giving the best result for each fold.

```
[ ]:
```