In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df_1 = pd.read_csv(r"C:\Users\vishnu reddy\OneDrive\Desktop\Micro-credit-Data-file.
df_2= pd.read_excel(r"C:\Users\vishnu reddy\OneDrive\Desktop\Micro-credit-card-Data
```

In [3]:
```
df_1.shape
df_1.head()
```

Out[3]:

| | Unnamed: 0 | label | msisdn | aon | daily_decr30 | daily_decr90 | rental30 | rental90 | last_rech_ |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 21408I70789 | 272.0 | 3055.050000 | 3065.150000 | 220.13 | 260.13 | |
| **1** | 2 | 1 | 76462I70374 | 712.0 | 12122.000000 | 12124.750000 | 3691.26 | 3691.26 | |
| **2** | 3 | 1 | 17943I70372 | 535.0 | 1398.000000 | 1398.000000 | 900.13 | 900.13 | |
| **3** | 4 | 1 | 55773I70781 | 241.0 | 21.228000 | 21.228000 | 159.42 | 159.42 | |
| **4** | 5 | 1 | 03813I82730 | 947.0 | 150.619333 | 150.619333 | 1098.90 | 1098.90 | |

5 rows × 37 columns

In [4]:
```
df=pd.concat([df_1,df_2])
print("shape of df is ",df.shape)
```

```
shape of df is  (209599, 37)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'label', 'msisdn', 'aon', 'daily_decr30', 'daily_decr90',
       'rental30', 'rental90', 'last_rech_date_ma', 'last_rech_date_da',
       'last_rech_amt_ma', 'cnt_ma_rech30', 'fr_ma_rech30',
       'sumamnt_ma_rech30', 'medianamnt_ma_rech30', 'medianmarechprebal30',
       'cnt_ma_rech90', 'fr_ma_rech90', 'sumamnt_ma_rech90',
       'medianamnt_ma_rech90', 'medianmarechprebal90', 'cnt_da_rech30',
       'fr_da_rech30', 'cnt_da_rech90', 'fr_da_rech90', 'cnt_loans30',
       'amnt_loans30', 'maxamnt_loans30', 'medianamnt_loans30', 'cnt_loans90',
       'amnt_loans90', 'maxamnt_loans90', 'medianamnt_loans90', 'payback30',
       'payback90', 'pcircle', 'pdate'],
      dtype='object')
```

In [6]:
```
#Since from the data I have seen that their is no use of column unnamed so I am dro
df.drop(['Unnamed: 0'], axis=1,inplace=True)
```

In [7]:
```
df.shape
```

Out[7]:
```
(209599, 36)
```

In [8]:
```
df.dtypes
```

Out[8]:
```
label                     int64
msisdn                   object
aon                     float64
daily_decr30            float64
daily_decr90            float64
rental30                float64
rental90                float64
last_rech_date_ma       float64
last_rech_date_da       float64
last_rech_amt_ma          int64
cnt_ma_rech30             int64
fr_ma_rech30            float64
sumamnt_ma_rech30       float64
medianamnt_ma_rech30    float64
medianmarechprebal30    float64
cnt_ma_rech90             int64
fr_ma_rech90              int64
sumamnt_ma_rech90         int64
medianamnt_ma_rech90    float64
medianmarechprebal90    float64
cnt_da_rech30           float64
fr_da_rech30            float64
cnt_da_rech90             int64
fr_da_rech90            float64
cnt_loans30               int64
amnt_loans30              int64
maxamnt_loans30         float64
medianamnt_loans30      float64
cnt_loans90             float64
amnt_loans90              int64
maxamnt_loans90           int64
medianamnt_loans90      float64
payback30               float64
payback90               float64
pcircle                  object
pdate                    object
dtype: object
```

In [9]:
```
#frequency of object features
for col in df.columns:
    if df[col].dtype=="object":
        print(df[col].value_counts())
        print()
```

```
msisdn
04581I85330    7
47819I90840    7
22038I88658    6
43096I88688    6
43430I70786    6
              ..
59686I90584    1
00504I91190    1
40868I82734    1
50882I95204    1
6128973512     1
Name: count, Length: 186249, dtype: int64

pcircle
UPW    209599
Name: count, dtype: int64

pdate
2016-07-04    3150
2016-07-05    3127
2016-07-07    3116
2016-06-20    3099
2016-06-17    3082
              ...
2016-06-04    1559
2016-08-18    1407
2016-08-19    1132
2016-08-20     788
2016-08-21     324
Name: count, Length: 82, dtype: int64
```

In [10]:
```
#I have change the date columns into the interger
df['pdate'].str.replace("-","").astype(int)
```

Out[10]:
```
0     20160720
1     20160810
2     20160819
3     20160606
4     20160622
        ...
1     20160724
2     20160713
3     20160730
4     20160706
5     20160814
Name: pdate, Length: 209599, dtype: int32
```

In [11]:
```
from sklearn.preprocessing import LabelEncoder

# Assuming 'msisdn' is the column you're trying to encode
# Convert numerical column to string type
df['msisdn'] = df['msisdn'].astype(str)

# Initialize LabelEncoder
le = LabelEncoder()

# Encode the column
df['msisdn'] = le.fit_transform(df['msisdn'])
```

In [12]:
```
from sklearn.preprocessing import LabelEncoder

# Assuming 'msisdn' is the column you're trying to encode
```

```
# Convert numerical column to string type
df['pcircle'] = df['pcircle'].astype(str)

# Initialize LabelEncoder
le = LabelEncoder()

# Encode the column
df['pcircle'] = le.fit_transform(df['pcircle'])
```

In [13]:
```
df['pdate'] = df['pdate'].astype(str)

# Initialize LabelEncoder
le = LabelEncoder()

# Encode the column
df['pdate'] = le.fit_transform(df['pdate'])
```

In [14]:
```
df.dtypes
```

Out[14]:
```
label                     int64
msisdn                    int32
aon                     float64
daily_decr30            float64
daily_decr90            float64
rental30                float64
rental90                float64
last_rech_date_ma       float64
last_rech_date_da       float64
last_rech_amt_ma          int64
cnt_ma_rech30             int64
fr_ma_rech30            float64
sumamnt_ma_rech30       float64
medianamnt_ma_rech30    float64
medianmarechprebal30    float64
cnt_ma_rech90             int64
fr_ma_rech90              int64
sumamnt_ma_rech90         int64
medianamnt_ma_rech90    float64
medianmarechprebal90    float64
cnt_da_rech30           float64
fr_da_rech30            float64
cnt_da_rech90             int64
fr_da_rech90            float64
cnt_loans30               int64
amnt_loans30              int64
maxamnt_loans30         float64
medianamnt_loans30      float64
cnt_loans90             float64
amnt_loans90              int64
maxamnt_loans90           int64
medianamnt_loans90      float64
payback30               float64
payback90               float64
pcircle                   int32
pdate                     int32
dtype: object
```

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
label                   0
msisdn                  0
aon                     0
daily_decr30            0
daily_decr90            0
rental30                0
rental90                0
last_rech_date_ma       0
last_rech_date_da       6
last_rech_amt_ma        0
cnt_ma_rech30           0
fr_ma_rech30            2
sumamnt_ma_rech30       0
medianamnt_ma_rech30    1
medianmarechprebal30    1
cnt_ma_rech90           0
fr_ma_rech90            0
sumamnt_ma_rech90       0
medianamnt_ma_rech90    0
medianmarechprebal90    0
cnt_da_rech30           0
fr_da_rech30            6
cnt_da_rech90           0
fr_da_rech90            6
cnt_loans30             0
amnt_loans30            0
maxamnt_loans30         0
medianamnt_loans30      0
cnt_loans90             0
amnt_loans90            0
maxamnt_loans90         0
medianamnt_loans90      0
payback30               3
payback90               3
pcircle                 0
pdate                   0
dtype: int64
```

In [16]:
```
df.dropna(inplace=True)
```

In [17]:
```
print(df.isnull().sum().sum())
```

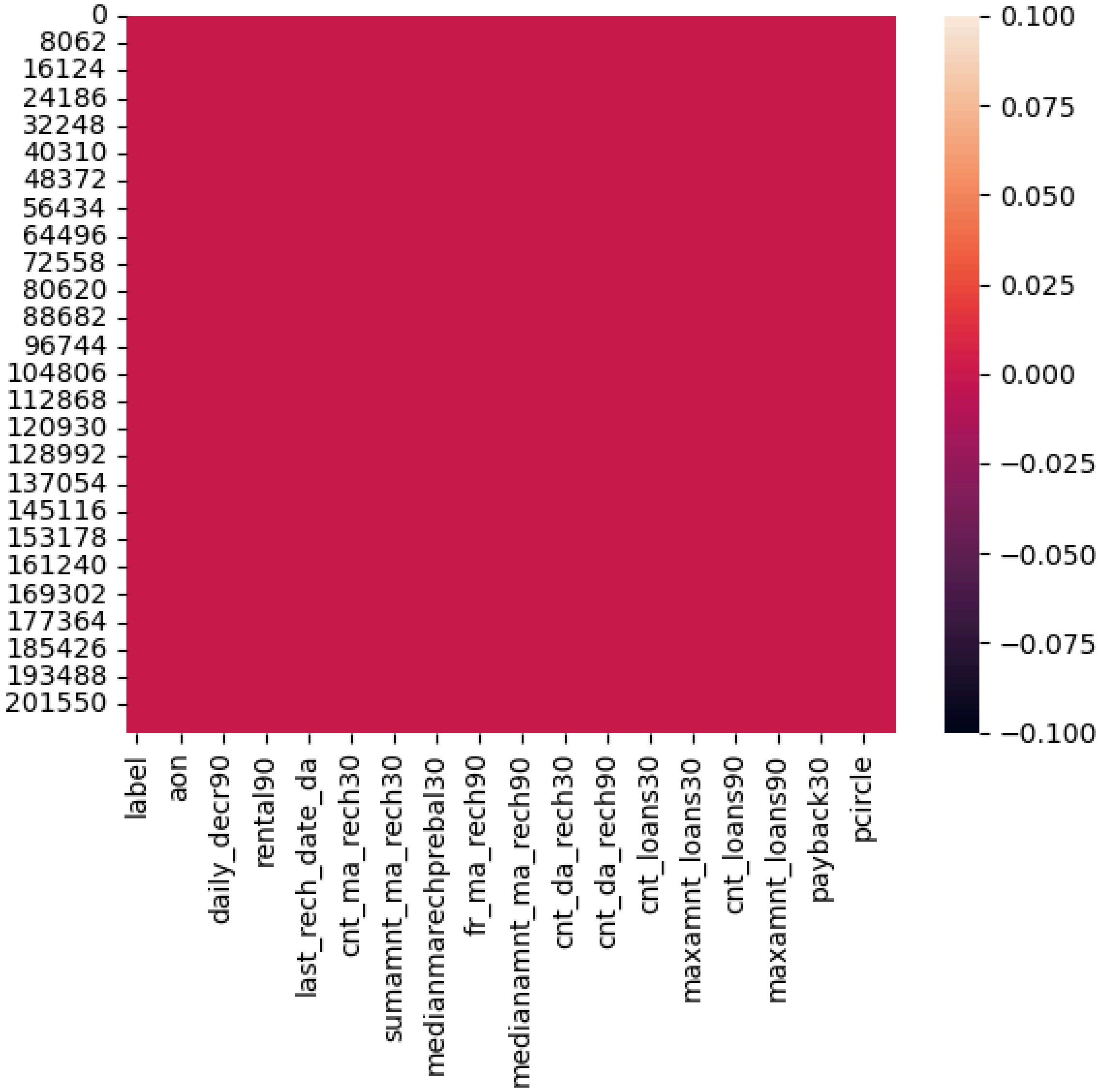
```
0
```

In [18]:
```
df.info()
```

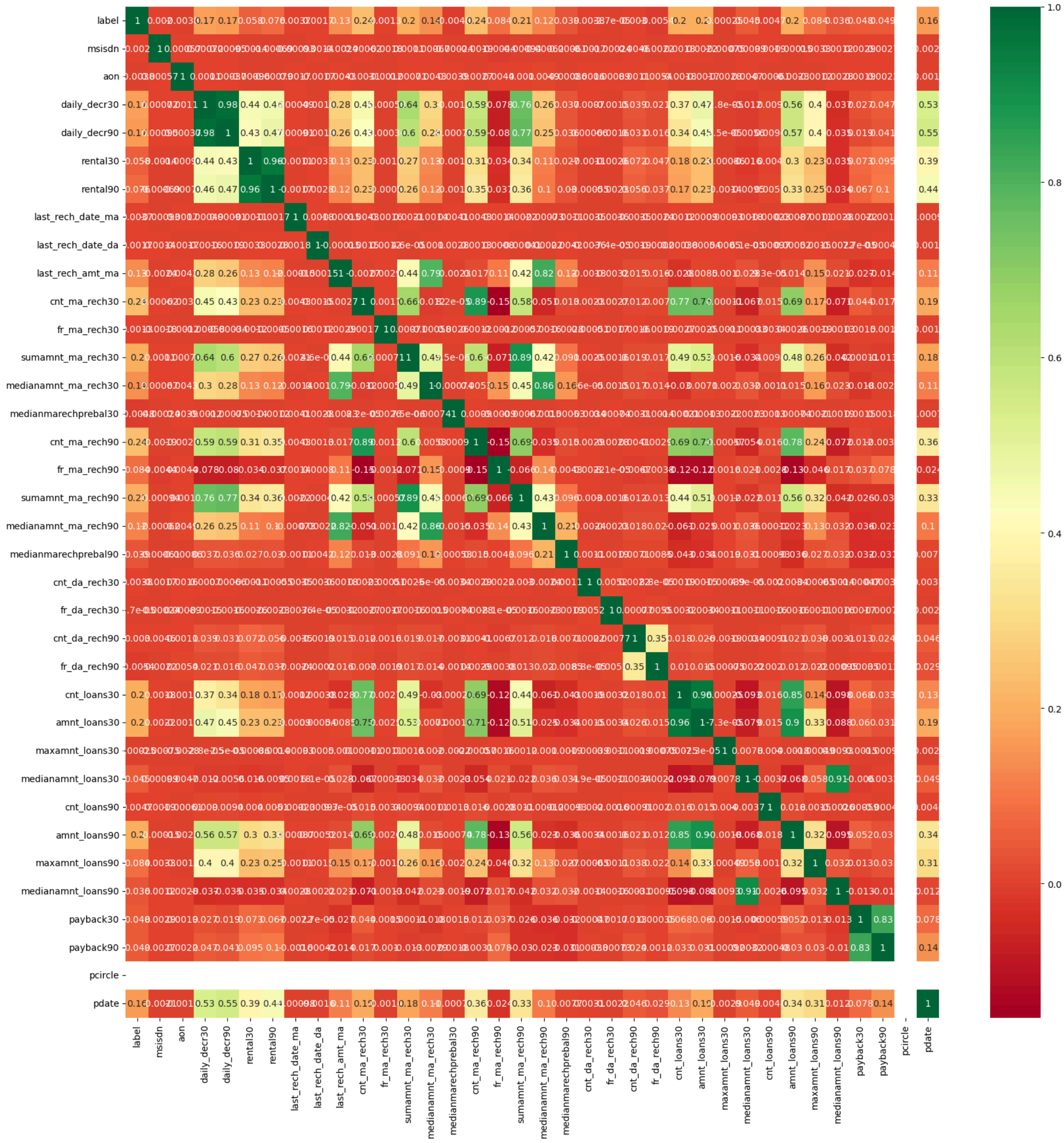
```
<class 'pandas.core.frame.DataFrame'>
Index: 209593 entries, 0 to 209592
Data columns (total 36 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   label                 209593 non-null  int64  
 1   msisdn                209593 non-null  int32  
 2   aon                   209593 non-null  float64
 3   daily_decr30          209593 non-null  float64
 4   daily_decr90          209593 non-null  float64
 5   rental30              209593 non-null  float64
 6   rental90              209593 non-null  float64
 7   last_rech_date_ma     209593 non-null  float64
 8   last_rech_date_da     209593 non-null  float64
 9   last_rech_amt_ma      209593 non-null  int64  
 10  cnt_ma_rech30         209593 non-null  int64  
 11  fr_ma_rech30          209593 non-null  float64
 12  sumamnt_ma_rech30     209593 non-null  float64
 13  medianamnt_ma_rech30  209593 non-null  float64
 14  medianmarechprebal30  209593 non-null  float64
 15  cnt_ma_rech90         209593 non-null  int64  
 16  fr_ma_rech90          209593 non-null  int64  
 17  sumamnt_ma_rech90     209593 non-null  int64  
 18  medianamnt_ma_rech90  209593 non-null  float64
 19  medianmarechprebal90  209593 non-null  float64
 20  cnt_da_rech30         209593 non-null  float64
 21  fr_da_rech30          209593 non-null  float64
 22  cnt_da_rech90         209593 non-null  int64  
 23  fr_da_rech90          209593 non-null  float64
 24  cnt_loans30           209593 non-null  int64  
 25  amnt_loans30          209593 non-null  int64  
 26  maxamnt_loans30       209593 non-null  float64
 27  medianamnt_loans30    209593 non-null  float64
 28  cnt_loans90           209593 non-null  float64
 29  amnt_loans90          209593 non-null  int64  
 30  maxamnt_loans90       209593 non-null  int64  
 31  medianamnt_loans90    209593 non-null  float64
 32  payback30             209593 non-null  float64
 33  payback90             209593 non-null  float64
 34  pcircle               209593 non-null  int32  
 35  pdate                 209593 non-null  int32  
dtypes: float64(22), int32(3), int64(11)
memory usage: 56.8 MB
```

In [19]:
```
sns.heatmap(df.isnull())
```

Out[19]:
```
<Axes: >
```



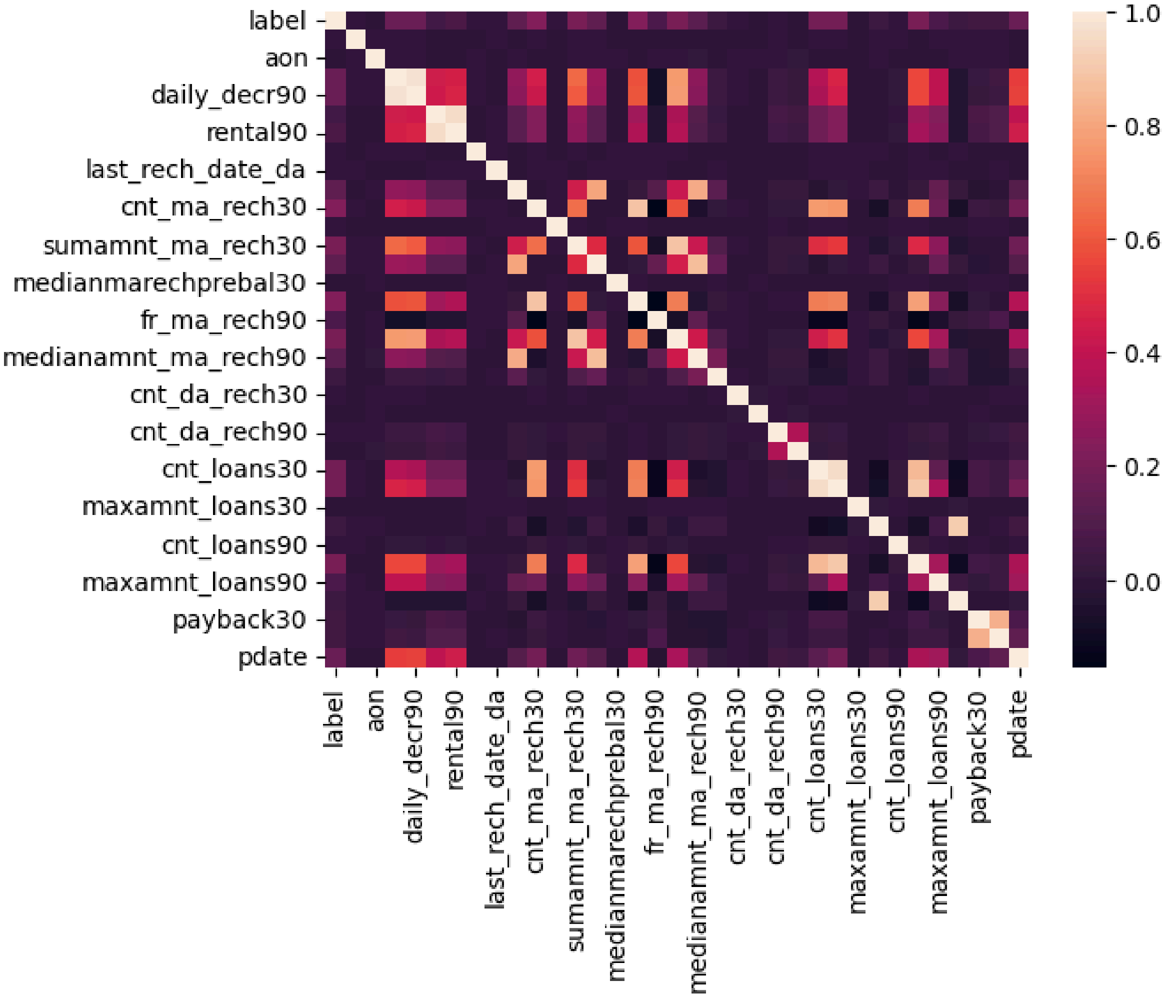

In [20]:
```
#get correlations of each feature in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



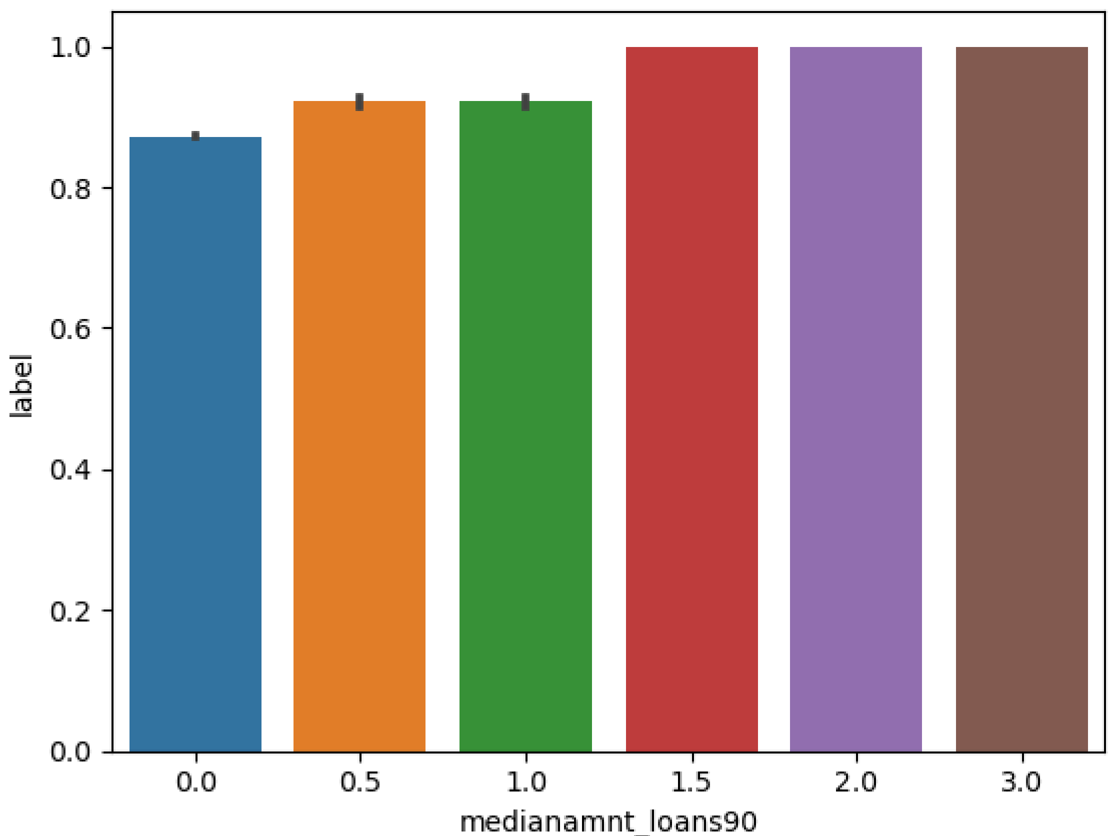
In [21]:
```
df.drop(['pcircle'], axis = 1, inplace = True)
```

In [22]:
```
dfcorr=df.corr()
sns.heatmap(dfcorr)
```

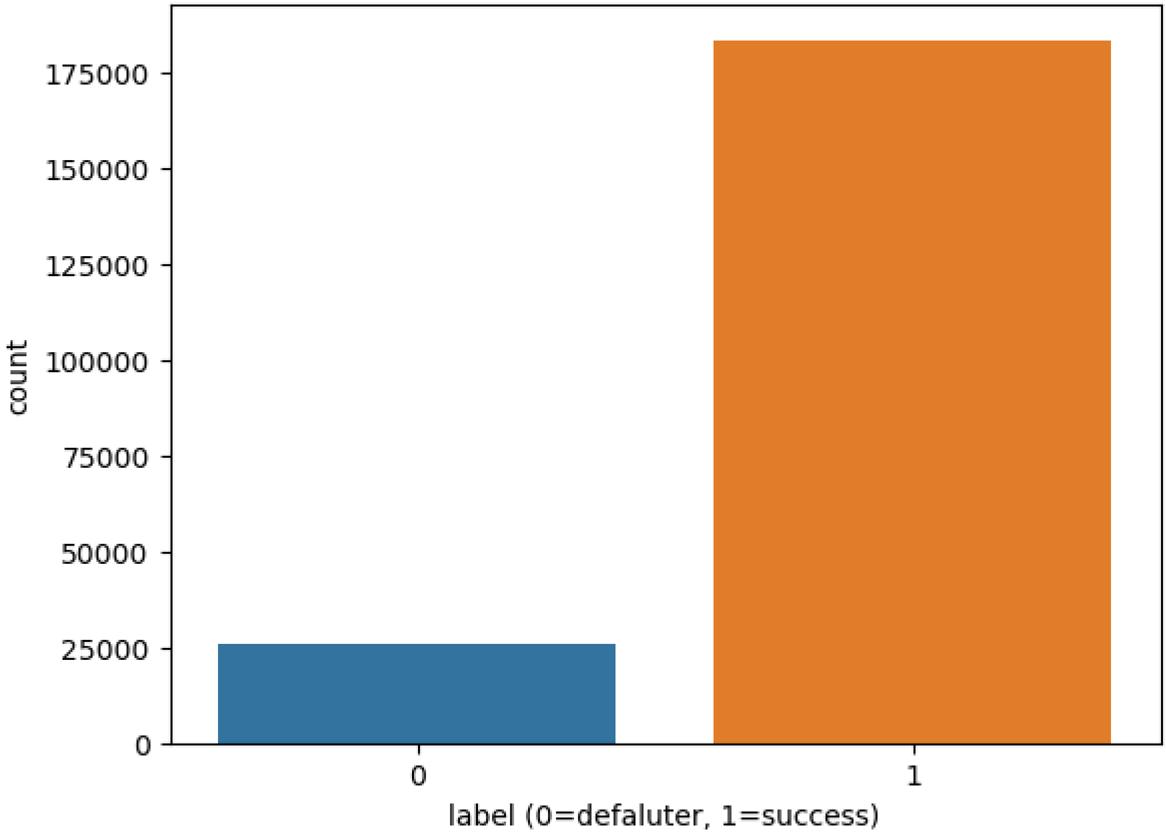
Out[22]: <Axes: >




In [23]:
```
sns.barplot(x='medianamnt_loans90',y='label',data=df)
plt.show()
```

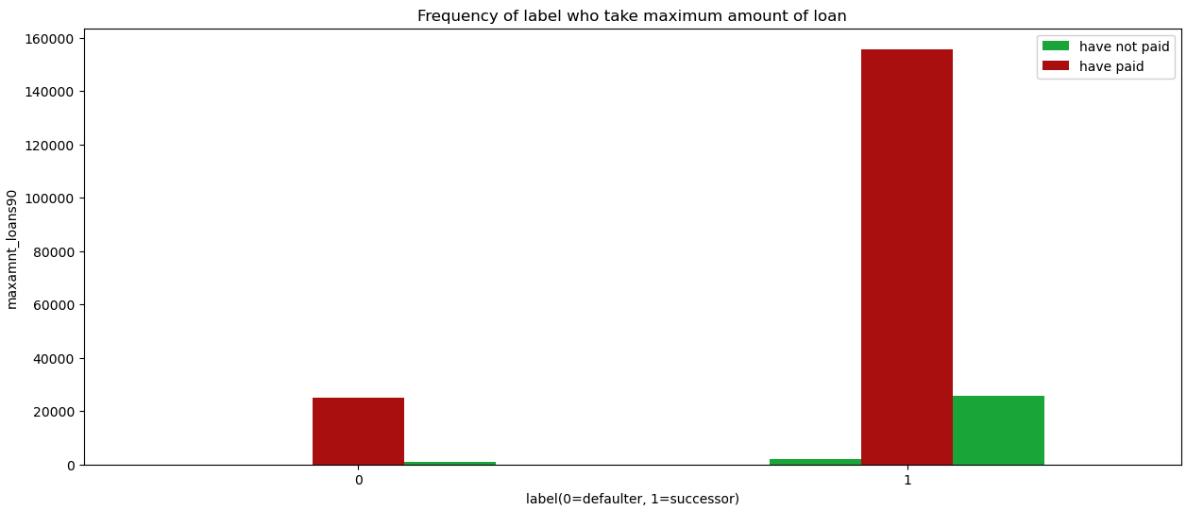


In [24]:
```
sns.countplot(x='label',data=df)
plt.xlabel('label (0=defaluter, 1=success)')
plt.show()
```



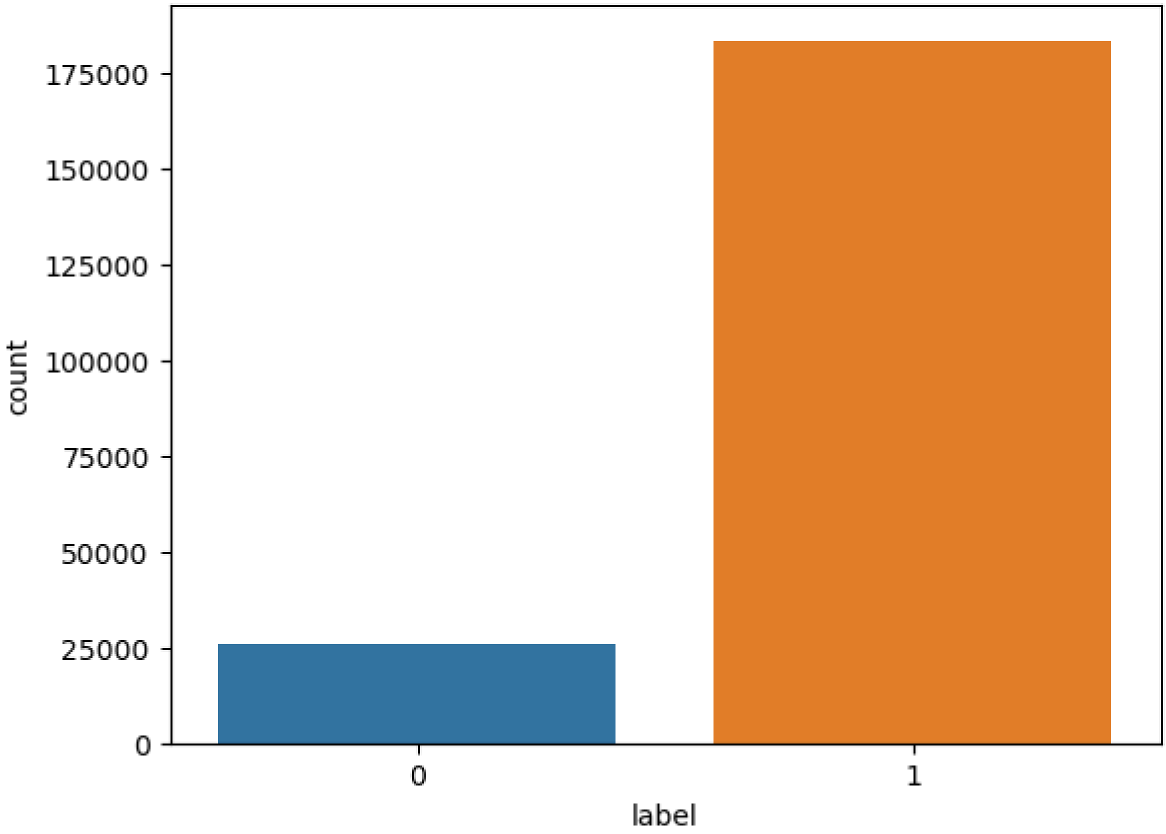

In [25]:
```
# pay back credit amount of successor are 175000 and failure to payback credit amou
```

In [26]:
```
pd.crosstab(df.label,df.maxamnt_loans90).plot(kind='bar',figsize=(15,6),color=['#1(
plt.title('Frequency of label who take maximum amount of loan')
plt.xlabel('label(0=defaulter, 1=successor)')
plt.xticks(rotation=0)
plt.legend(['have not paid', 'have paid'])
plt.ylabel('maxamnt_loans90')
plt.show()
#maxamnt_loans90
```

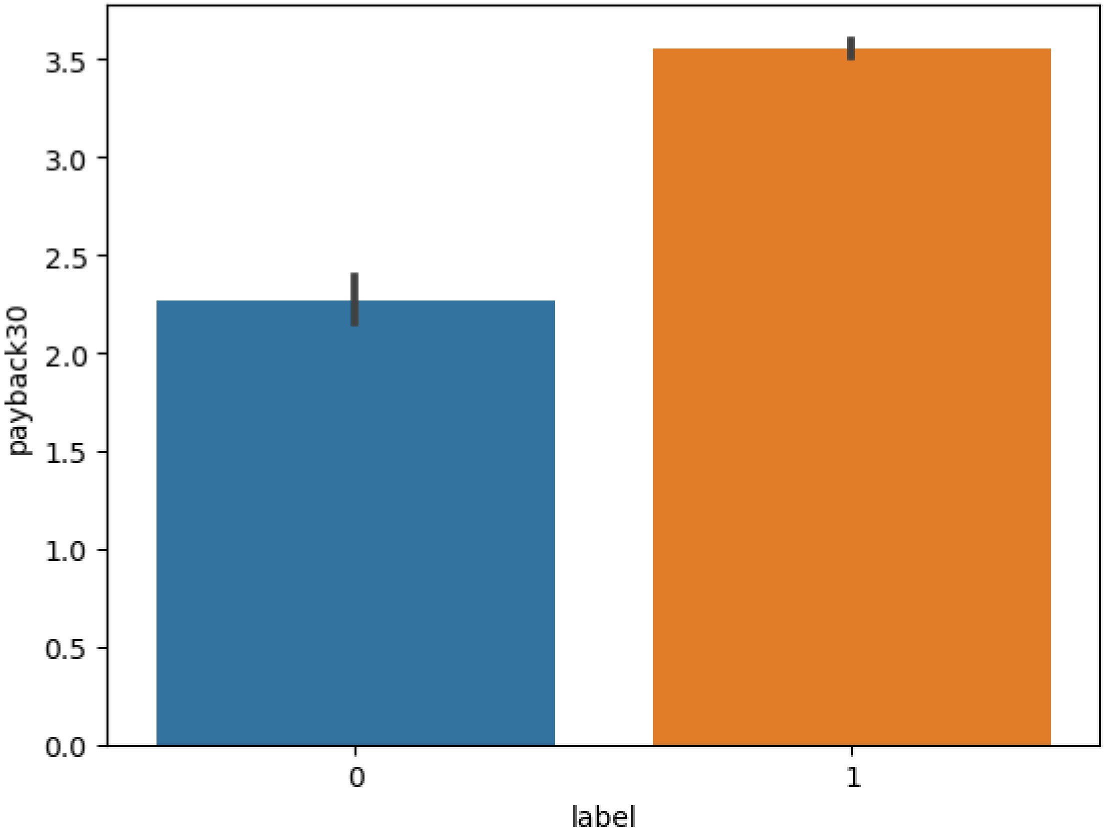


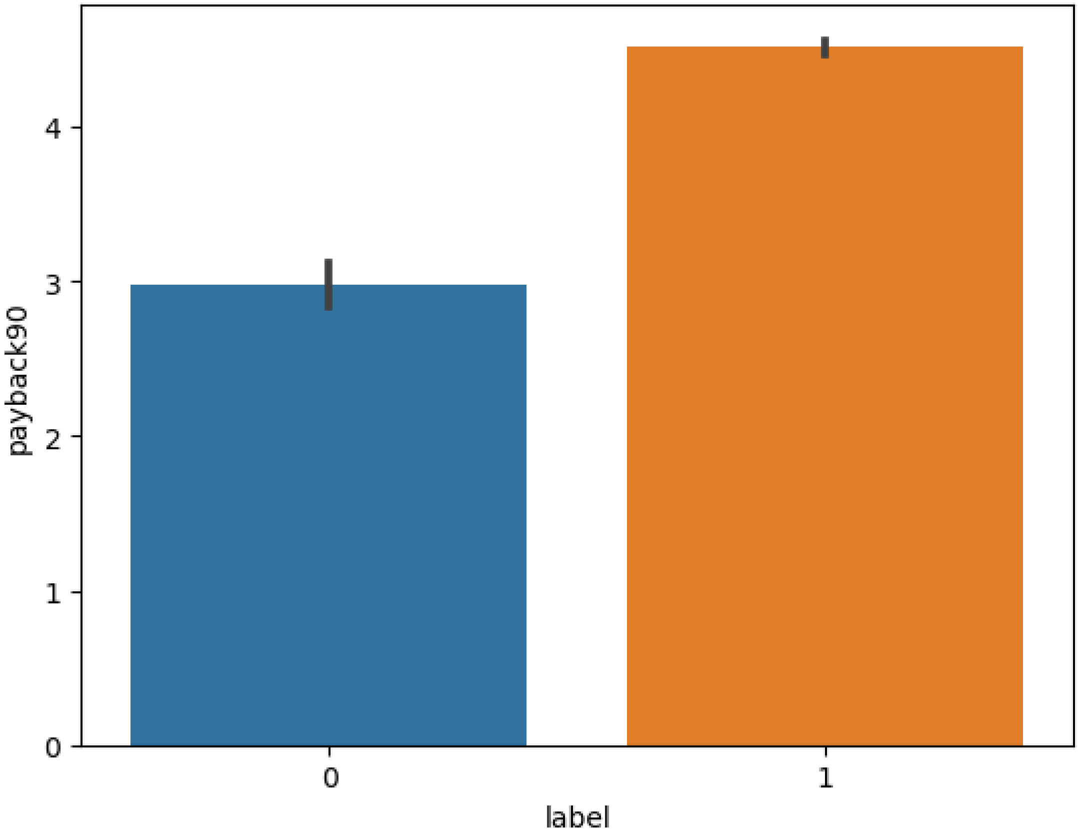

In [27]:
```
#Maximum amount of loan taken by the user in last 90 days and who have paid is the
```

In [28]:
```
sns.countplot(x="label",data=df)
plt.show()
#the users that didn't paid back the credit amount within 5 days is around 1/8 th c
```

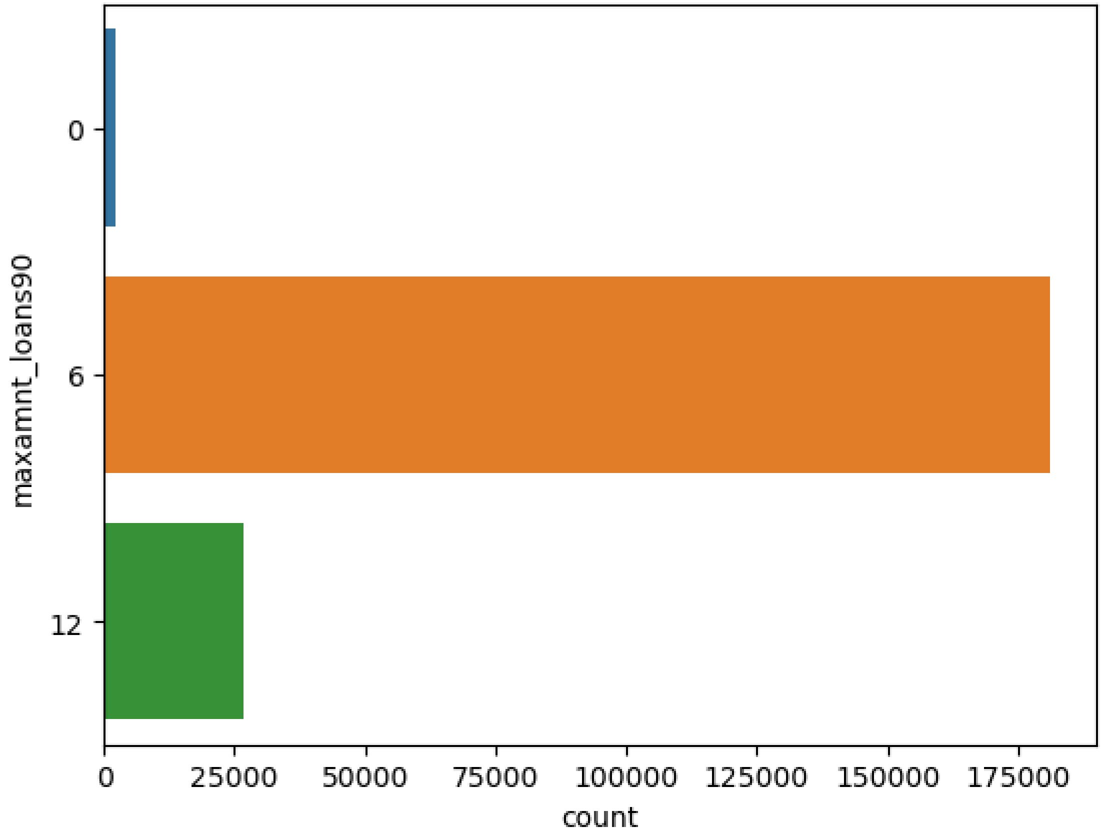


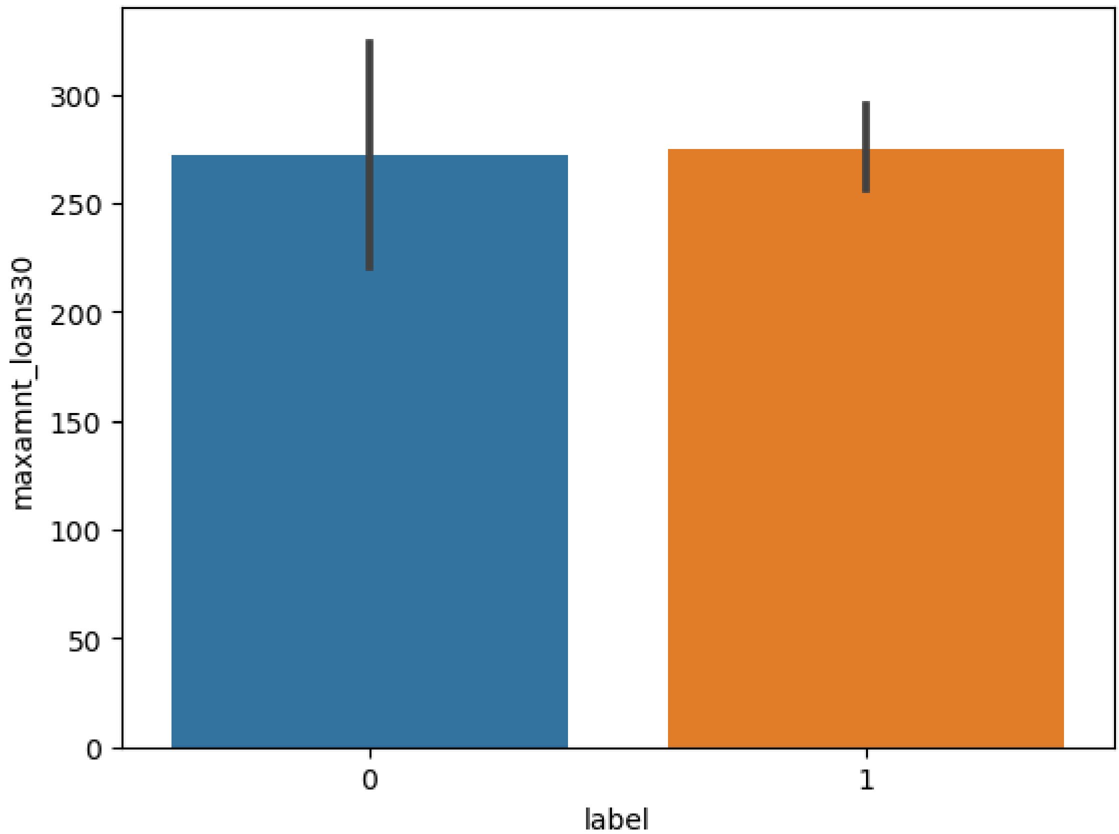

In [29]:
```
sns.barplot(y="payback30",x="label",data=df)
plt.show()
sns.barplot(y="payback90",x="label",data=df)
plt.show()
# average loan payback time is 3-4 days.
```

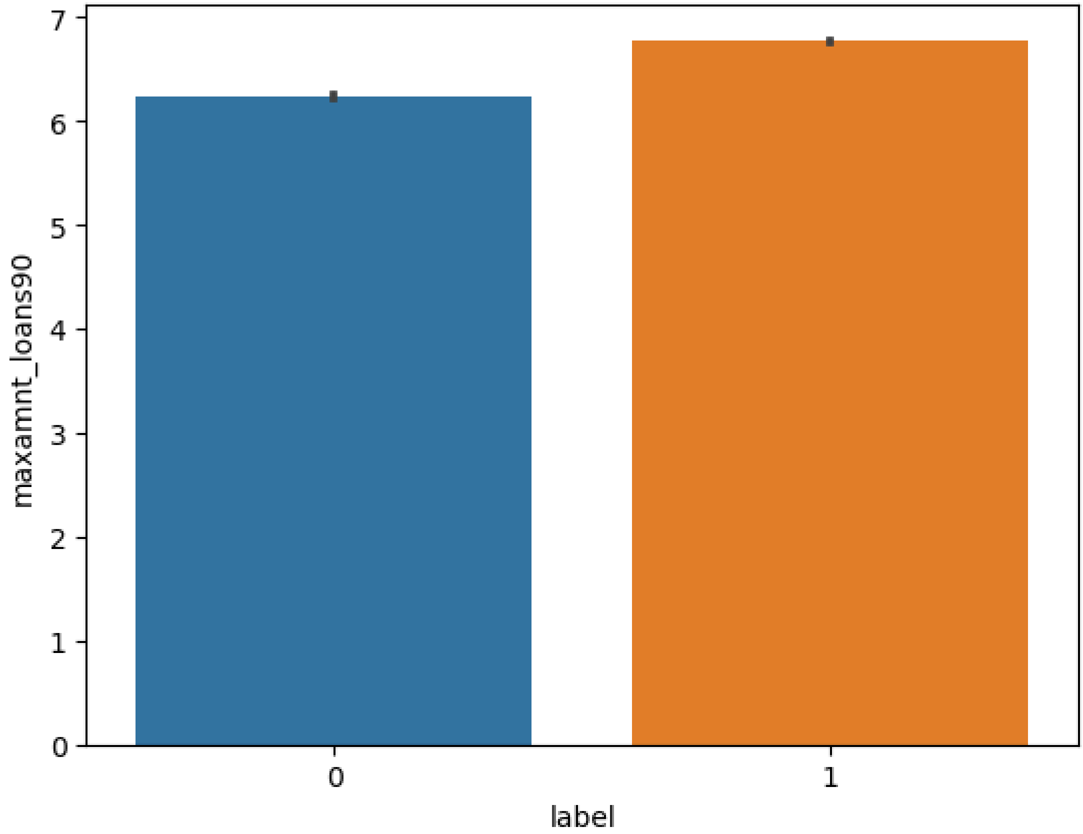


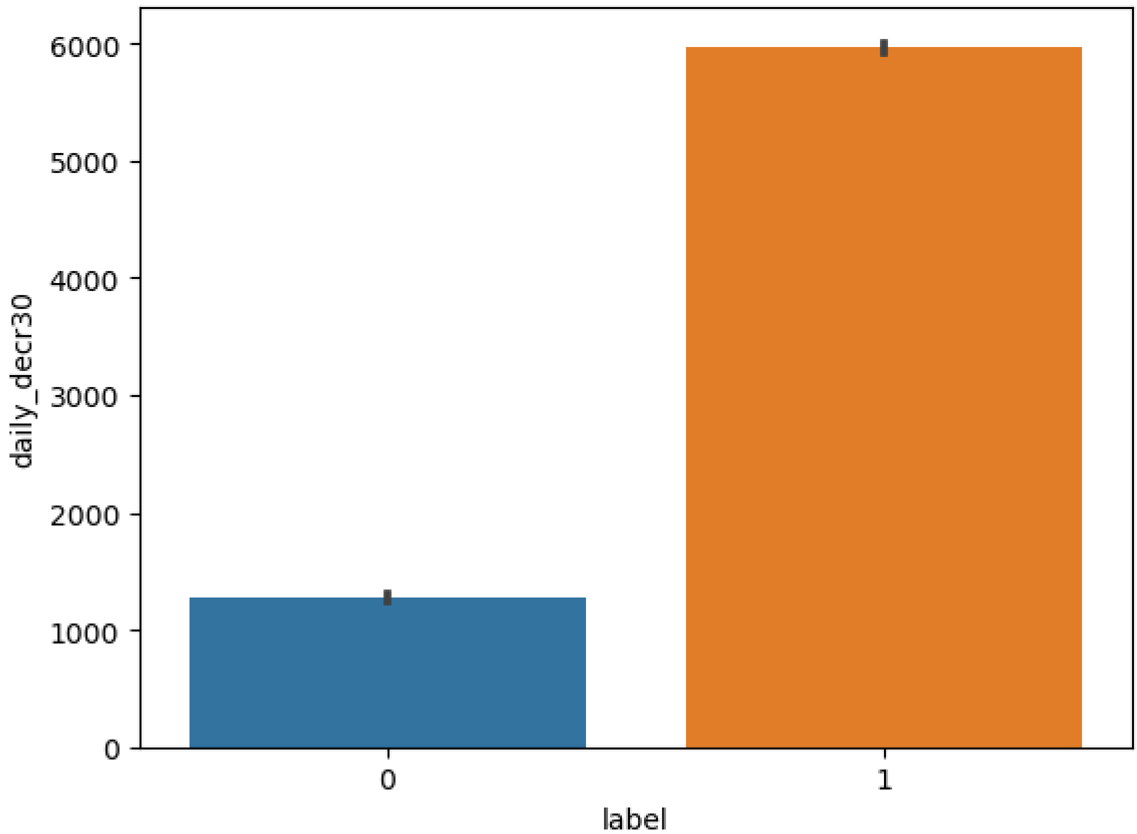

In [30]:
```
sns.countplot(y="maxamnt_loans90",data=df)
plt.show()
sns.barplot(y="maxamnt_loans30",x="label",data=df)
plt.show()
sns.barplot(y="maxamnt_loans90",x="label",data=df)
plt.show()
#maximum amount of loan taken by each user in 90 days is 5 Rs for which they had to
#we also see outliers present in maximum amount loan taken in 30 days. And 50% user
```

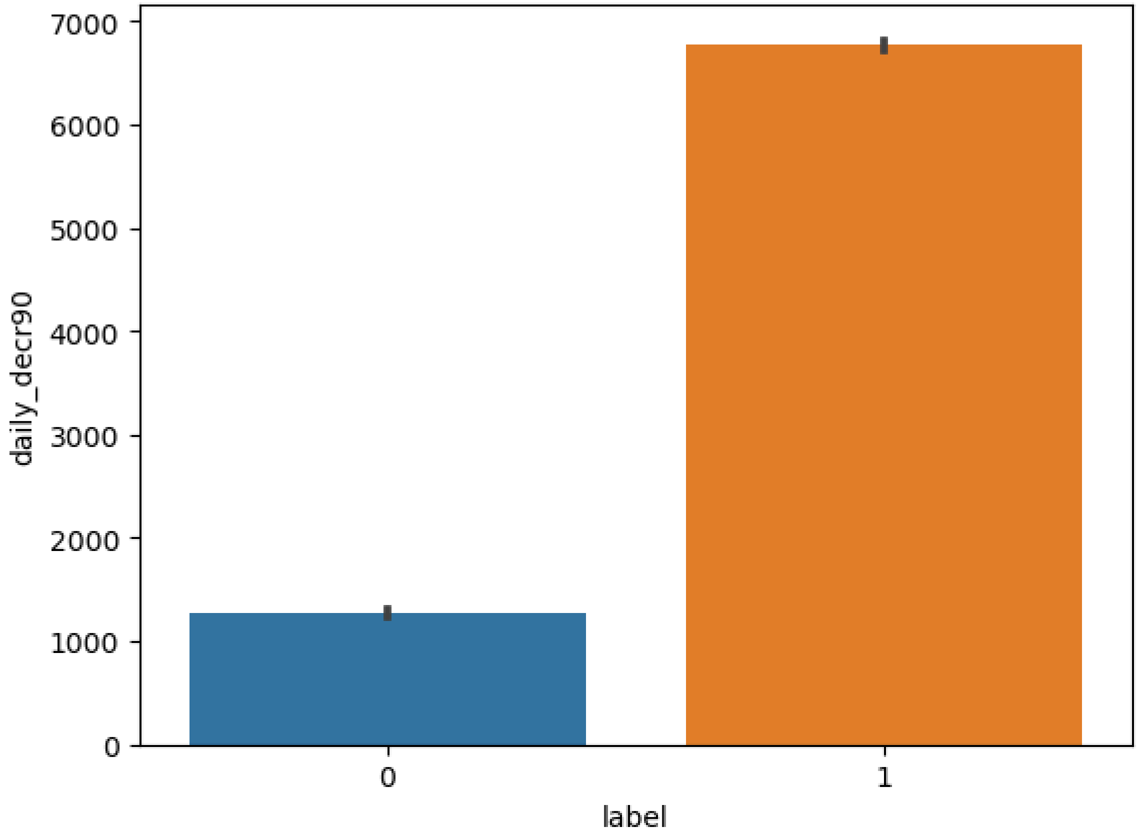

0
6
12
maxamnt_loans90
0
25000
50000
75000
100000
125000
150000
175000
count


300
250
200
150
100
50
0
maxamnt_loans30
0
1
label

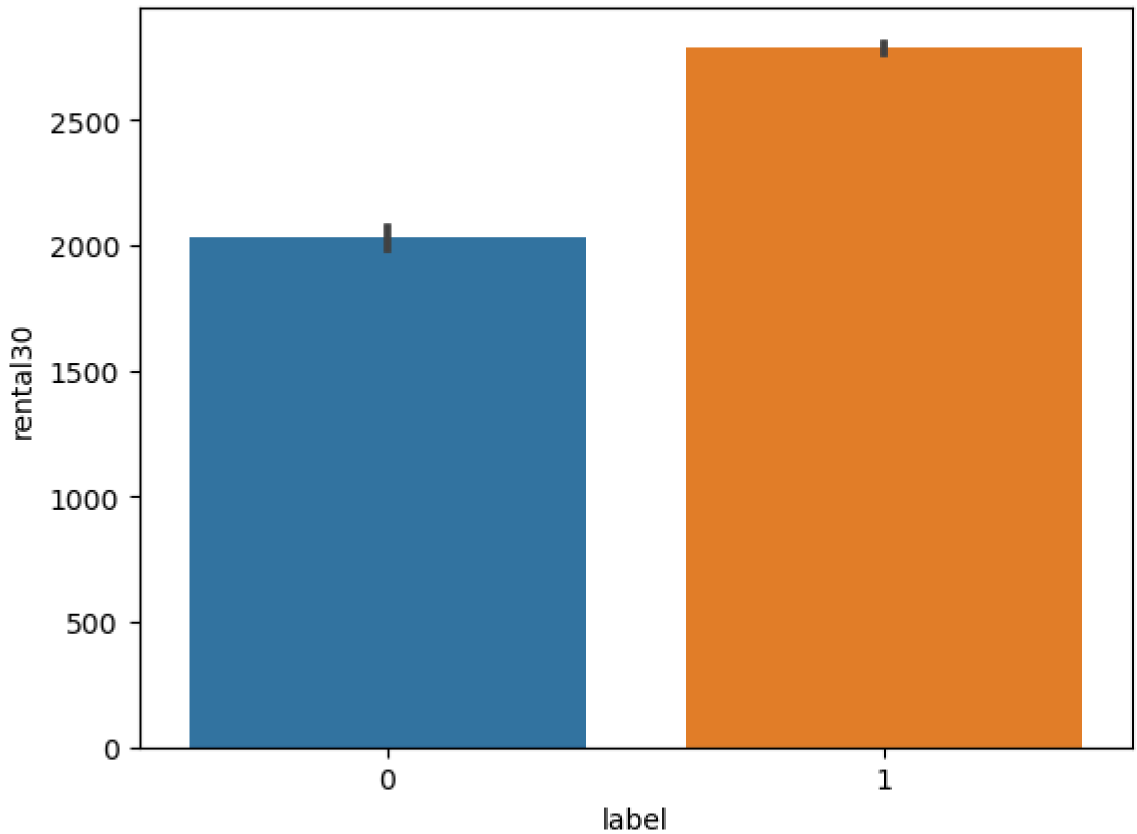


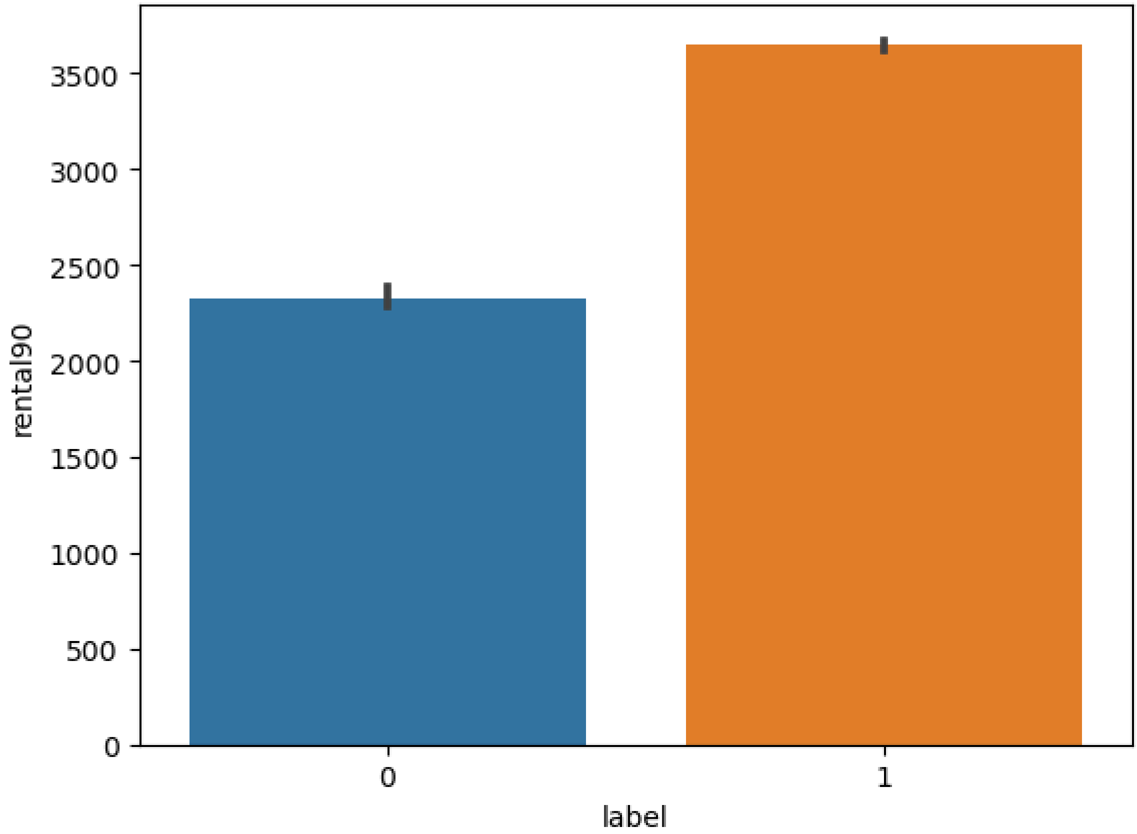

In [31]:
```
sns.barplot(y="daily_decr30",x="label",data=df)
plt.show()
sns.barplot(y="daily_decr90",x="label",data=df)
plt.show()
#non defaulters spent 6 times higher daily amount from main account within 30 days
#non defaulters spent 7 times higher daily amount from main account within 90 days
```



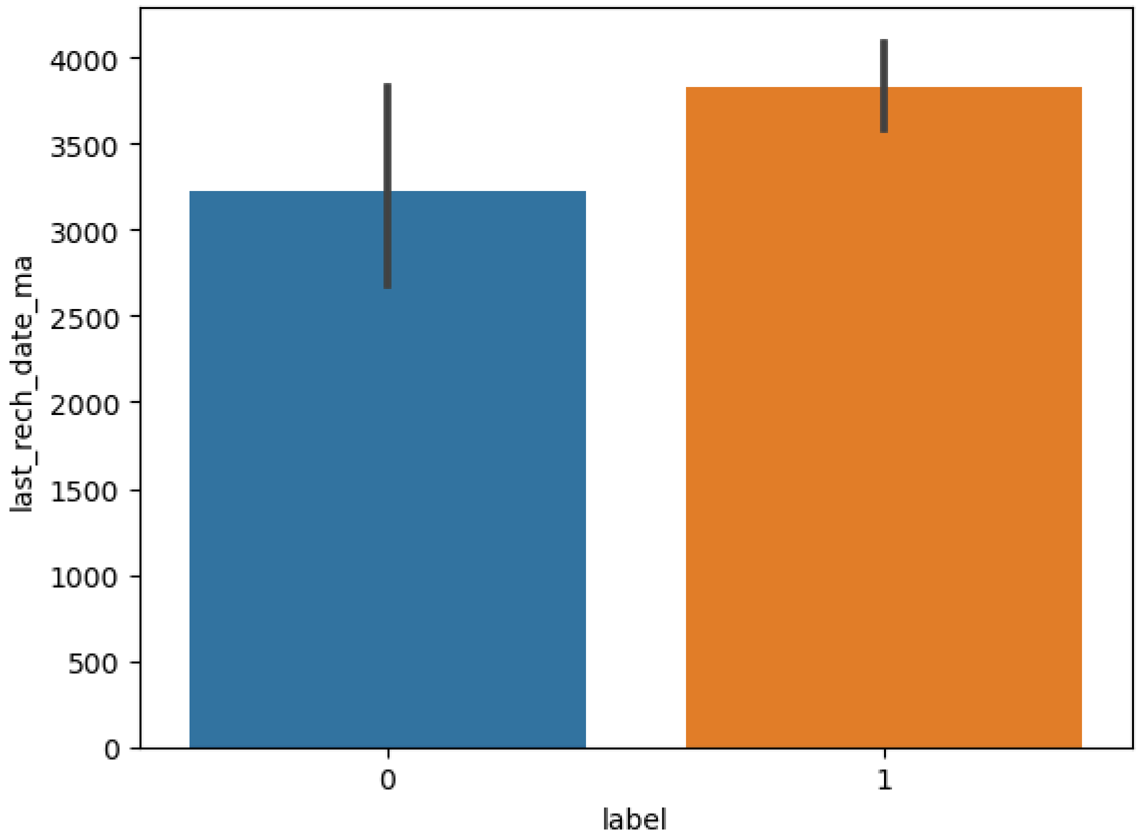


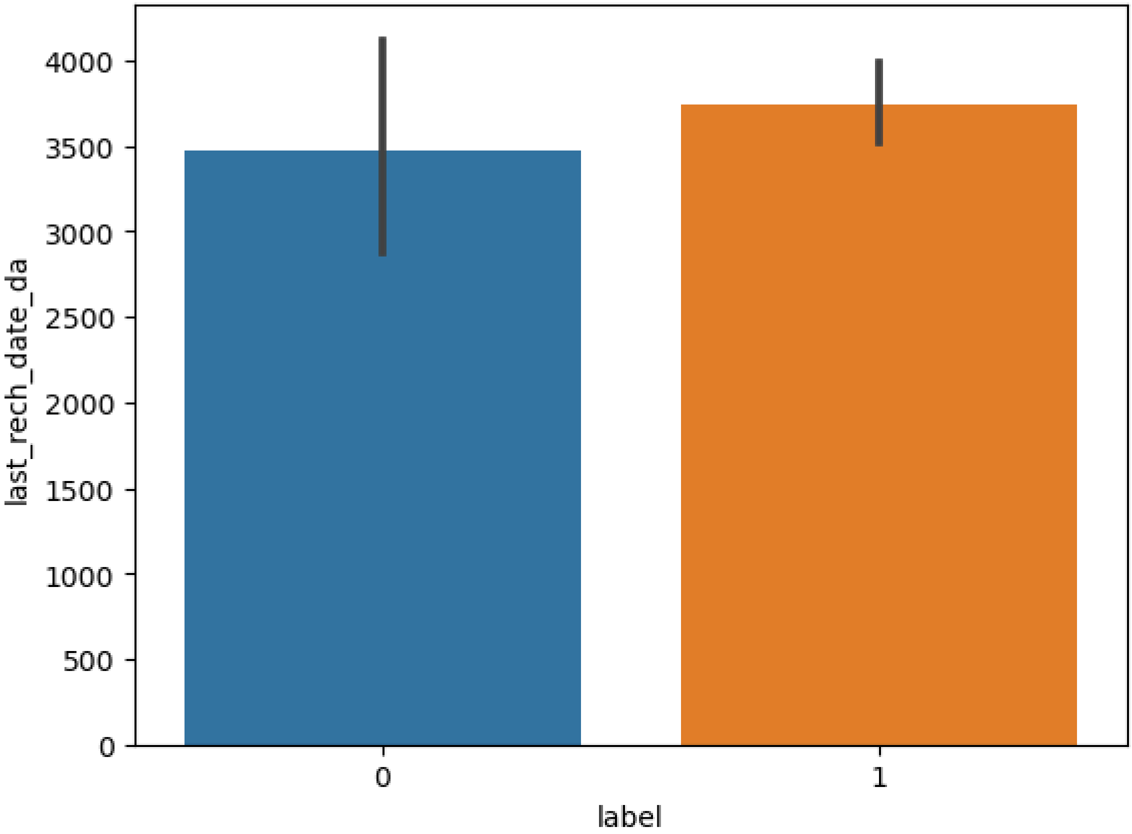

In [32]:
```
sns.barplot(y="rental30",x="label",data=df)
plt.show()
sns.barplot(y="rental90",x="label",data=df)
plt.show()
#Average main account balance is high for non defaulters
```



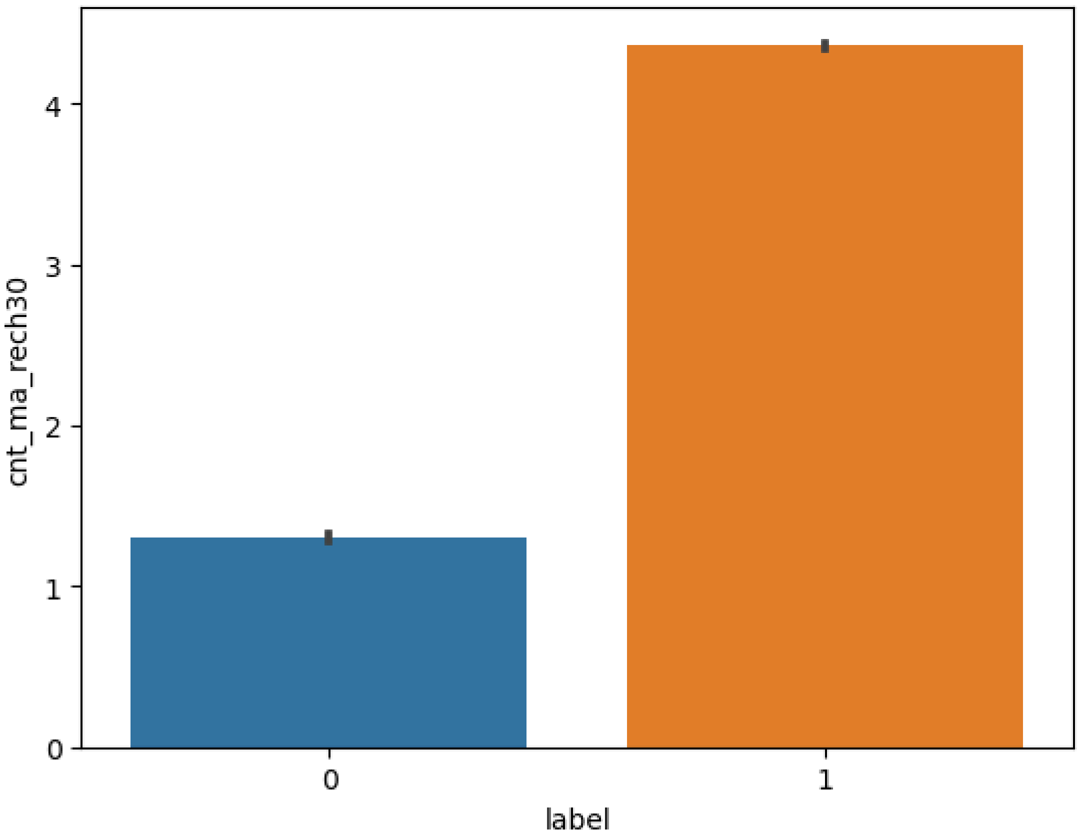


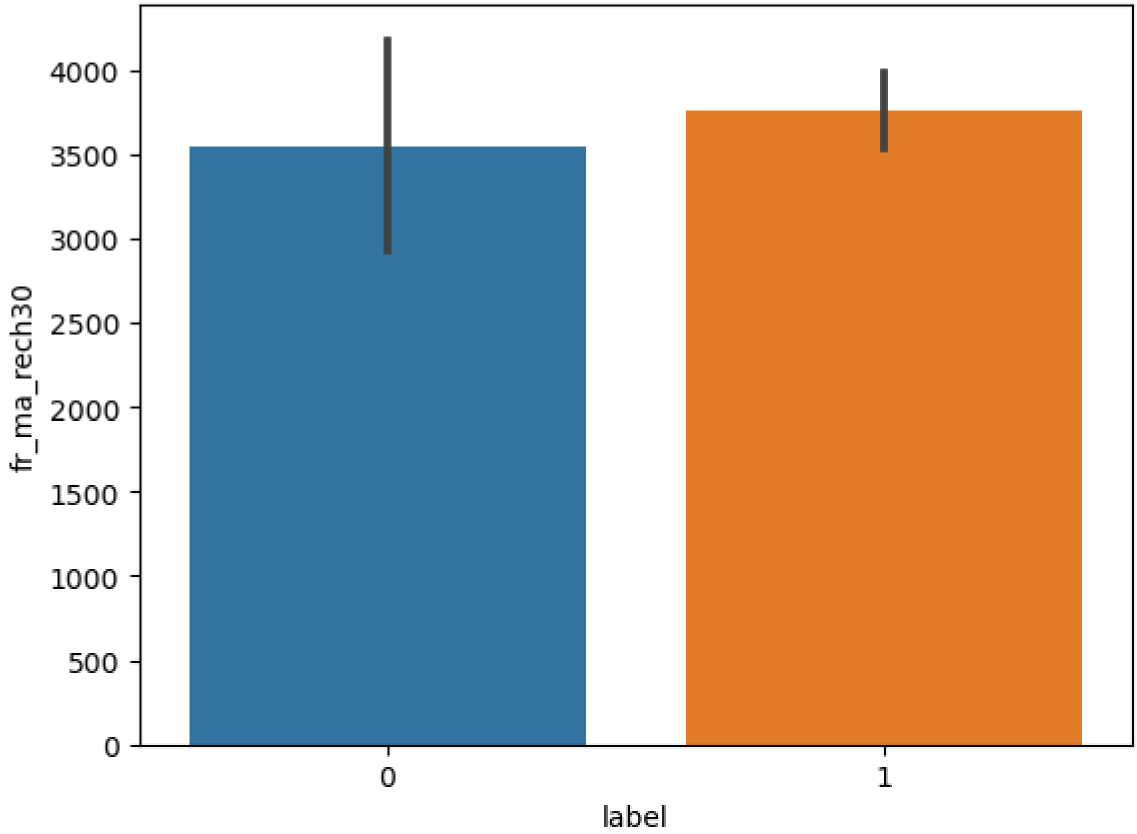

In [33]:
```
sns.barplot(y="last_rech_date_ma",x="label",data=df)
plt.show()
sns.barplot(y="last_rech_date_da",x="label",data=df)
plt.show()
#Number of days till last recharge of main account & data account is higher for nor
#outliers are present.
```



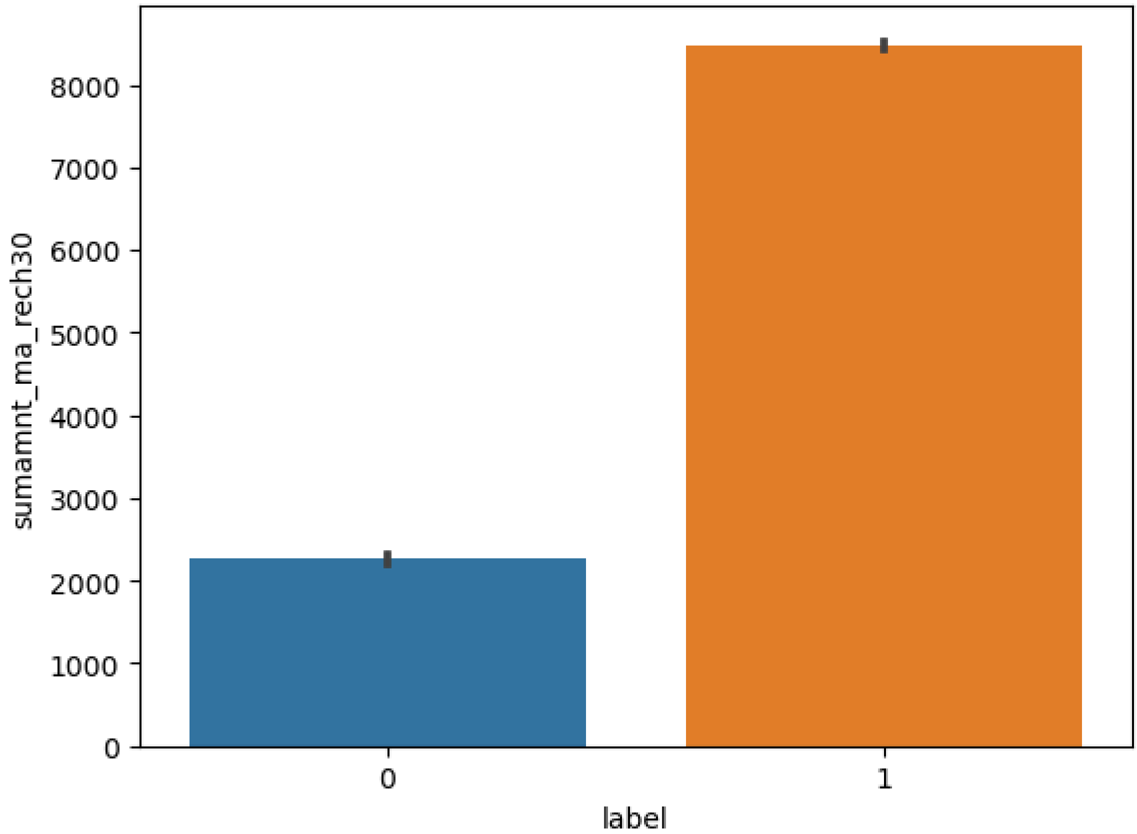


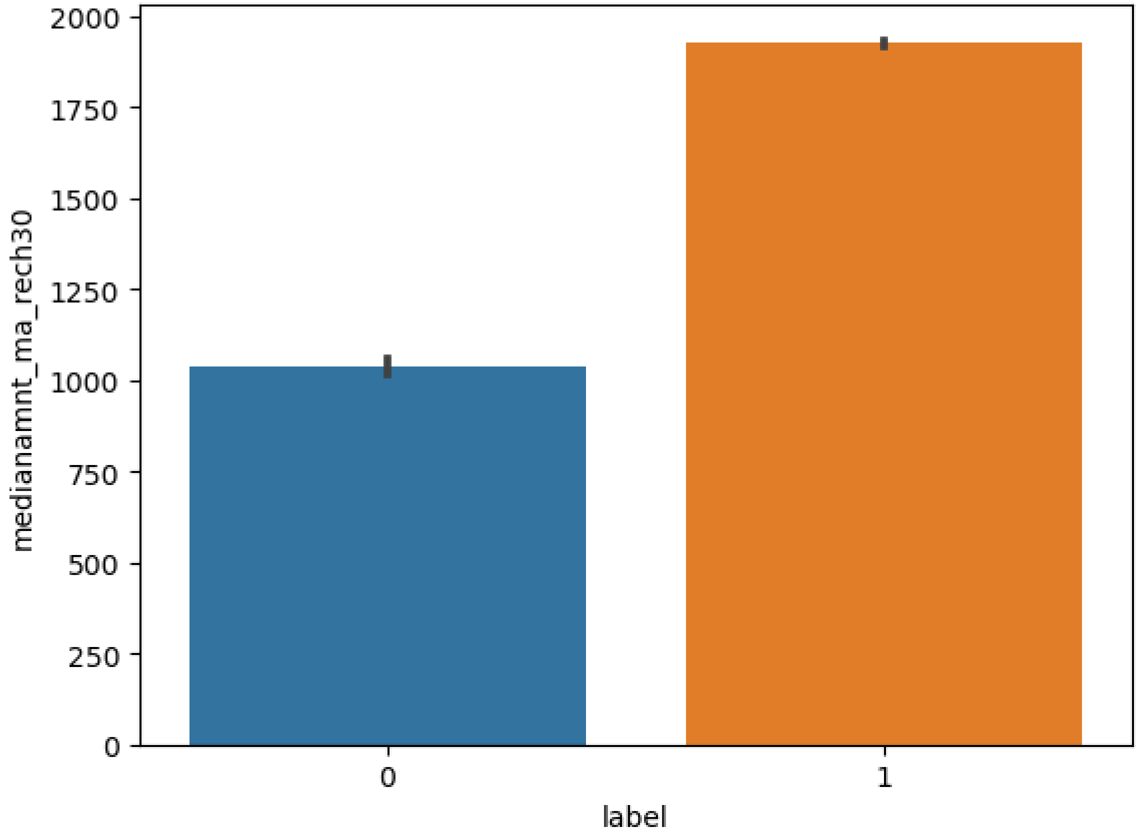

In [34]:
```
sns.barplot(y="cnt_ma_rech30",x="label",data=df)
plt.show()
sns.barplot(y="fr_ma_rech30",x="label",data=df)
plt.show()
#Number of times main account got recharged is higher for non defaulters in last 30
#Frequency of main account recharged in last 30 days is slight higher for non defau
```

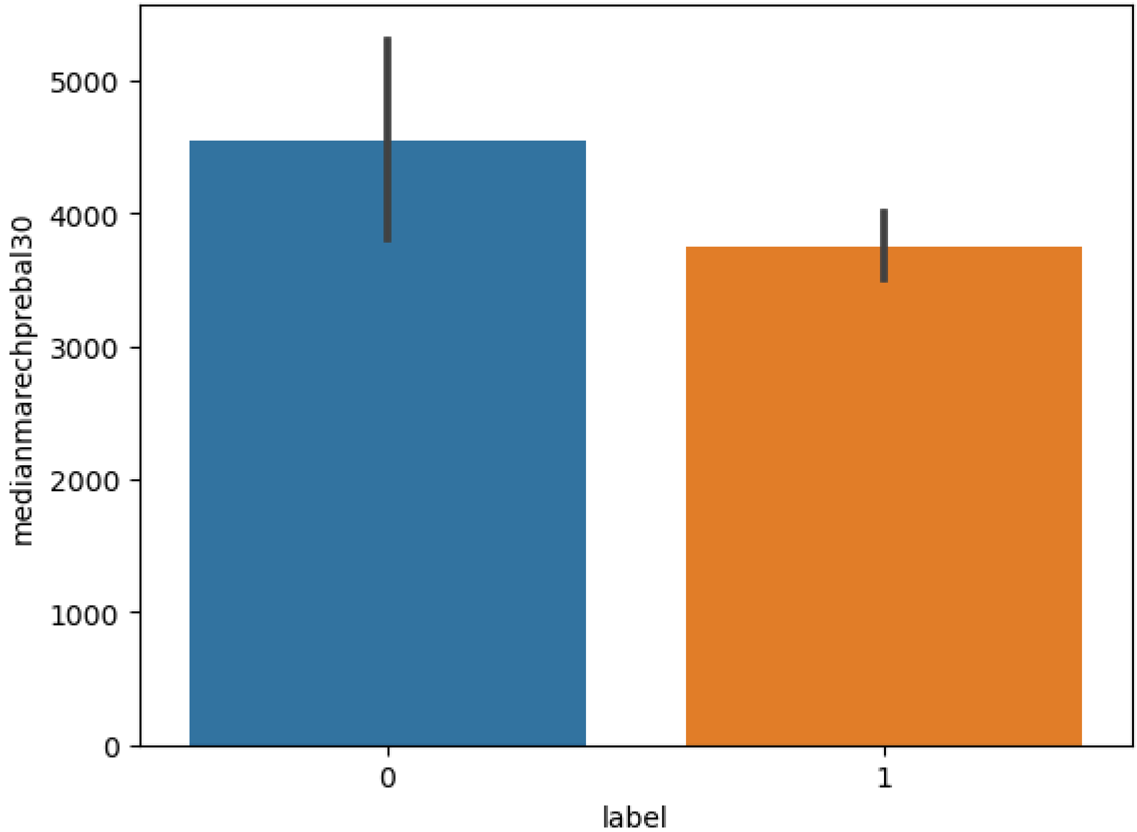




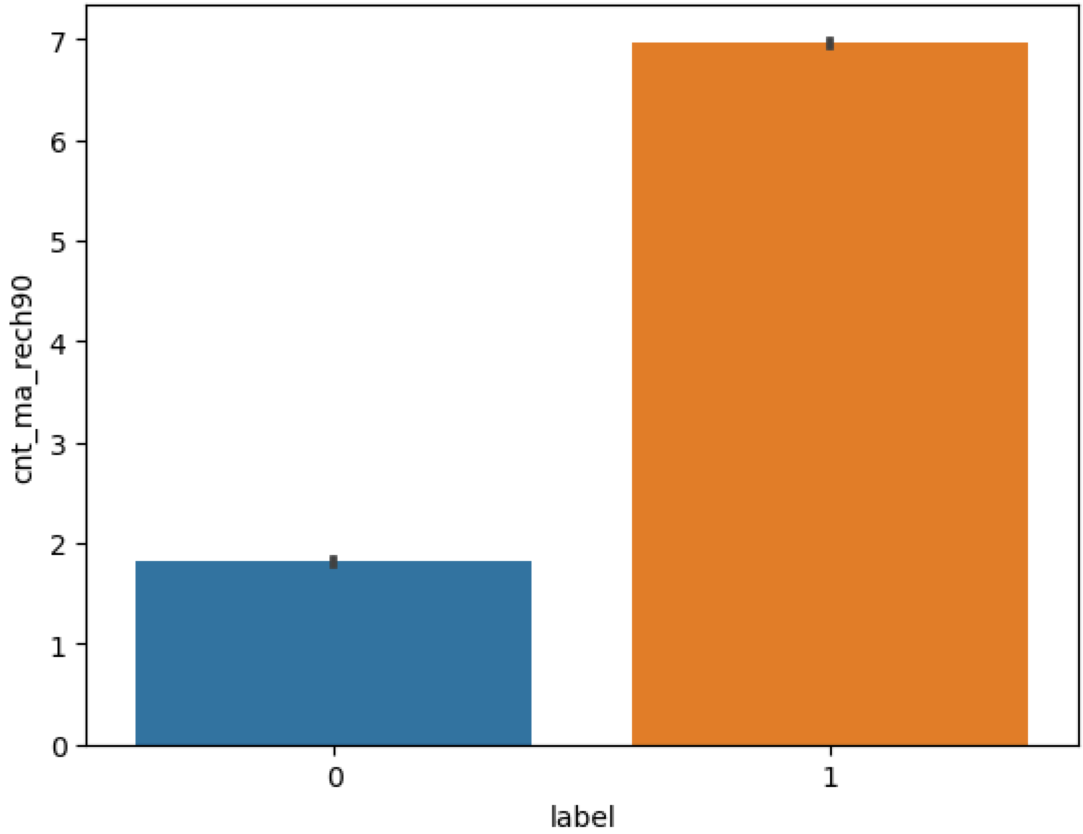

In [35]:
```
sns.barplot(y="sumamnt_ma_rech30",x="label",data=df)
plt.show()
sns.barplot(y="medianamnt_ma_rech30",x="label",data=df)
plt.show()
sns.barplot(y="medianmarechprebal30",x="label",data=df)
plt.show()
#Total amount of recharge in main account over last 30 days is higher for non defau
#Median of main account balance just before recharge in last 30 is higher for non d
#we also see outliers present in Median of main account balance just before recharg
```

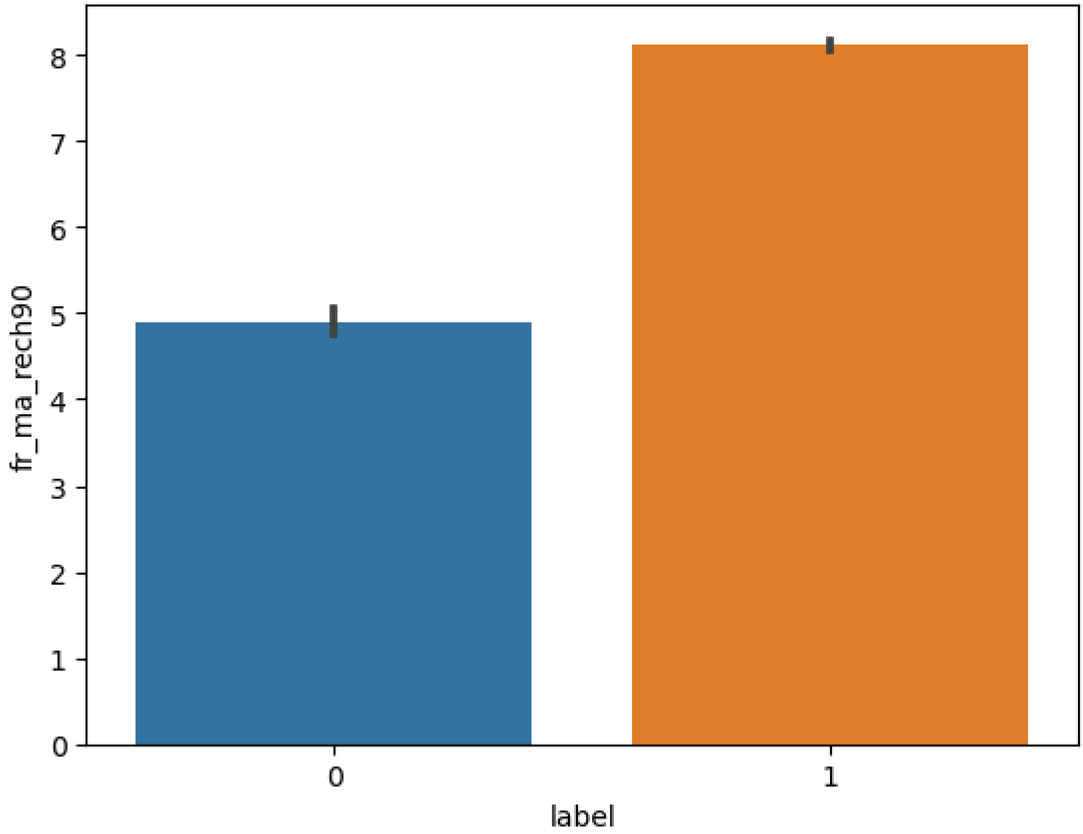


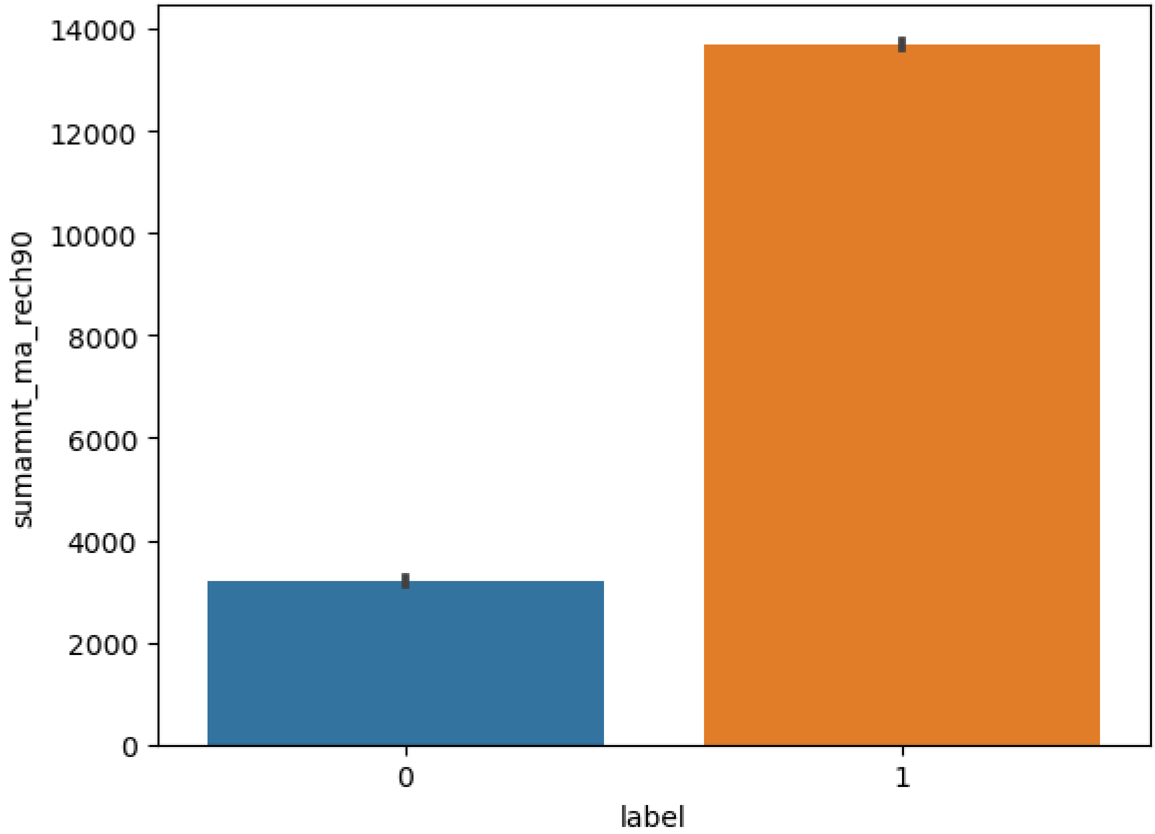


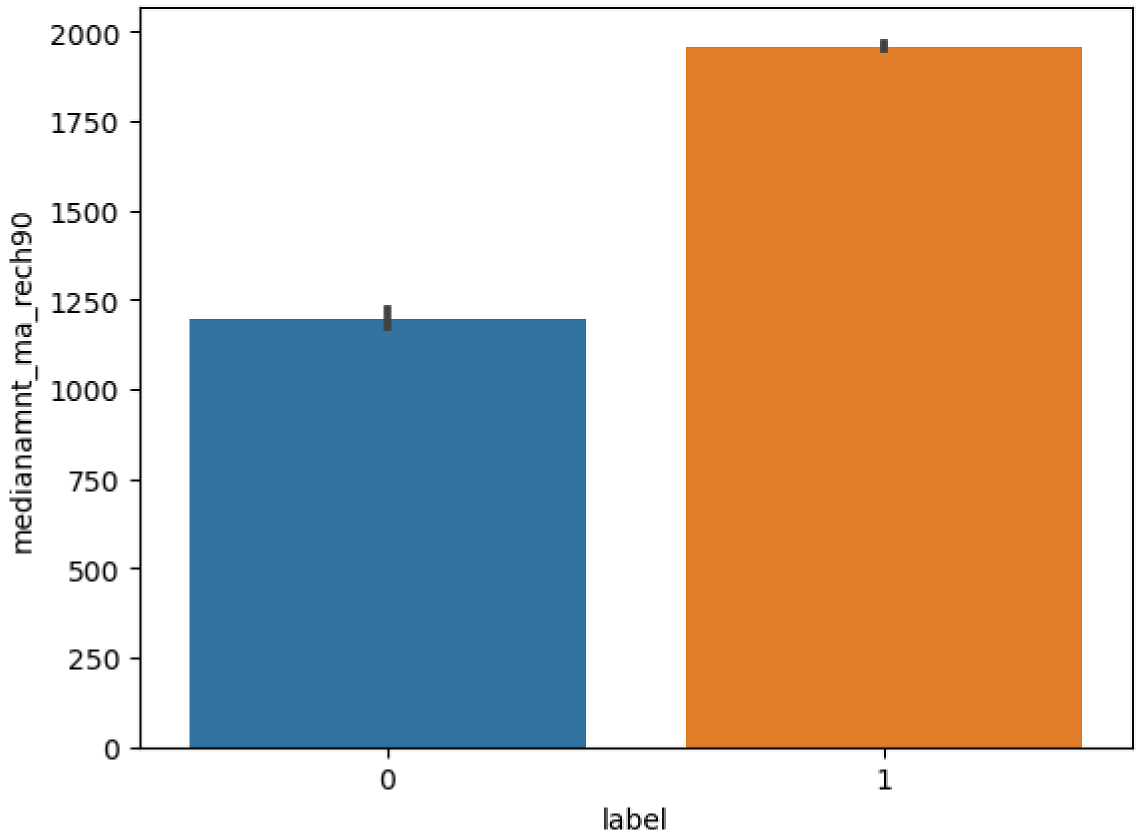

In [36]:
```
sns.barplot(y="cnt_ma_rech90",x="label",data=df)
plt.show()
sns.barplot(y="fr_ma_rech90",x="label",data=df)
plt.show()
sns.barplot(y="sumamnt_ma_rech90",x="label",data=df)
plt.show()
#Number of times main account got recharged is higher for non defaulters in last 90
#Frequency of main account recharged in last 90 days is slight higher for non defau
#Total amount of recharge in main account over last 90 days is higher for non defau
```

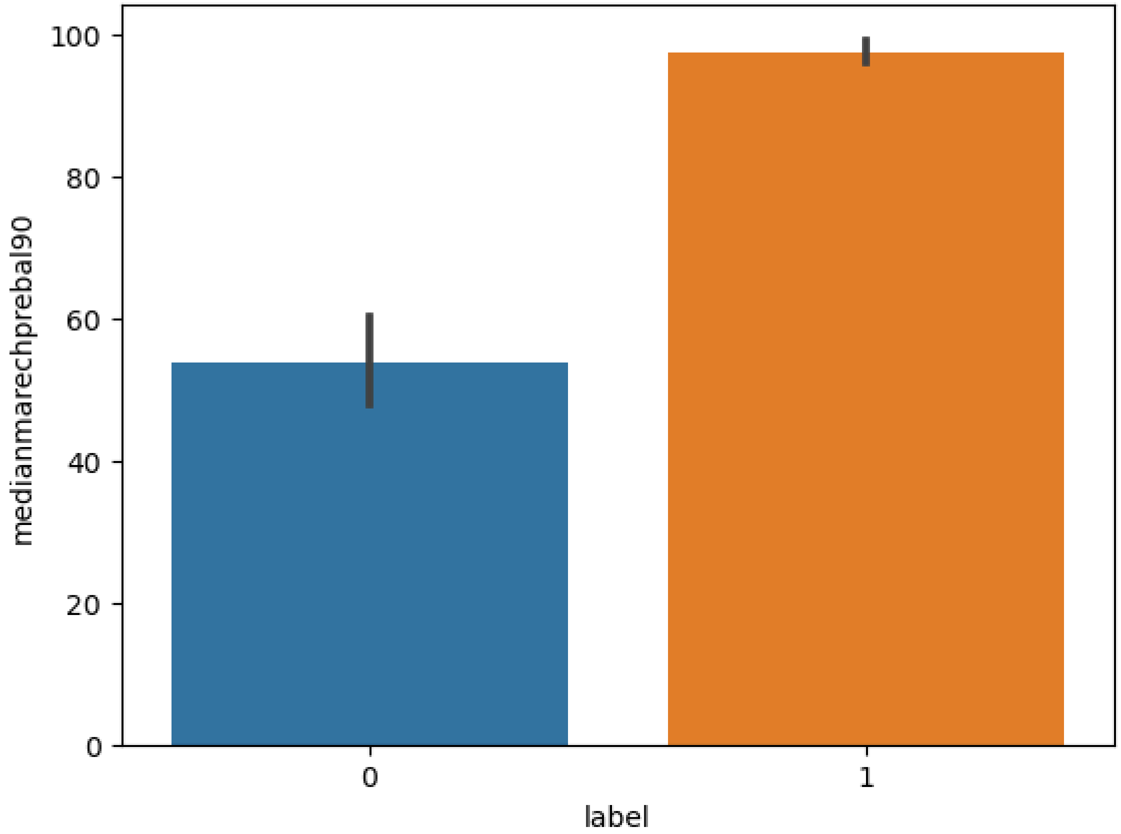

cnt_ma_rech90
0
1
2
3
4
5
6
7
0
1
label


fr_ma_rech90
0
1
2
3
4
5
6
7
8
0
1
label

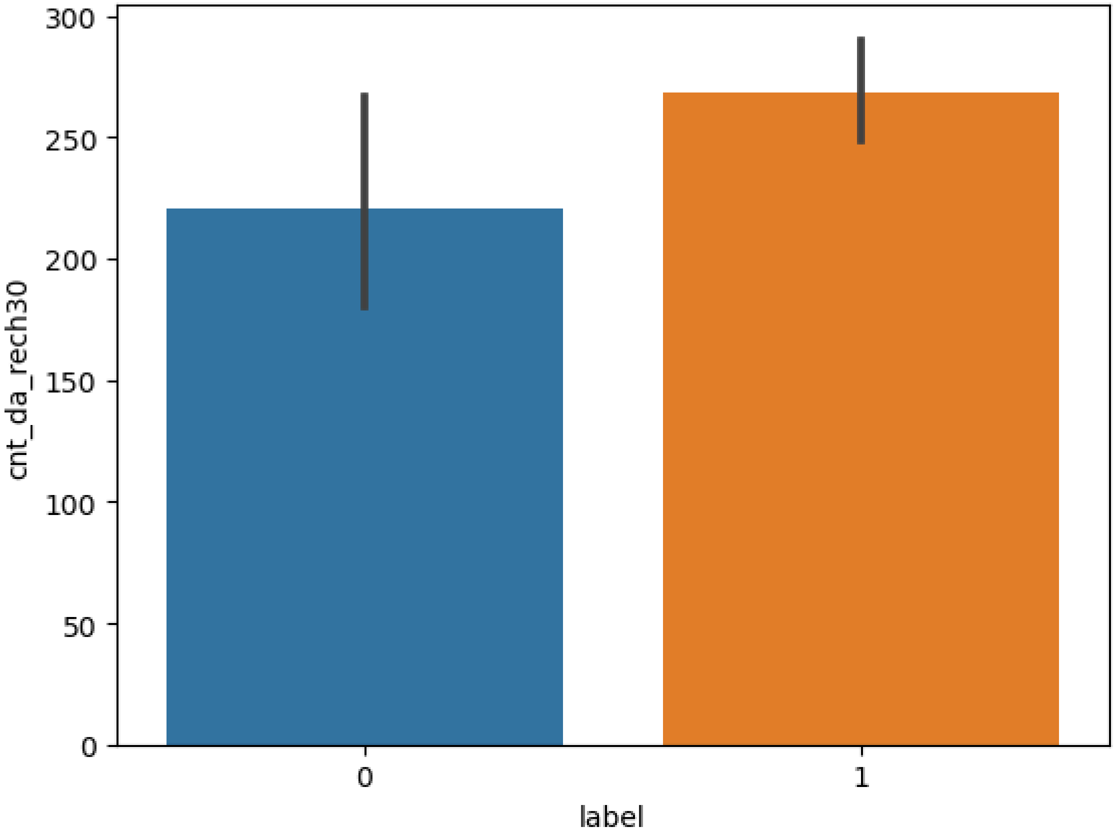


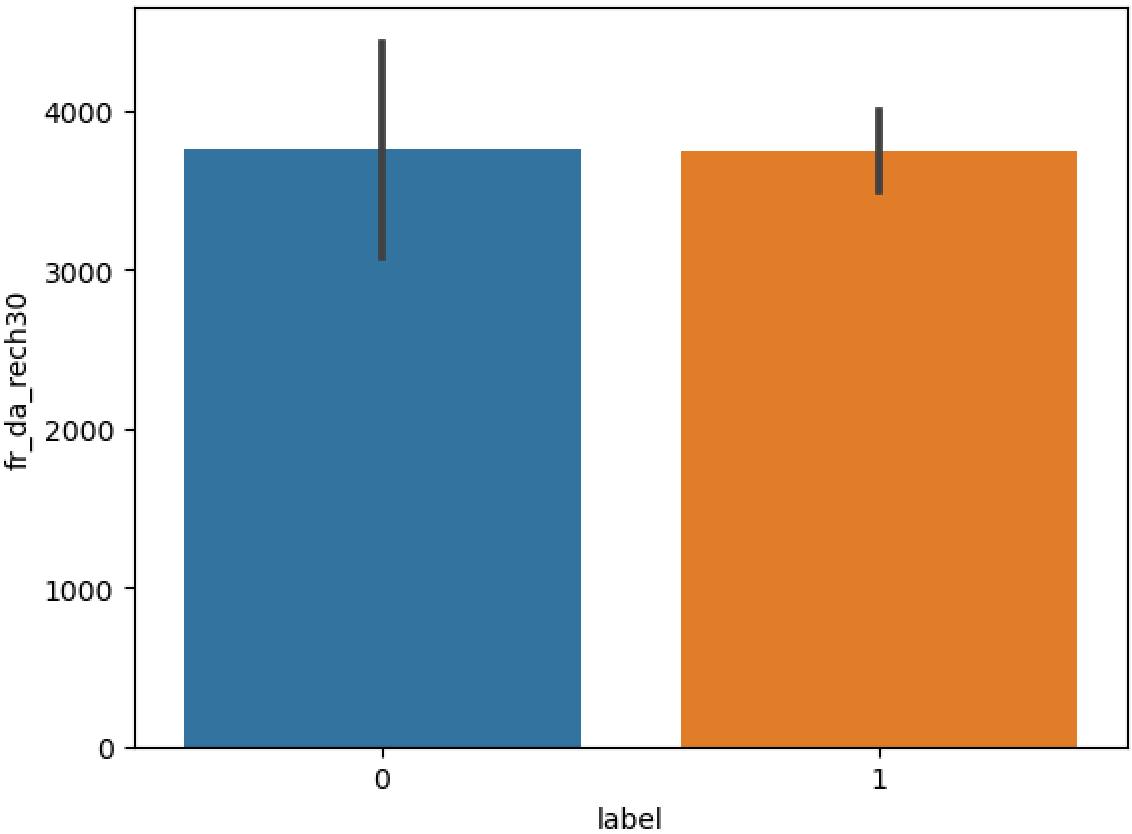

In [37]:
```
sns.barplot(y="medianamnt_ma_rech90",x="label",data=df)
plt.show()
sns.barplot(y="medianmarechprebal90",x="label",data=df)
plt.show()
#Median of main account balance just before recharge in last 90 is higher for non d
#we also see outliers present in Median of main account balance just before recharg
```

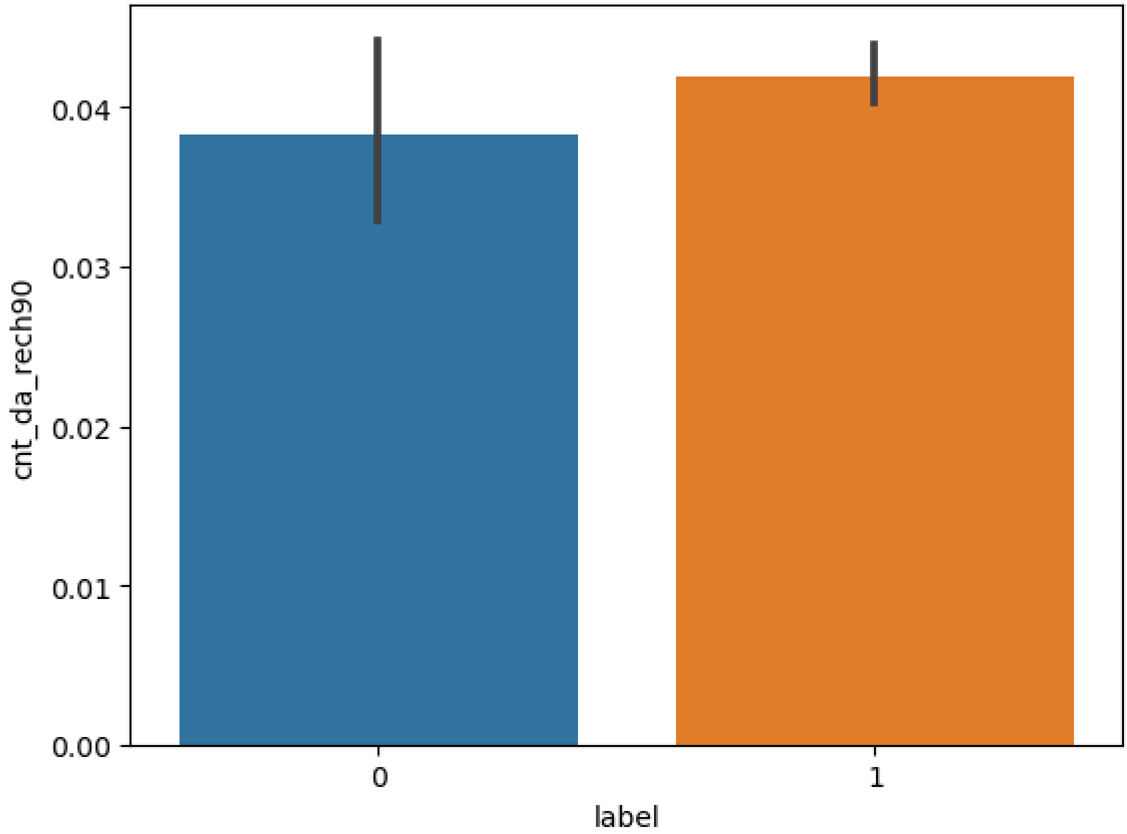


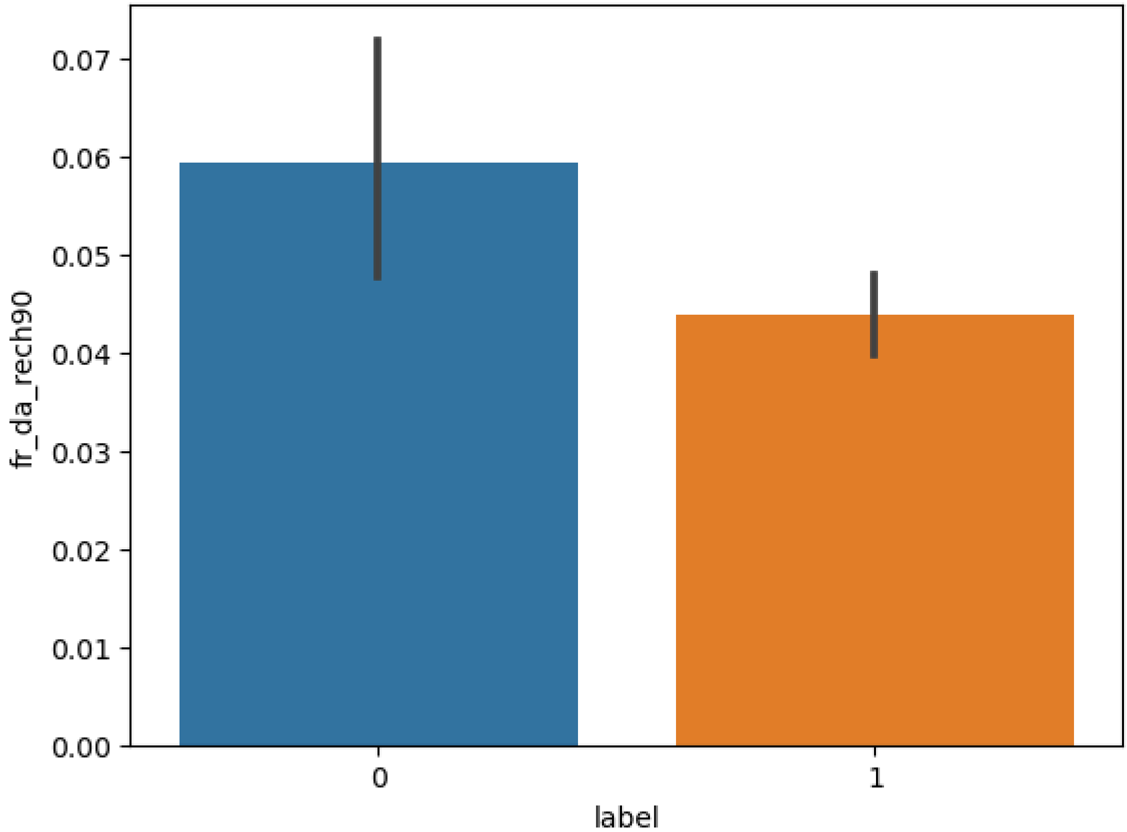



In [38]:
```
sns.barplot(y="cnt_da_rech30",x="label",data=df)
plt.show()
sns.barplot(y="fr_da_rech30",x="label",data=df)
plt.show()
sns.barplot(y="cnt_da_rech90",x="label",data=df)
plt.show()
sns.barplot(y="fr_da_rech90",x="label",data=df)
plt.show()
#non defaulters recharged the data account more than defaulters in last 30 days.
#Frequency of data account recharged is almost same defaulters and non defaulters i
#non defaulters recharged the data account more than defaulters in last 90 days.
#outliers are present
```


cnt_da_rech30
300
250
200
150
100
50
0
0
1
label

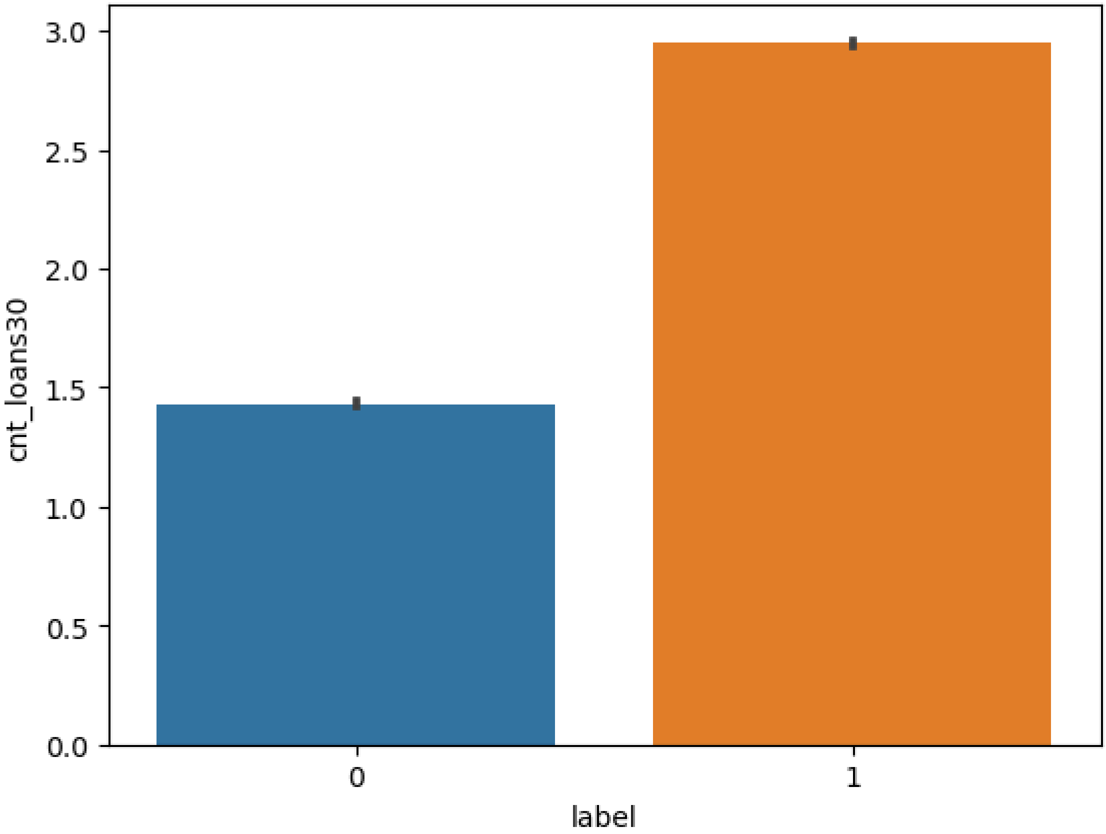

fr_da_rech30
4000
3000
2000
1000
0
0
1
label

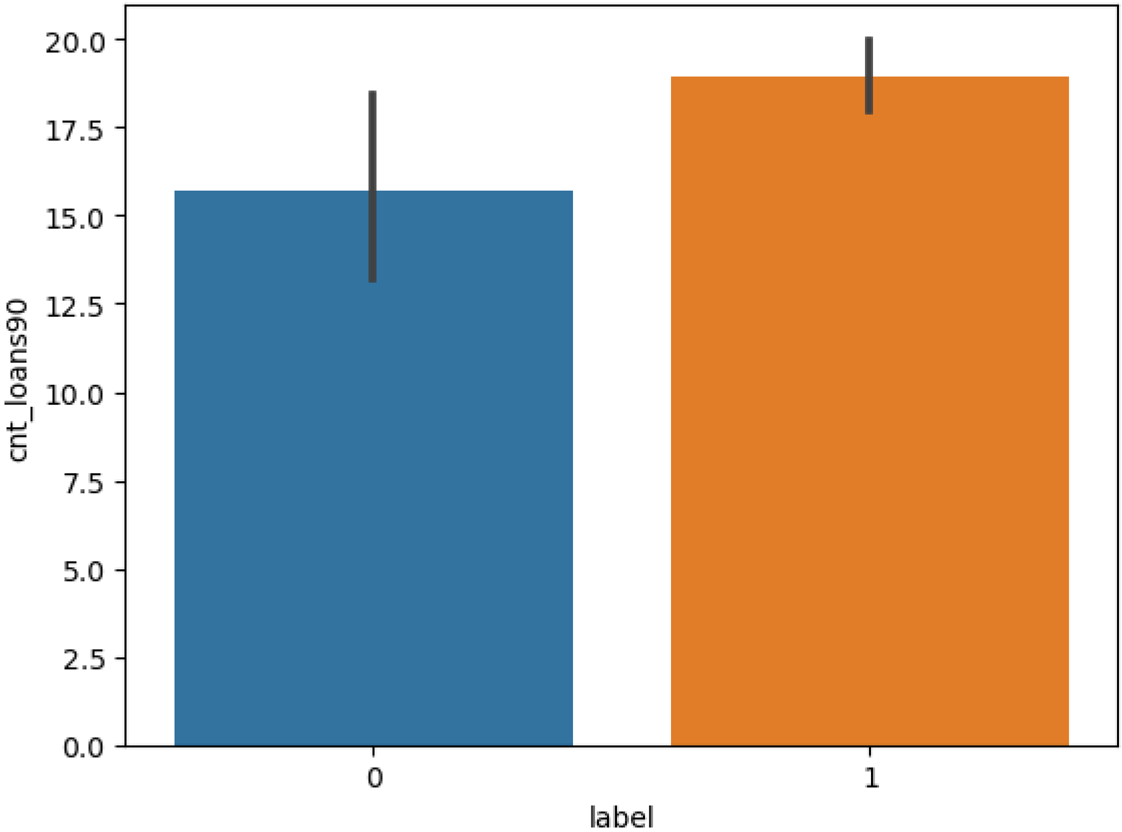


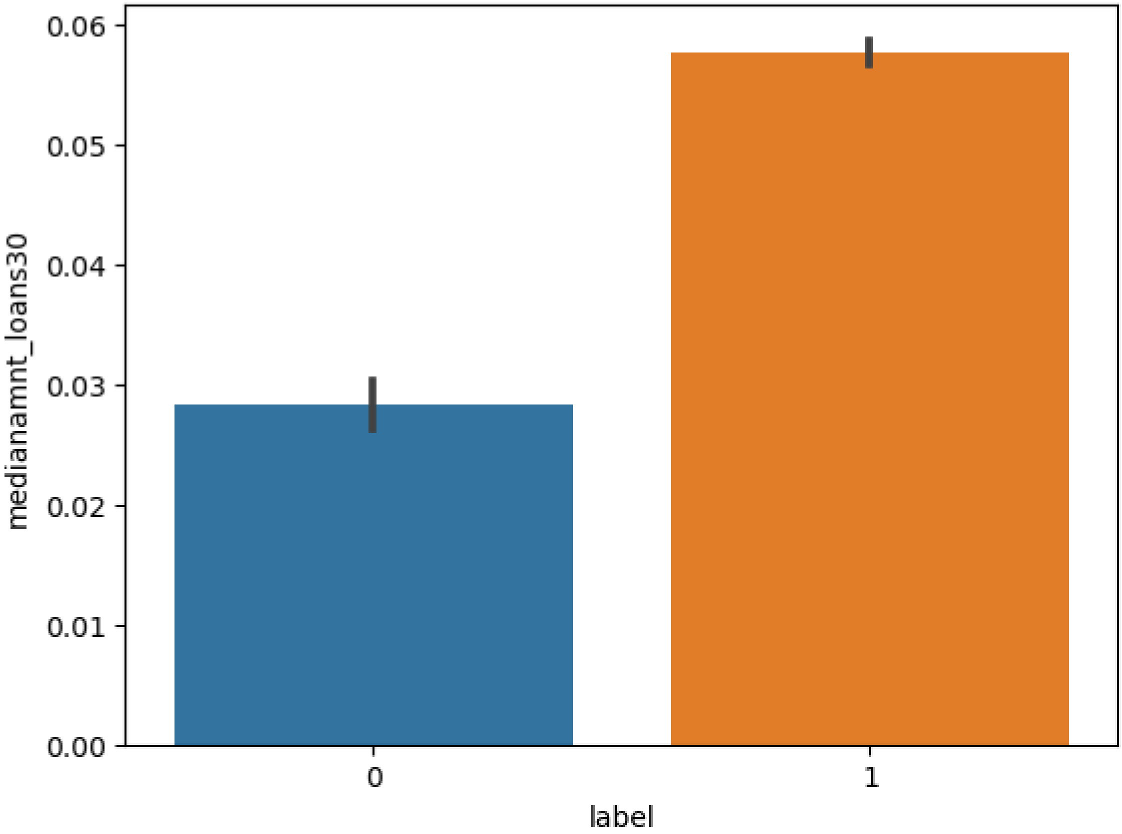

In [39]:

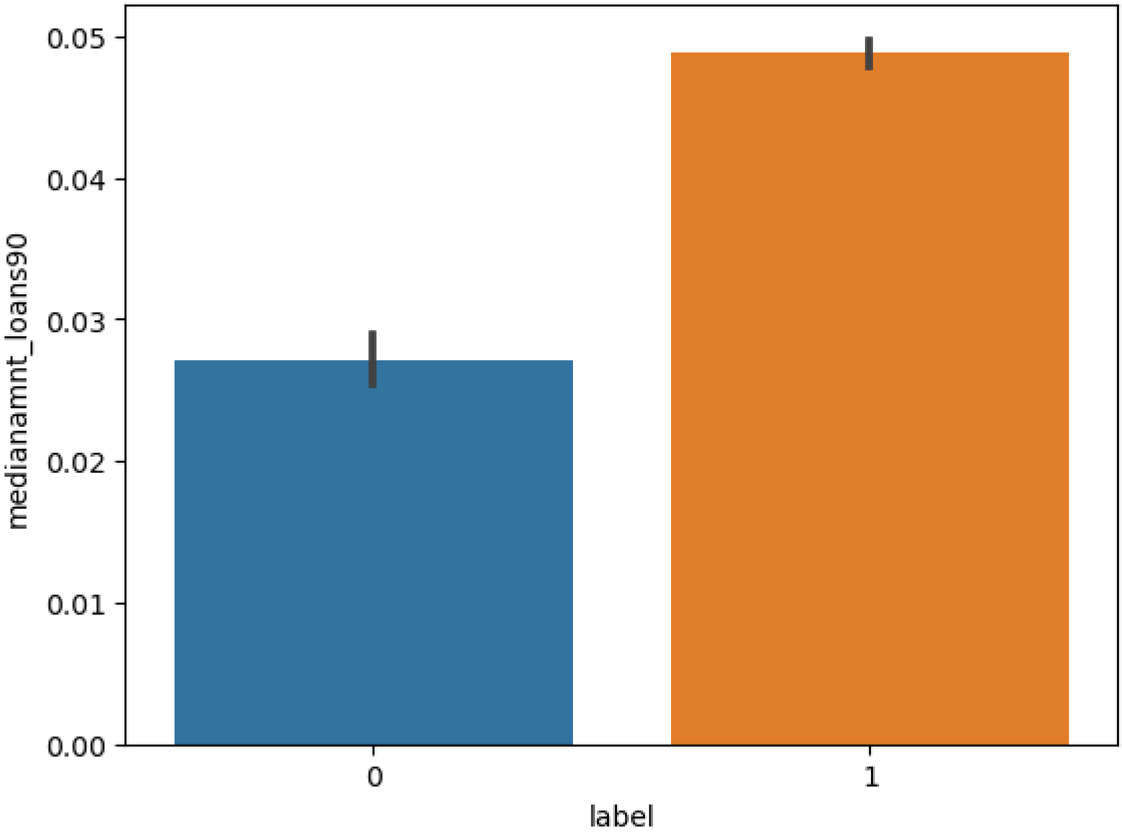
```
sns.barplot(y="cnt_loans30",x="label",data=df)
plt.show()
sns.barplot(y="cnt_loans90",x="label",data=df)
plt.show()
#Number of loans taken by user in last 30 & 90 days is higher for non defaulters.
#outliers are present in Number of loans taken by user in last 90 days
```



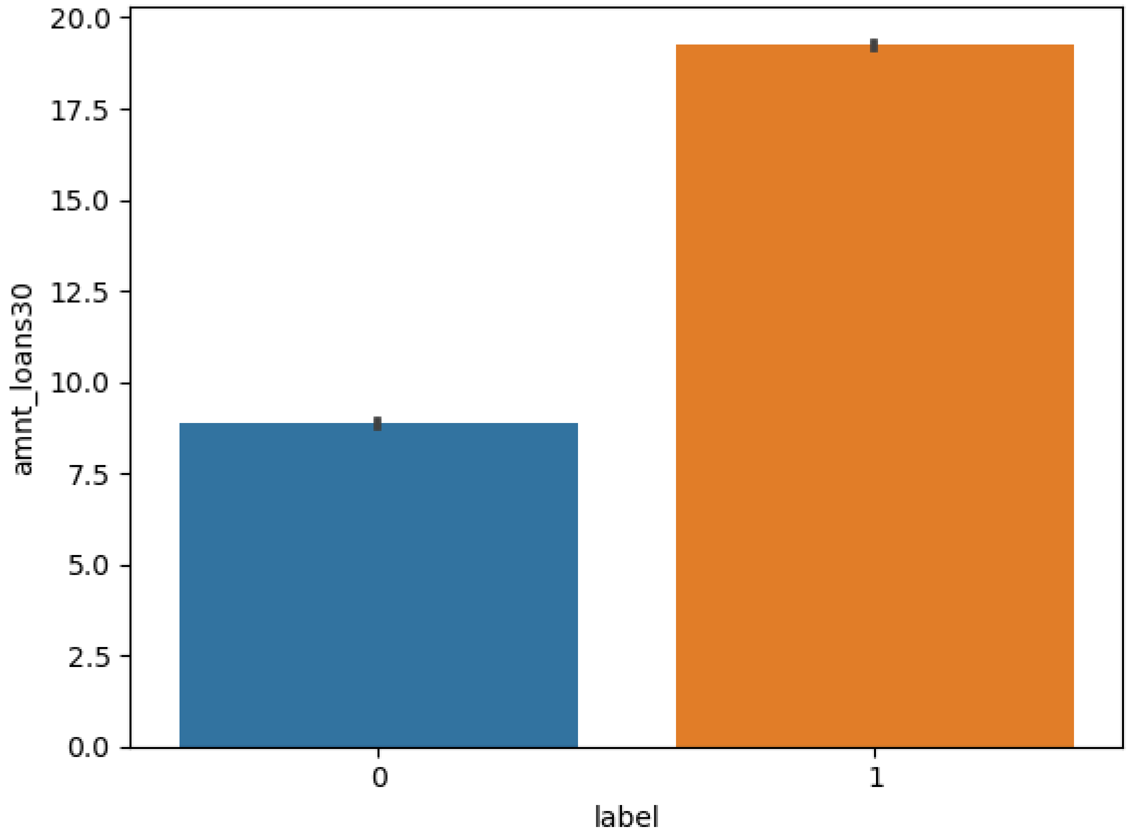



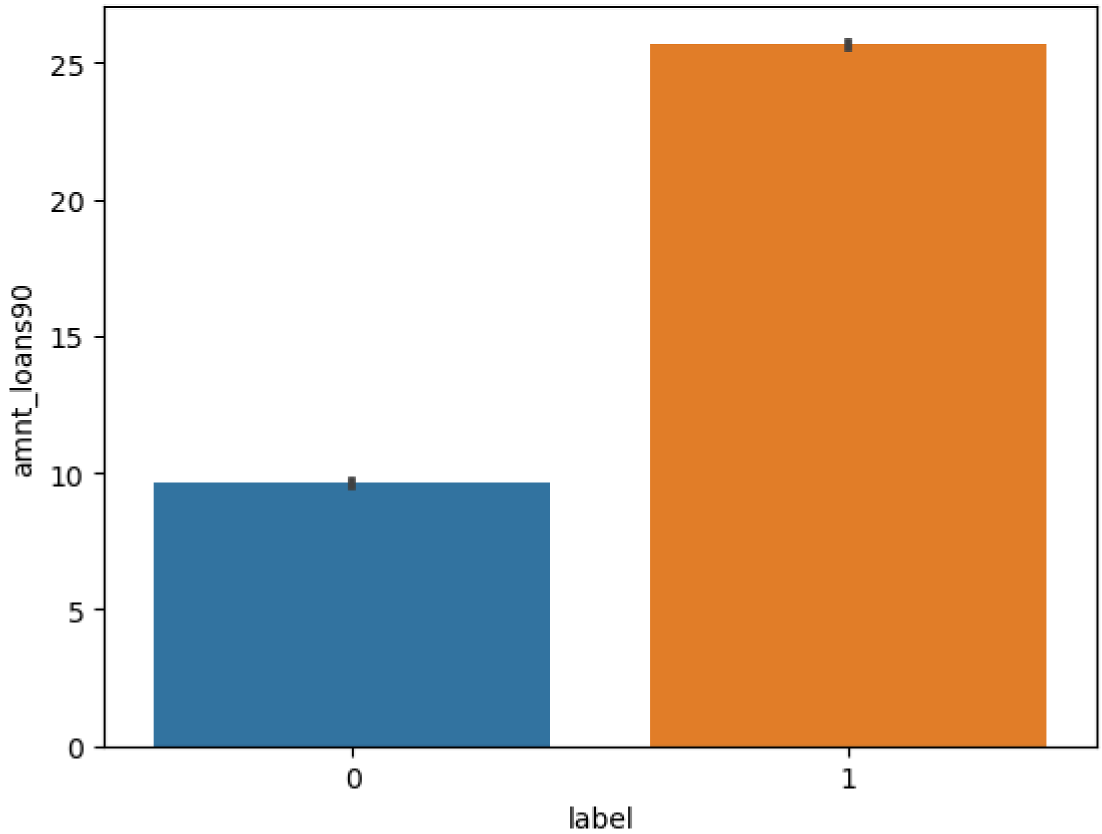

In [40]:
```
sns.barplot(y="medianamnt_loans30",x="label",data=df)
plt.show()
sns.barplot(y="medianamnt_loans90",x="label",data=df)
plt.show()
#Median of amounts of loan taken by the user in last 30 & 90 days is higher for nor
```



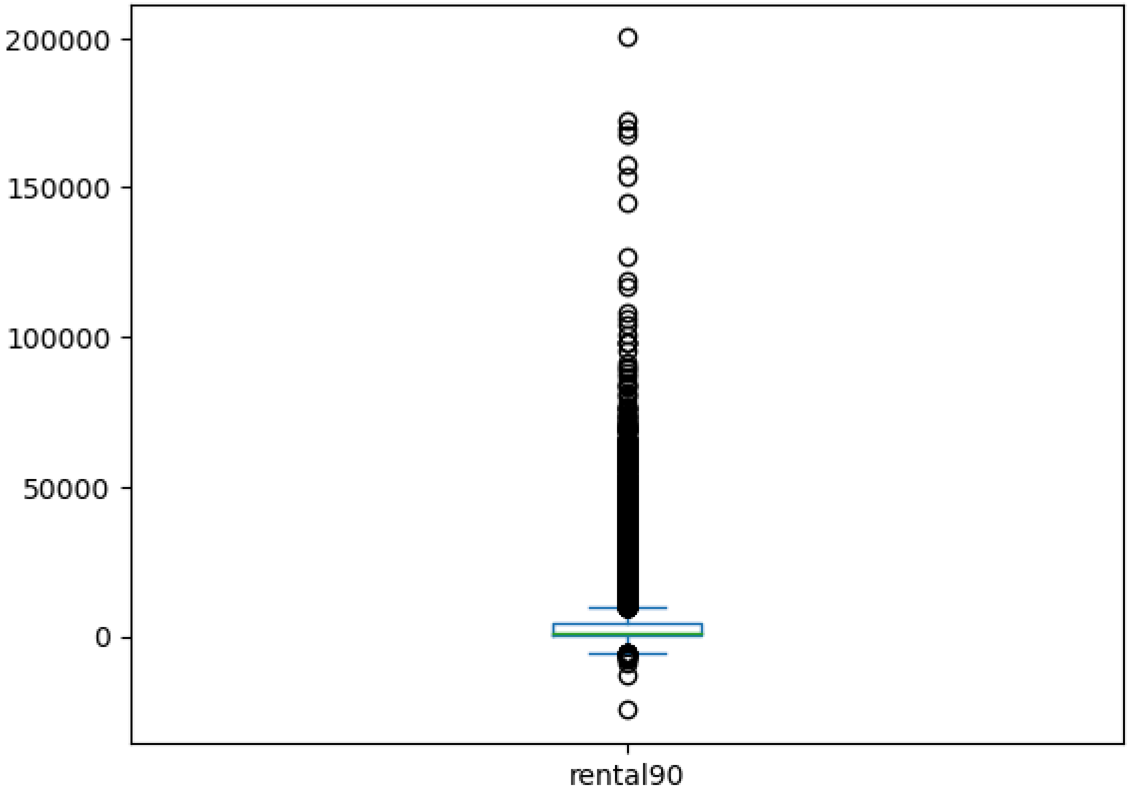

In [41]:
```
sns.barplot(y="amnt_loans30",x="label",data=df)
plt.show()
sns.barplot(y="amnt_loans90",x="label",data=df)
plt.show()
#Total amount of loans taken by user in last 30 & 90 days is higher for non default
```




In [42]:
```
df['rental90'].plot.box()
```

Out[42]: <Axes: >




In [43]:
```
df.skew()
```

Out[43]:
```
label                   -2.270254
msisdn                   0.000717
aon                     10.392949
daily_decr30             3.946230
daily_decr90             4.252565
rental30                 4.521929
rental90                 4.437681
last_rech_date_ma       14.790974
last_rech_date_da       14.814857
last_rech_amt_ma         3.781149
cnt_ma_rech30            3.283842
fr_ma_rech30            14.772833
sumamnt_ma_rech30        6.386787
medianamnt_ma_rech30     3.512324
medianmarechprebal30    14.779875
cnt_ma_rech90            3.425254
fr_ma_rech90             2.285423
sumamnt_ma_rech90        4.897950
medianamnt_ma_rech90     3.752706
medianmarechprebal90    44.880503
cnt_da_rech30           17.818364
fr_da_rech30            14.776430
cnt_da_rech90           27.267278
fr_da_rech90            28.988083
cnt_loans30              2.713421
amnt_loans30             2.975719
maxamnt_loans30         17.658052
medianamnt_loans30       4.551043
cnt_loans90             16.594408
amnt_loans90             3.150006
maxamnt_loans90          1.678304
medianamnt_loans90       4.895720
payback30                8.310695
payback90                6.899951
pdate                    0.116409
dtype: float64
```

In [44]:
```
from scipy.stats import zscore
zscore=abs(zscore(df))
print(df.shape)
```

```
(209593, 35)
```

In [45]:
```
threshold=3
print(np.where(zscore>3))
```

```
(array([    21,     22,     22, ..., 209586, 209587, 209587], dtype=int64), array
([16, 16, 33, ..., 29, 27, 31], dtype=int64))
```

In [46]:
```
df_new=df[(zscore<3).all(axis=1)]
```

In [47]:
```
df.shape
```

Out[47]:
```
(209593, 35)
```

In [48]:
```
df_new.shape
```

Out[48]:
```
(161465, 35)
```

In [49]:
```
#from the above the 48128 outliers are get removed
```

In [50]:
```
from sklearn.decomposition import PCA
```

In [51]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(df_new)
x=pd.DataFrame(x,columns=df_new.columns)
```

In [52]:
```
x.shape
```

Out[52]:
```
(161465, 35)
```

In [53]:
```
pca=PCA(n_components=10)
```

In [54]:
```
x=pca.fit_transform(x)
```

In [55]:
```
y=df_new.iloc[:,0].values
```

In [56]:
```
y
```

Out[56]:
```
array([0, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [57]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
y
```

Out[57]:
```
array([0, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [58]:
```
from sklearn.model_selection import train_test_split,cross_val_score
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=9,str
```

In [59]:
```
print(x_train.shape,x_test.shape)
```

```
(113025, 10) (48440, 10)
```

In [60]:
```
print(y_train.shape,y_test.shape)
```

```
(113025,) (48440,)
```

In [61]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```

In [62]:
```
KNN=KNeighborsClassifier(n_neighbors=6)
LR=LogisticRegression()
DT=DecisionTreeClassifier(random_state=6)
GNB=GaussianNB()
```

In [63]:
```
models = []
models.append(('KNeighborsClassifier',KNN))
models.append(('LogisticRegression',LR))
models.append(('DecisionTreeClassifier',DT))
models.append(('GaussianNB',GNB))
```

In [64]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,r
```

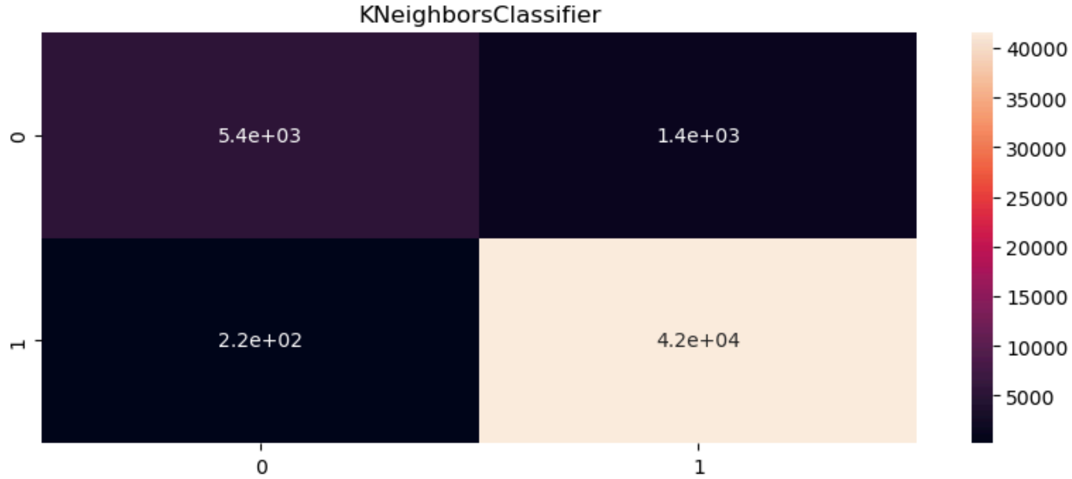
In [65]:
```
Model = []
score = []
cvs=[]
rocscore=[]
```

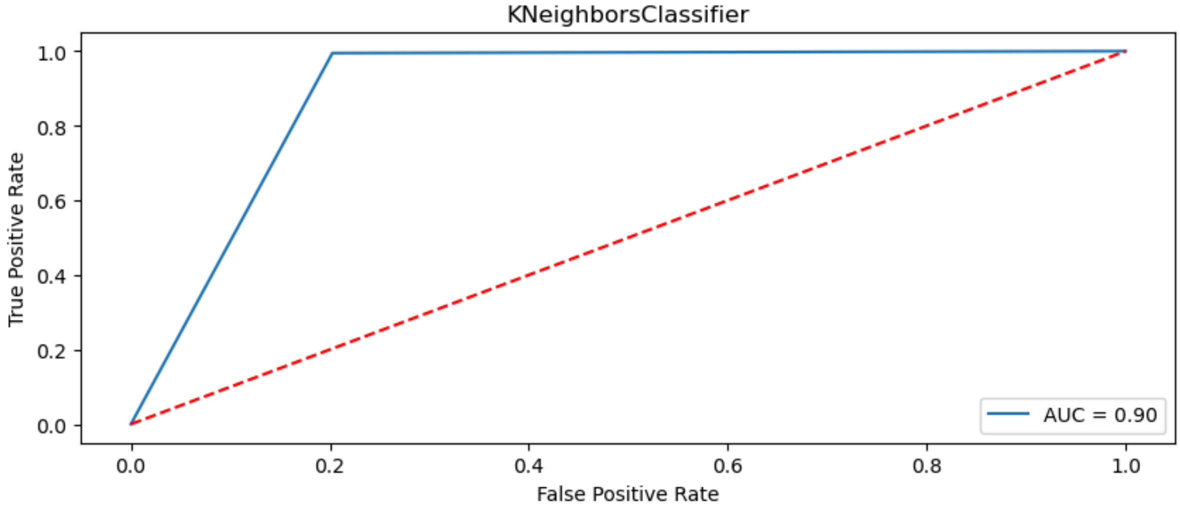
```
for name,model in models:
    print('*******************',name,'*******************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=accuracy_score(y_test,pre)
    print('Accuracy_Score = ',AS)
    score.append(AS*100)
    print('\n')
    sc = cross_val_score(model, x, y, cv=10, scoring='accuracy').mean()
    print('Cross_Val_Score = ',sc)
    cvs.append(sc*100)
    print('\n')
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,pre)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    print('roc_auc_score = ',roc_auc)
    rocscore.append(roc_auc*100)
    print('\n')
    print('Classification_report\n',classification_report(y_test,pre))
    print('\n')
    cm=confusion_matrix(y_test,pre)
    print(cm)
    print('\n')
    plt.figure(figsize=(10,40))
    plt.subplot(911)
    plt.title(name)
    print(sns.heatmap(cm,annot=True))
    plt.subplot(912)
    plt.title(name)
    plt.plot(false_positive_rate, true_positive_rate, label='AUC = %0.2f'% roc_auc)
    plt.plot([0,1],[0,1],'r--')
    plt.legend(loc='lower right')
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    print('\n\n')
```

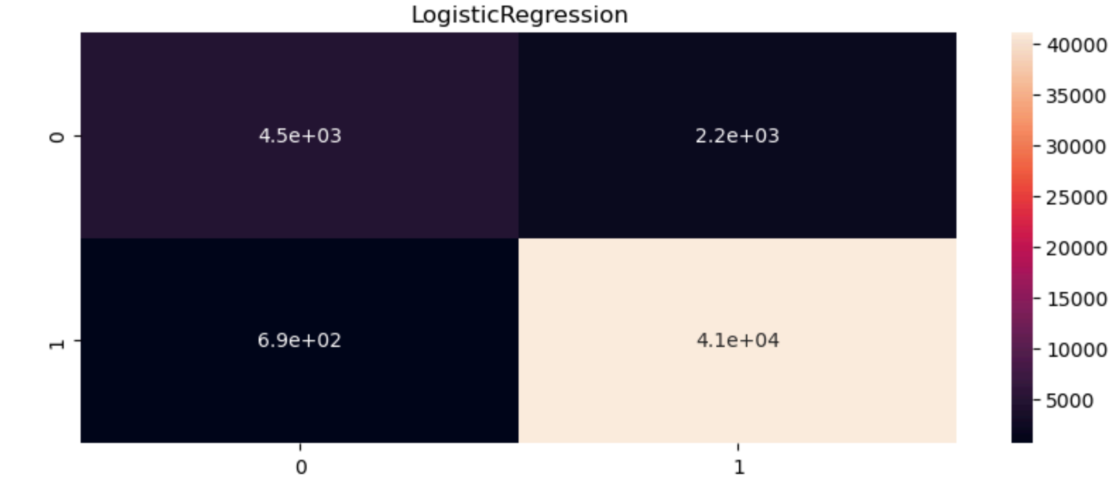
```
****************** KNeighborsClassifier ******************


KNeighborsClassifier(n_neighbors=6)


Accuracy_Score =  0.967382328654005


Cross_Val_Score =  0.9690397401073231


roc_auc_score =  0.8961104785874081


Classification_report
               precision    recall  f1-score   support

           0       0.96      0.80      0.87      6720
           1       0.97      0.99      0.98     41720

    accuracy                           0.97     48440
   macro avg       0.96      0.90      0.93     48440
weighted avg       0.97      0.97      0.97     48440



[[ 5359  1361]
 [  219 41501]]


Axes(0.125,0.807358;0.62x0.0726415)



****************** LogisticRegression ******************


LogisticRegression()


Accuracy_Score =  0.9402146985962014


Cross_Val_Score =  0.940098479816586


roc_auc_score =  0.8275936800894854


Classification_report
               precision    recall  f1-score   support

           0       0.87      0.67      0.76      6720
           1       0.95      0.98      0.97     41720

    accuracy                           0.94     48440
   macro avg       0.91      0.83      0.86     48440
weighted avg       0.94      0.94      0.94     48440



[[ 4514  2206]
 [  690 41030]]
```

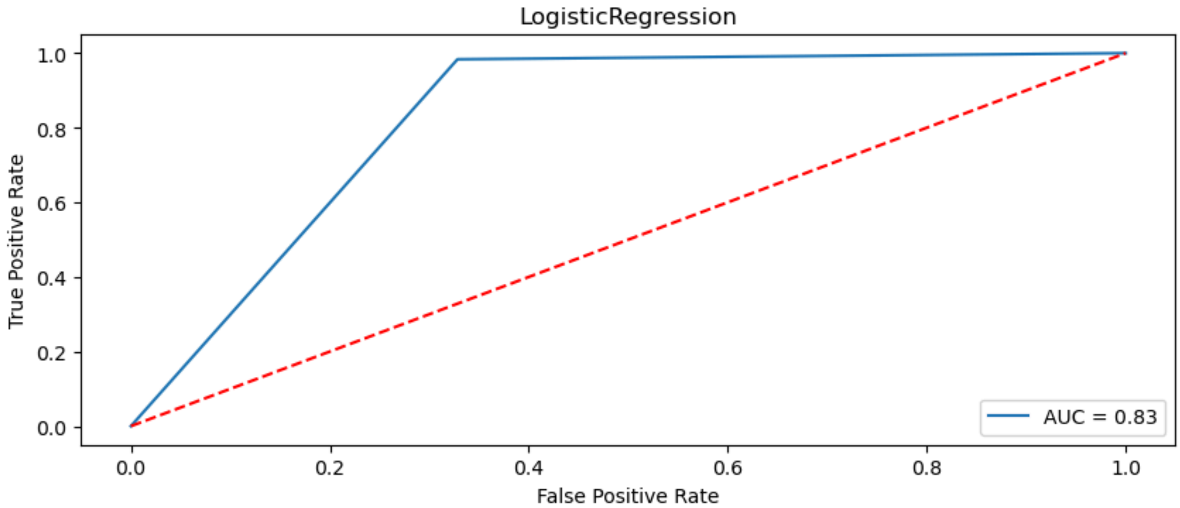
```
Axes(0.125,0.807358;0.62x0.0726415)


******************* DecisionTreeClassifier *******************

DecisionTreeClassifier(random_state=6)

Accuracy_Score =  0.9571635012386458

Cross_Val_Score =  0.9583748889789048

roc_auc_score =  0.9094034635666347

Classification_report
               precision    recall  f1-score   support

           0       0.85      0.84      0.85      6720
           1       0.97      0.98      0.98     41720

    accuracy                           0.96     48440
   macro avg       0.91      0.91      0.91     48440
weighted avg       0.96      0.96      0.96     48440



[[ 5667  1053]
 [ 1022 40698]]

Axes(0.125,0.807358;0.62x0.0726415)


******************* GaussianNB *******************

GaussianNB()

Accuracy_Score =  0.829335260115607

Cross_Val_Score =  0.8320193660635148

roc_auc_score =  0.8022990572067752

Classification_report
               precision    recall  f1-score   support

           0       0.43      0.76      0.55      6720
           1       0.96      0.84      0.89     41720

    accuracy                           0.83     48440
   macro avg       0.70      0.80      0.72     48440
```

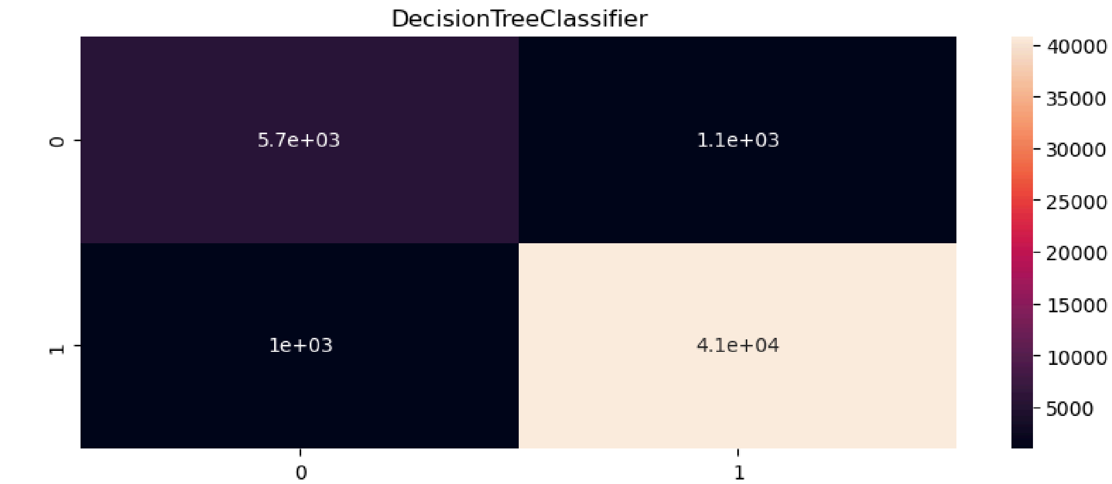
```
weighted avg       0.88      0.83      0.85     48440


[[ 5140  1580]
 [ 6687 35033]]


Axes(0.125,0.807358;0.62x0.0726415)
```

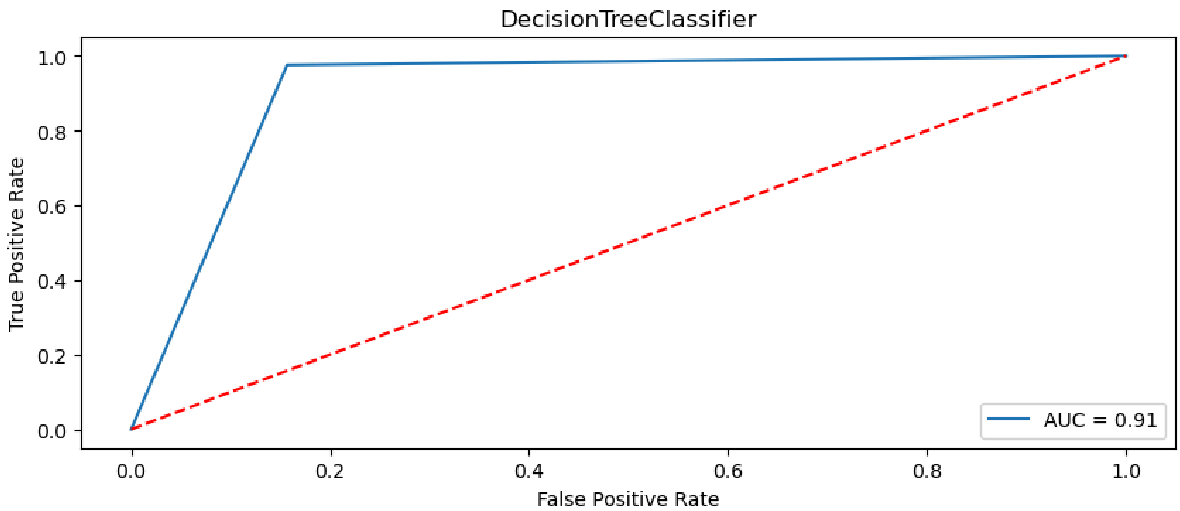


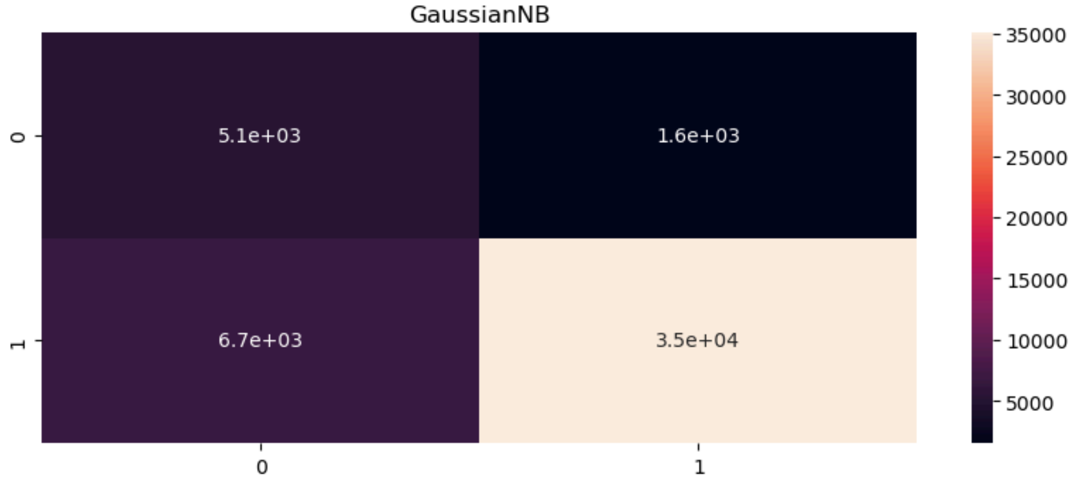



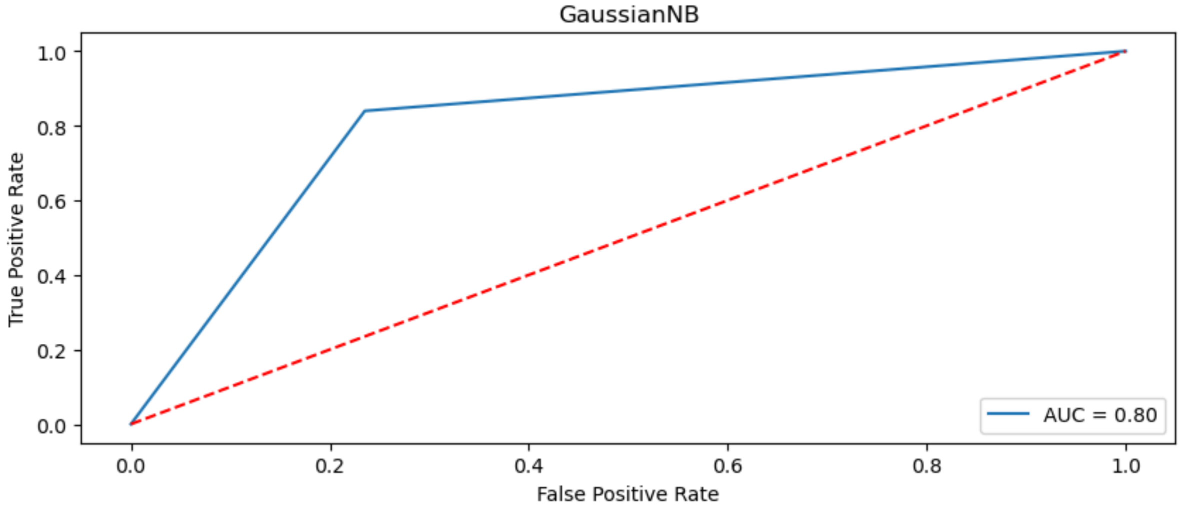

LogisticRegression
4.5e+03
2.2e+03
6.9e+02
4.1e+04
0
1
0
1
40000
35000
30000
25000
20000
15000
10000
5000


LogisticRegression
True Positive Rate
False Positive Rate
AUC = 0.83
0.0
0.2
0.4
0.6
0.8
1.0


DecisionTreeClassifier
5.7e+03
1.1e+03
1e+03
4.1e+04
0
1
0
1
40000
35000
30000
25000
20000
15000
10000
5000


DecisionTreeClassifier
True Positive Rate
False Positive Rate
AUC = 0.91
0.0
0.2
0.4
0.6
0.8
1.0




In [66]:
```
result = pd.DataFrame({'Model': Model, 'Accuracy_score': score,'Cross_val_score':cv
result
```

Out[66]:

| | Model | Accuracy_score | Cross_val_score | Roc_auc_curve |
|---|---|---|---|---|
| 0 | KNeighborsClassifier | 96.738233 | 96.903974 | 89.611048 |
| 1 | LogisticRegression | 94.021470 | 94.009848 | 82.759368 |
| 2 | DecisionTreeClassifier | 95.716350 | 95.837489 | 90.940346 |
| 3 | GaussianNB | 82.933526 | 83.201937 | 80.229906 |

Since from the above table, it's clear that KNeighborsClassifier,LogisticRegression,DecisionTreeClassifier and GaussianNB all are performing very well. KNeighborsClassifier is being chosen as the final model because it perform well on the dataset Accuracy_score = 96.75 Cross_val_score = 96.90 Roc_auc_curve = 89.63

In [67]:
```
parameters={
    'n_estimators':[100,200],
    'learning_rate':[0.001,0.01,0.1,0.2,0.5],
    'algorithm':['SAMME', 'SAMME.R']
}
```

In [ ]: