Predicting Coupon Usage

1 Bussiness Problems

ABC family of companies is looking to put a new product they tested it in some new locations and though it is a great product people are not buying it. ABC family of companies want to put out coupons for the product to generate buzz behind the product, but they only want to send coupons customers who are likely to use the coupon.

2 Data Cleaning

2.1 Import

All imports will be located below.

In [70]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt #Draws graphes
import seaborn as sns #Draws intuitive graphs
import graphviz
np.random.seed(0)
#splits data
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import cross_val_score
#Creates accuracy score, confustion matris, and classification repor
from sklearn.metrics import accuracy_score, confusion_matrix, classificat
#creates decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.tree import plot_tree
from sklearn.preprocessing import MinMaxScaler
#creaters Bagging Classifier and Random Forest
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve, auc
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
import warnings #Removes warnings


warnings.filterwarnings("ignore")
%matplotlib inline
```

executed in 46ms, finished 12:04:47 2021-07-09

In [71]:

```
data = pd.read_csv(r"C:\Users\laure\Flatiron\Phase3-project\python_upload
data.head()
```

executed in 227ms, finished 12:04:47 2021-07-09

Out[71]:

	id	campaign_id	coupon_id	redemption_status	customer_id	age_range	marital_status	re
0	1	13	27	0	1053	46-55	NaN	
1	2	13	116	0	48	36-45	Married	
2	6	9	635	0	205	46-55	Married	
3	7	13	644	0	0	NaN	NaN	
4	9	8	1017	0	1489	46-55	Married	

In [72]:
```
1 data.describe()
```
executed in 136ms, finished 12:04:48 2021-07-09

Out[72]:

	id	campaign_id	coupon_id	redemption_status	customer_id	ren
count	78369.000000	78369.000000	78369.000000	78369.000000	78369.000000	78369.000
mean	64347.975449	13.974441	566.363243	0.009302	394.739591	0.029
std	37126.440855	8.019215	329.966054	0.095999	507.905423	0.169
min	1.000000	1.000000	1.000000	0.000000	0.000000	0.000
25%	32260.000000	8.000000	280.000000	0.000000	0.000000	0.000
50%	64318.000000	13.000000	597.000000	0.000000	17.000000	0.000
75%	96577.000000	13.000000	857.000000	0.000000	775.000000	0.000
max	128595.000000	30.000000	1115.000000	1.000000	1581.000000	1.000

In [73]:
```
1 data.info()
```
executed in 74ms, finished 12:04:48 2021-07-09

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78369 entries, 0 to 78368
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 78369 non-null  int64 
 1   campaign_id        78369 non-null  int64 
 2   coupon_id          78369 non-null  int64 
 3   redemption_status  78369 non-null  int64 
 4   customer_id        78369 non-null  int64 
 5   age_range          39906 non-null  object
 6   marital_status     22023 non-null  object
 7   rented             78369 non-null  int64 
 8   family_size        78369 non-null  int64 
 9   no_of_children     10301 non-null  object
 10  income_bracket     78369 non-null  int64 
dtypes: int64(8), object(3)
memory usage: 6.6+ MB
```

In [74]:
```
1 df = data
```
executed in 25ms, finished 12:04:48 2021-07-09

2.2 Fill in NULL values

In [75]:
```
print(str(round(((((df['age_range'].isna().sum()))/len(df))*100),2))
+ '% Null in age_range')
print(str(round(((((df['marital_status'].isna().sum()))/len(df))*100),2)
+ '% Null in marital_status')
print(str(round(((((df['no_of_children'].isna().sum()))/len(df))*100),2)
+ '% Null in no_of_children')
```

executed in 58ms, finished 12:04:48 2021-07-09

```
49.08% Null in age_range
71.9% Null in marital_status
86.86% Null in no_of_children
```

Need to fill in NULL values in:

- age_range
- marital_status
- no_of_children

In [76]:
```
df_columns = df.columns
for i in df_columns:
    print('\033[1m' + i.upper() + '\033[0m')
    print(df[[i]].value_counts(ascending=False))
    print('_____________________________________________________')
```

executed in 254ms, finished 12:04:48 2021-07-09

```
REDEMPTION_STATUS
redemption_status
0                   77640
1                     729
dtype: int64
_____________________________________________________
CUSTOMER_ID
customer_id
0              38463
590              193
189              175
1192             173
1378             161
                ...
1256              13
1196              12
1222              11
371               11
180                8
Length: 648, dtype: int64
```

Distrubution of income bracket as a percentage.

In [77]:

```
5.0
#1
print('1:' + str(3279/39906))
#2
print('2:' + str(3737/39906))
#3
print('3:' + str(3486/39906))
#4
print('4:' + str(7983/39906))
#5
print('5:' + str(10205/39906))
#6
print('6:' + str(4465/39906))
#7
print('7:' + str(1713/39906))
#8
print('8:' + str(1783/39906))
#9
print('9:' + str(1818/39906))
#10
print('10:' + str(513/39906))
#11
print('11:' + str(234/39906))
#12
print('12:' + str(690/39906))

```

executed in 24ms, finished 12:04:48 2021-07-09

```
1:0.08216809502330477
2:0.09364506590487646
3:0.08735528491956097
4:0.2000451059990979
5:0.255725955996592
6:0.11188793665113016
7:0.04292587580814915
8:0.04467999799528893
9:0.04555705908885882
10:0.012855209742895805
11:0.005863779882724403
12:0.017290632987520673
```

In [78]:

```
df['income_bracket']=df['income_bracket'].replace(0, np.nan)
```

executed in 28ms, finished 12:04:48 2021-07-09

In [79]:

```
df['income_bracket'].isnull().sum()
```

executed in 16ms, finished 12:04:48 2021-07-09

Out[79]: 38463

In [80]:
```
df['income_bracket'] = df['income_bracket'].fillna(pd.Series(np.random.cl
    '1.0', '2.0', '3.0', '4.0', '5.0', '6.0', '7.0', '8.0',
    '9.0', '10.0', '11.0', '12.0'],
    p=[.08, .09, .09, .20, .26, .11, .04, .04, .05, .01, .01, .02],
    size= len(df))))
```

executed in 76ms, finished 12:04:48 2021-07-09

Use single number to Represent age catorgory.
Use single number to represent material status (1,2,3)
Use single number to represent number of children

AGE_RANGE
18-25 will be 1
26-35 will be 2
36-45 will be 3
46-55 will be 4
56-70 will be 5
70+ will be 6

In [81]:
```
df['age_range']=df['age_range'].replace({'18-25' : '1', '26-35' : '2',
            '36-45' : '3', '46-55' : '4', '56-70' : '5', '70+' : '6'})
df['age_range']
```

executed in 95ms, finished 12:04:48 2021-07-09

Out[81]:
```
0          4
1          3
2          4
3        NaN
4          4
        ...
78364      6
78365      6
78366    NaN
78367    NaN
78368    NaN
Name: age_range, Length: 78369, dtype: object
```

In [82]:
```
df[['age_range']].isnull().sum()
```

executed in 44ms, finished 12:04:48 2021-07-09

Out[82]:
```
age_range    38463
dtype: int64
```

Distribution of age as a percentage.

In [83]:
```
# 1
print(2398/ 39906)
# 2
print(7203/ 39906)
# 3
print(9534/ 39906)
# 4
print(14094/ 39906)
# 5
print(3583/ 39906)
# 6
print(3094/ 39906)

```
executed in 18ms, finished 12:04:48 2021-07-09

```
0.06009121435373127
0.1804991730566832
0.23891144188843783
0.3531799729364005
0.08978599709316895
0.0775322006715782
```

In [84]:
```
df['age_range'] = df['age_range'].fillna(pd.Series(np.random.choice(['1',
                '3', '4', '5', '6'], p=[.06, .18, .24, .35, .09, .08],
                    size= len(df))))
```
executed in 58ms, finished 12:04:48 2021-07-09

MARITAL_STATUS

Married will be 1

Single will be 0

Nan will be 2

In [85]:
```
df['marital_status']=df['marital_status'].replace({'Married': '1',
                                                   'Single': '0'})
df['marital_status']
```
executed in 49ms, finished 12:04:48 2021-07-09

Out[85]:
```
0        NaN
1          1
2          1
3        NaN
4          1
        ...
78364      1
78365      1
78366    NaN
78367    NaN
78368    NaN
Name: marital_status, Length: 78369, dtype: object
```

In [86]:
```
#Convert NaN to 2 for marital_status
df['marital_status'].fillna(2, inplace=True)
df['marital_status']
```

executed in 39ms, finished 12:04:49 2021-07-09

Out[86]:
```
0        2
1        1
2        1
3        2
4        1
        ..
78364    1
78365    1
78366    2
78367    2
78368    2
Name: marital_status, Length: 78369, dtype: object
```

In [87]:
```
df['age_range'] = df['age_range'].astype(str).astype(int)
```

executed in 73ms, finished 12:04:49 2021-07-09

In [88]:
```
df['marital_status'] = df['marital_status'].astype(str).astype(int)
```

executed in 82ms, finished 12:04:49 2021-07-09

In [89]:
```
df['income_bracket'] = df['income_bracket'].astype(str).astype(float)
```

executed in 138ms, finished 12:04:49 2021-07-09

In [90]:
```
df.info()
```

executed in 45ms, finished 12:04:49 2021-07-09

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78369 entries, 0 to 78368
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 78369 non-null  int64  
 1   campaign_id        78369 non-null  int64  
 2   coupon_id          78369 non-null  int64  
 3   redemption_status  78369 non-null  int64  
 4   customer_id        78369 non-null  int64  
 5   age_range          78369 non-null  int32  
 6   marital_status     78369 non-null  int32  
 7   rented             78369 non-null  int64  
 8   family_size        78369 non-null  int64  
 9   no_of_children     10301 non-null  object 
 10  income_bracket     78369 non-null  float64
dtypes: float64(1), int32(2), int64(7), object(1)
memory usage: 6.0+ MB
```

In [ ]:
```
1
```

2.3 Drop Values

In [91]:
```
1 df.columns
```
executed in 16ms, finished 12:04:49 2021-07-09

Out[91]:
```
Index(['id', 'campaign_id', 'coupon_id', 'redemption_status', 'customer_i
d',
       'age_range', 'marital_status', 'rented', 'family_size',
       'no_of_children', 'income_bracket'],
      dtype='object')
```

Id is an unique value that does not contribute value to this information.

In [92]:
```
1 df = df.drop('id', axis = 1)
```
executed in 24ms, finished 12:04:49 2021-07-09

campaign_id is useful information but without detailed information about what when in to the campaign it is difficult to use it as a classifier. Making it worth dropping.

In [93]:
```
1 df = df.drop('campaign_id', axis = 1)
```
executed in 16ms, finished 12:04:49 2021-07-09

Coupons can be both unique and not. The given data does not give an indicator of which are and which are not. They also do have a way of grouping by type, such as household products, animal items, ect.

In [94]:
```
1 df = df.drop('coupon_id', axis = 1)
```
executed in 17ms, finished 12:04:49 2021-07-09

Customer_id is getting drop since it does not give enough information.

In [95]:
```
1 df = df.drop('customer_id', axis = 1)
```
executed in 18ms, finished 12:04:49 2021-07-09

Will drop no_of_children since high amount of null values and can not predict values from other columns.

In [96]:
```
1 df = df.drop('no_of_children', axis = 1)
```
executed in 33ms, finished 12:04:49 2021-07-09

In [97]:
```
1 df.info()
```

executed in 37ms, finished 12:04:49 2021-07-09

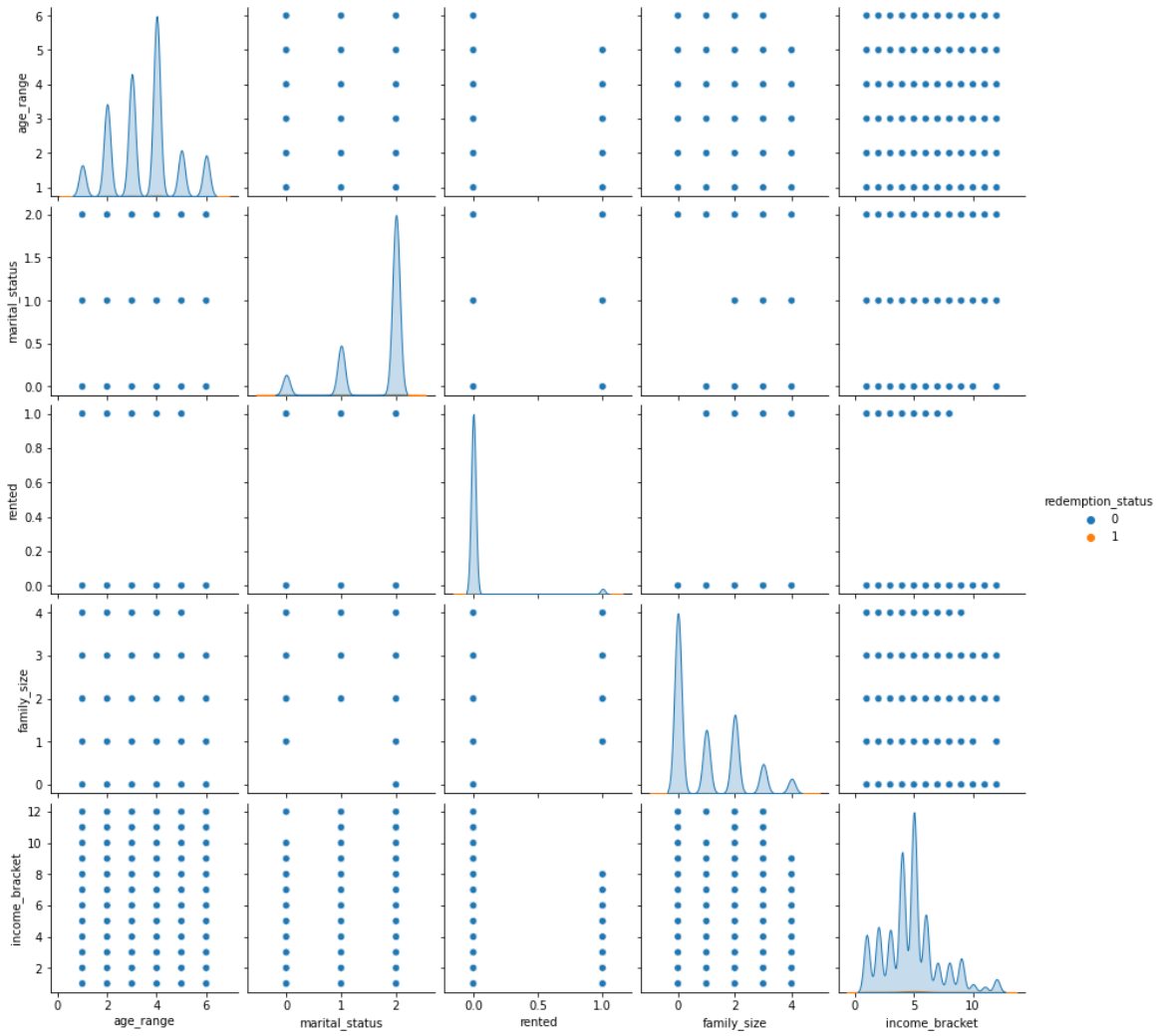
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78369 entries, 0 to 78368
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   redemption_status  78369 non-null  int64  
 1   age_range          78369 non-null  int32  
 2   marital_status     78369 non-null  int32  
 3   rented             78369 non-null  int64  
 4   family_size        78369 non-null  int64  
 5   income_bracket     78369 non-null  float64
dtypes: float64(1), int32(2), int64(3)
memory usage: 3.0 MB
```

3 Data Exploration

In [98]:
```
sns.pairplot(df, hue = 'redemption_status')

```
executed in 3m 36s, finished 12:08:25 2021-07-09

Out[98]: <seaborn.axisgrid.PairGrid at 0x133f05fde80>



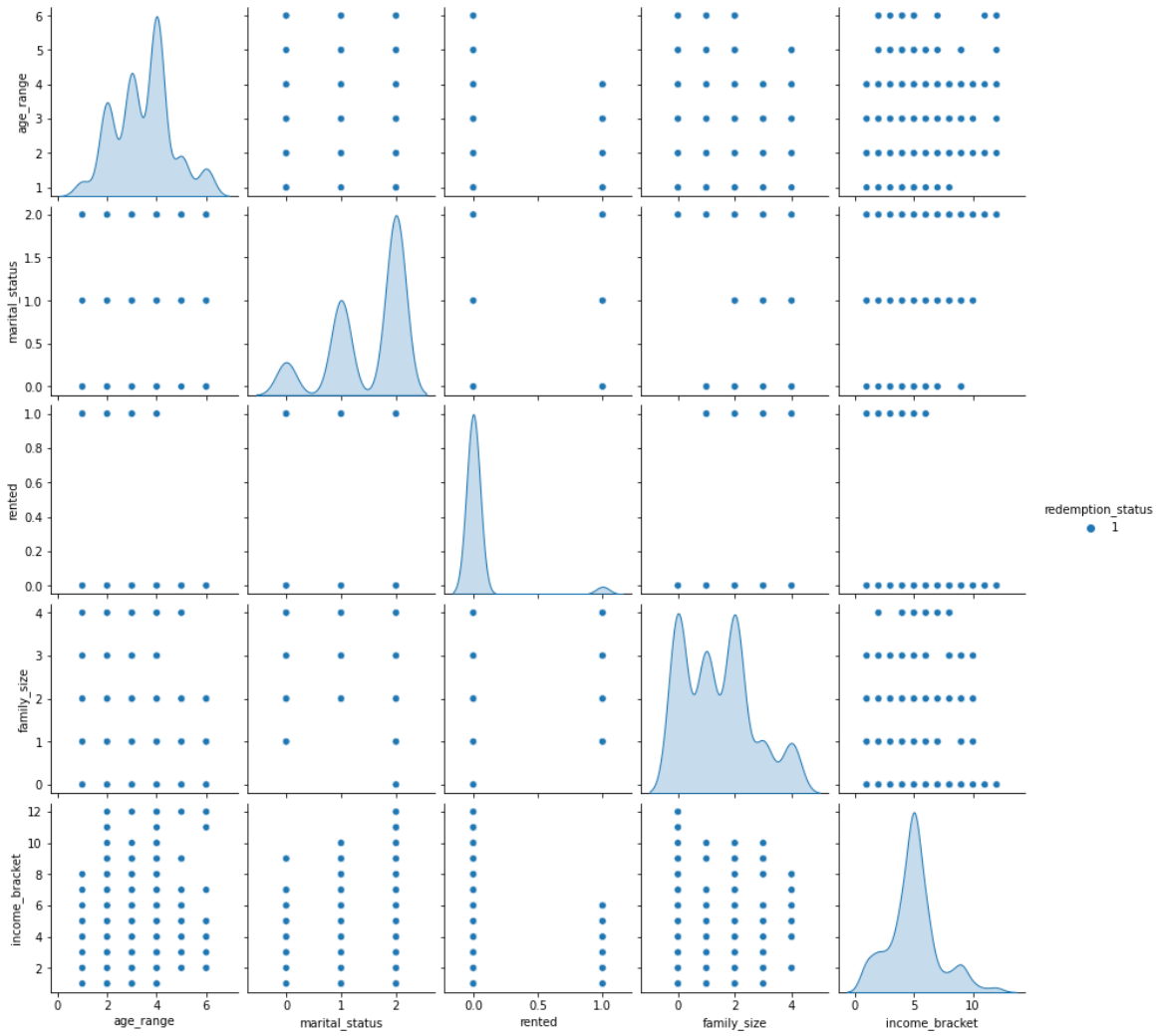
In [99]:
```
df_redeem = df.loc[df['redemption_status'] == 1]
```
executed in 15ms, finished 12:08:25 2021-07-09

In [100]:
```
df_not_redeem = df.loc[df['redemption_status'] == 0]
```
executed in 31ms, finished 12:08:25 2021-07-09

In [101]:
```
1 sns.pairplot(df_redeem, hue = 'redemption_status')
2
```

executed in 14.2s, finished 12:08:39 2021-07-09

Out[101]: <seaborn.axisgrid.PairGrid at 0x133f0643520>



In [102]:
```
1 df_redeem.describe()
2
```
executed in 67ms, finished 12:08:39 2021-07-09

Out[102]:

	redemption_status	age_range	marital_status	rented	family_size	income_bracket
count	729.0	729.000000	729.000000	729.000000	729.000000	729.000000
mean	1.0	3.458162	1.477366	0.039781	1.418381	4.968450
std	0.0	1.154883	0.681261	0.195577	1.232233	2.103854
min	1.0	1.000000	0.000000	0.000000	0.000000	1.000000
25%	1.0	3.000000	1.000000	0.000000	0.000000	4.000000
50%	1.0	4.000000	2.000000	0.000000	1.000000	5.000000
75%	1.0	4.000000	2.000000	0.000000	2.000000	6.000000
max	1.0	6.000000	2.000000	1.000000	4.000000	12.000000

In [103]:
```
1 df_not_redeem.describe()
```
executed in 100ms, finished 12:08:39 2021-07-09

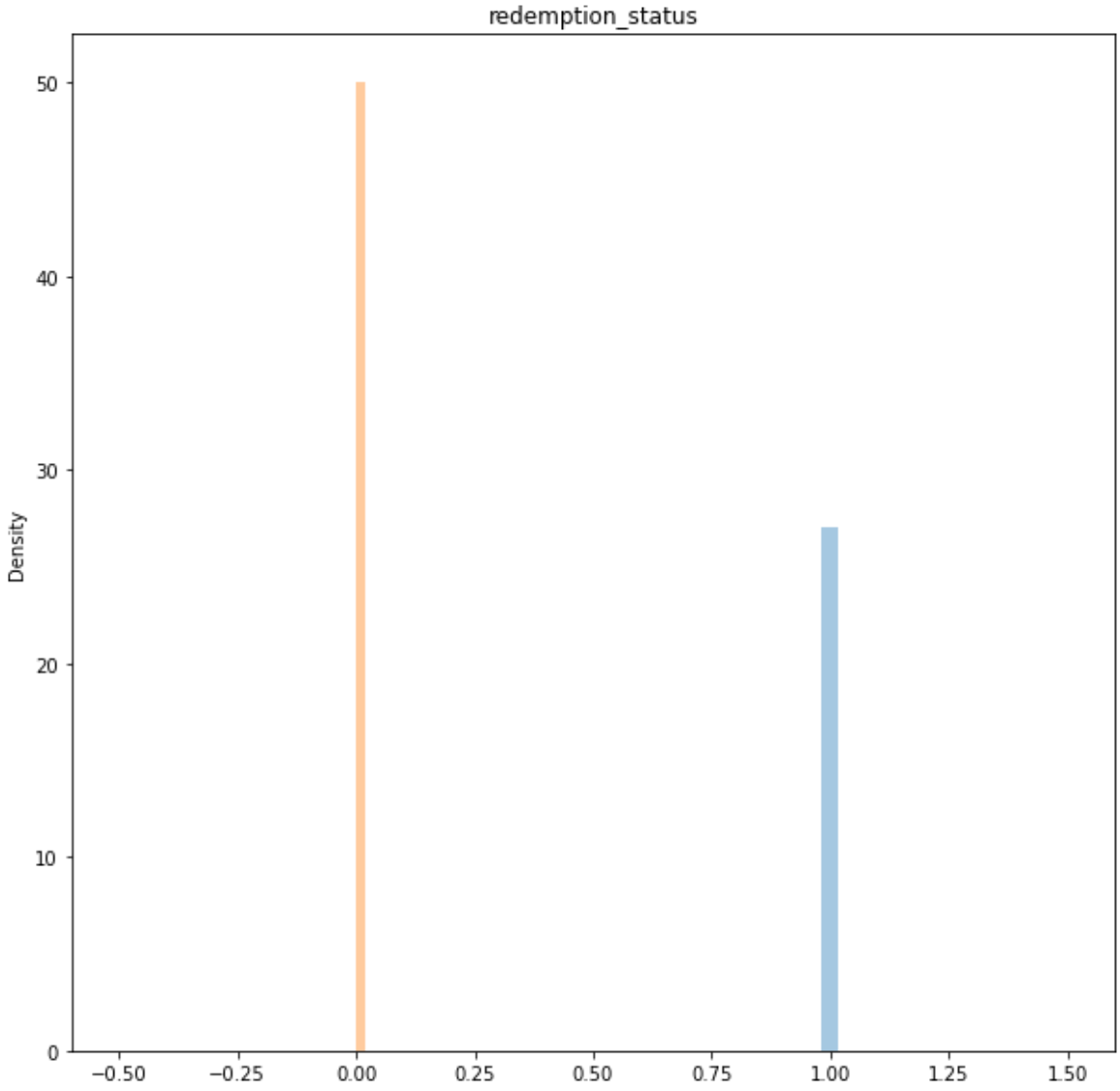
Out[103]:

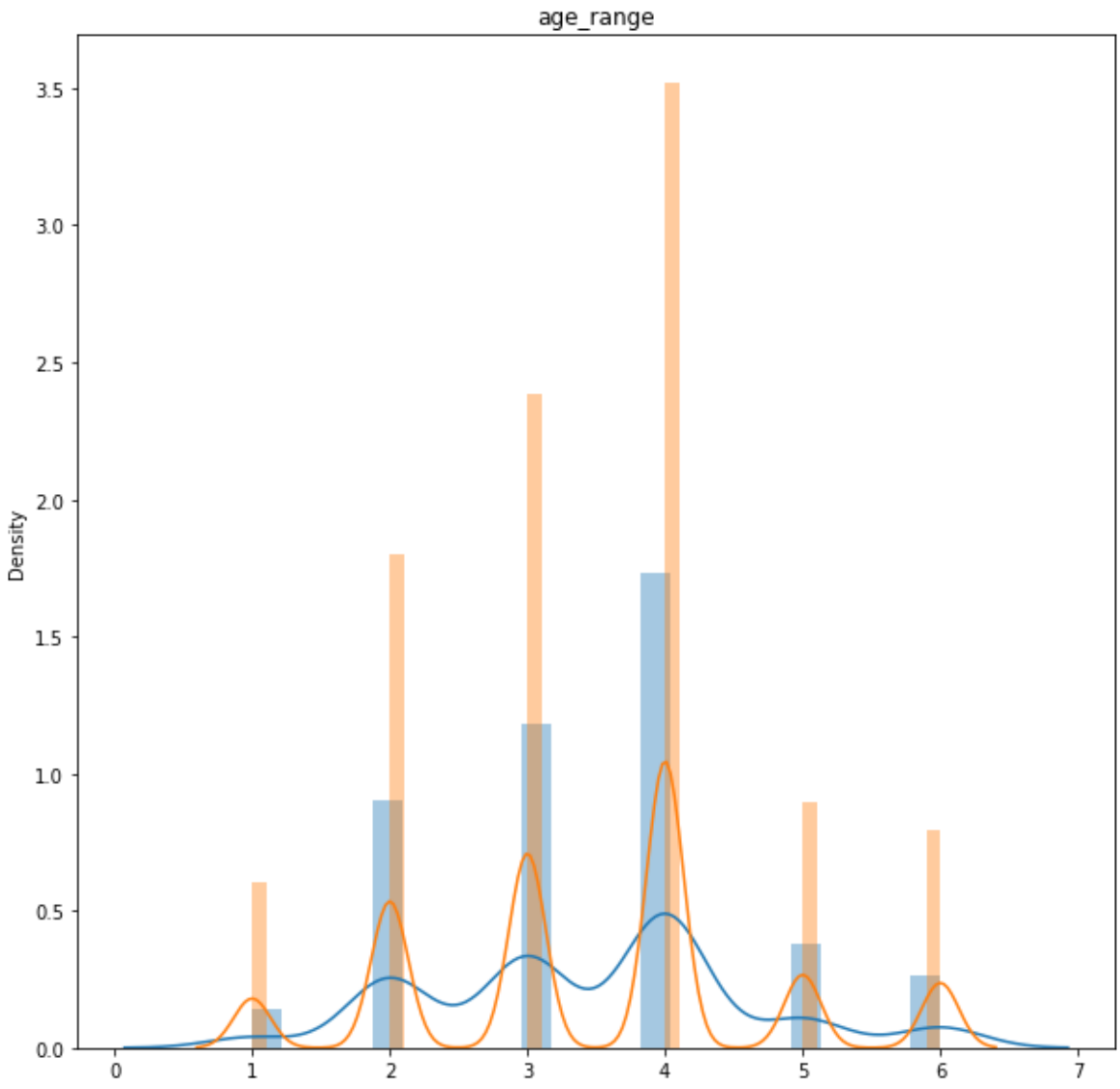
	redemption_status	age_range	marital_status	rented	family_size	income_bı
count	77640.0	77640.000000	77640.000000	77640.000000	77640.000000	77640.0
mean	0.0	3.469178	1.638782	0.029547	1.000914	4.7
std	0.0	1.276163	0.627436	0.169334	1.176351	2.3
min	0.0	1.000000	0.000000	0.000000	0.000000	1.0
25%	0.0	3.000000	1.000000	0.000000	0.000000	3.0
50%	0.0	4.000000	2.000000	0.000000	1.000000	5.0
75%	0.0	4.000000	2.000000	0.000000	2.000000	6.0
max	0.0	6.000000	2.000000	1.000000	4.000000	12.0

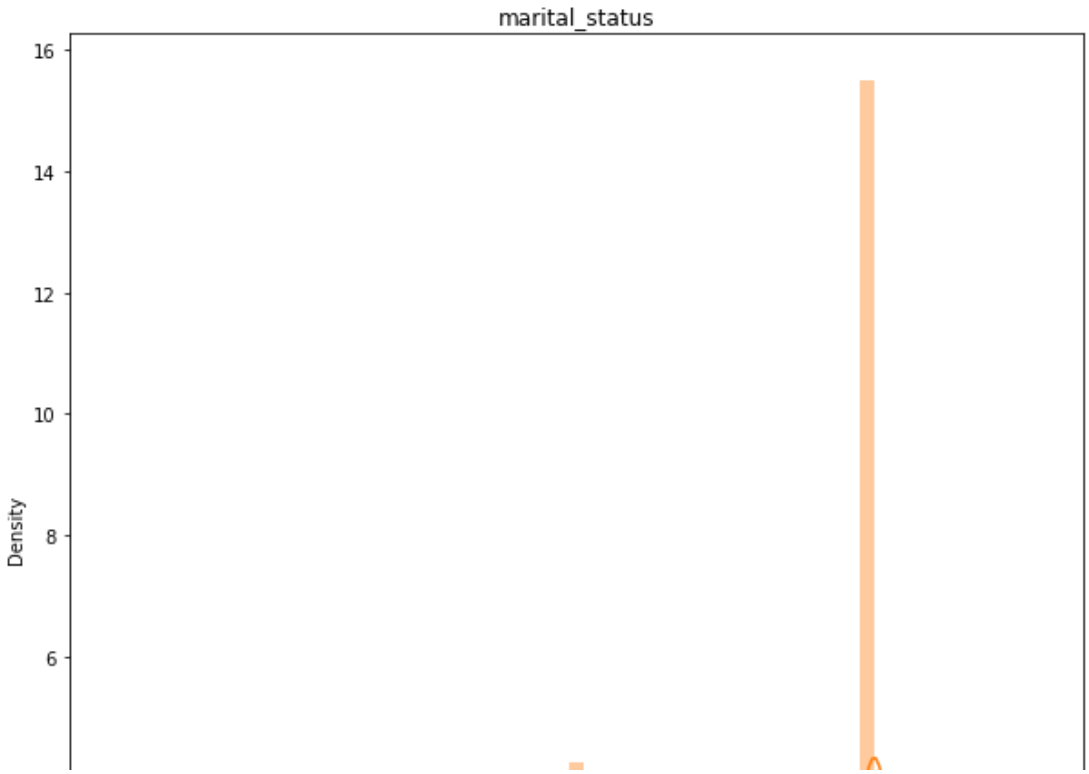
In [104]:

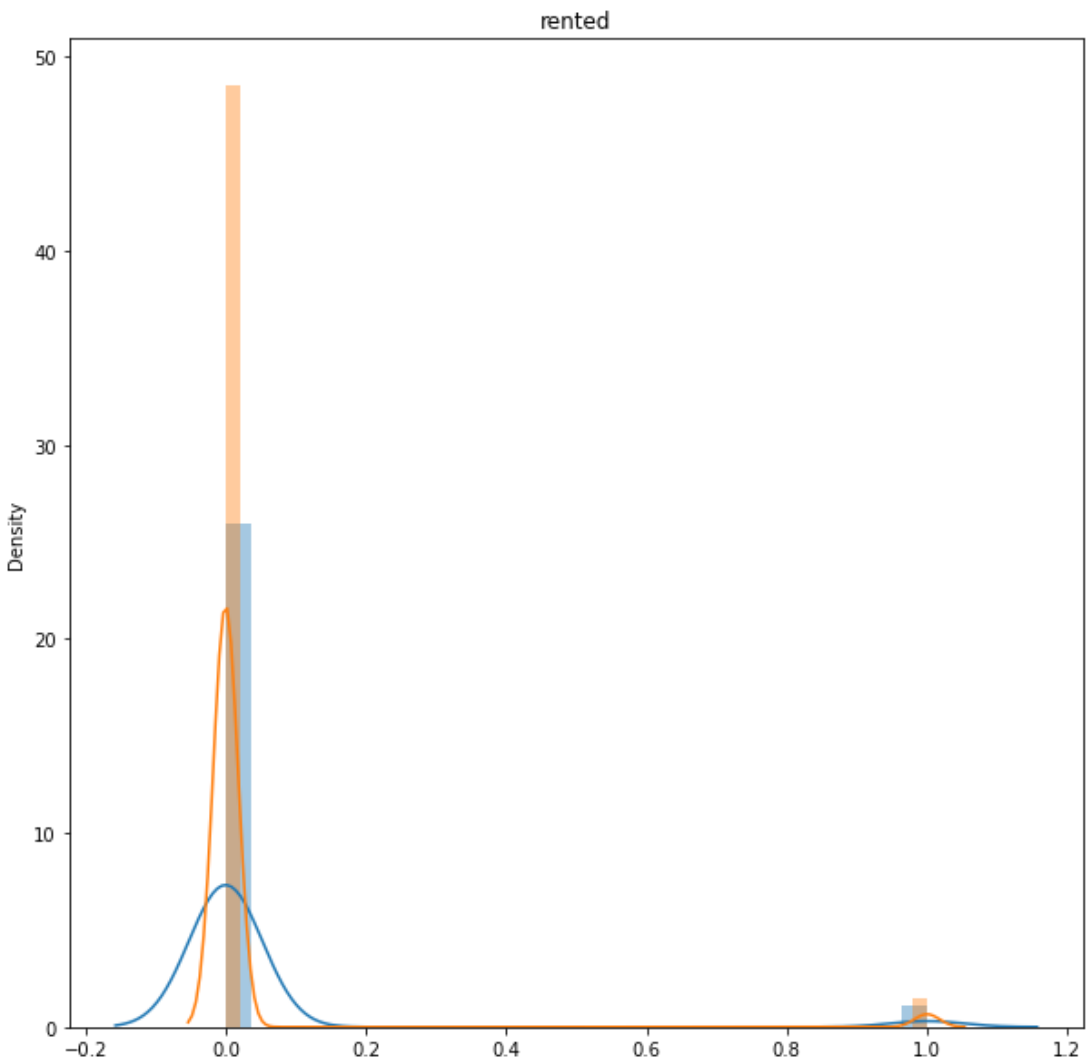
```
df_columns = df.columns
for i in df_columns:
    figure, ax= plt.subplots(figsize = (10,10))
    sns.distplot(df_redeem, x = df_redeem[i])
    sns.distplot(df_not_redeem, x = df_not_redeem[i])
    plt.title(i)
```

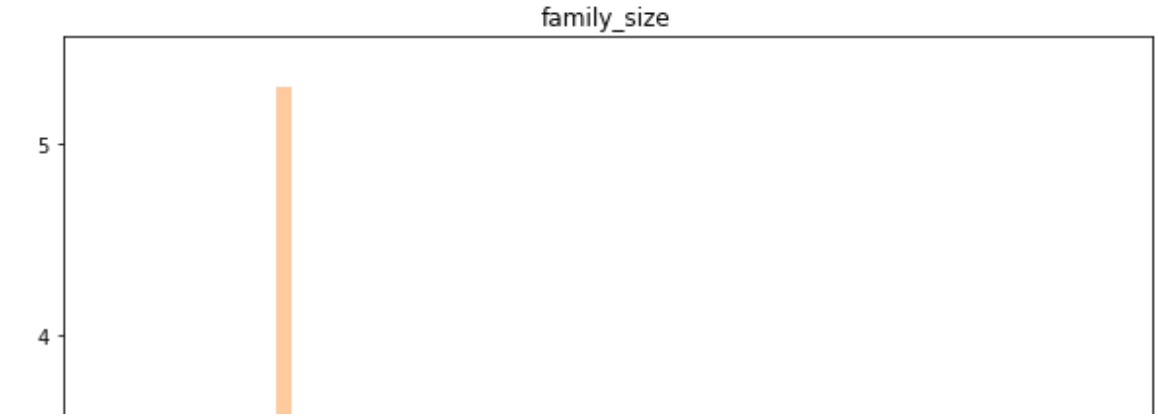
executed in 8.78s, finished 12:08:48 2021-07-09

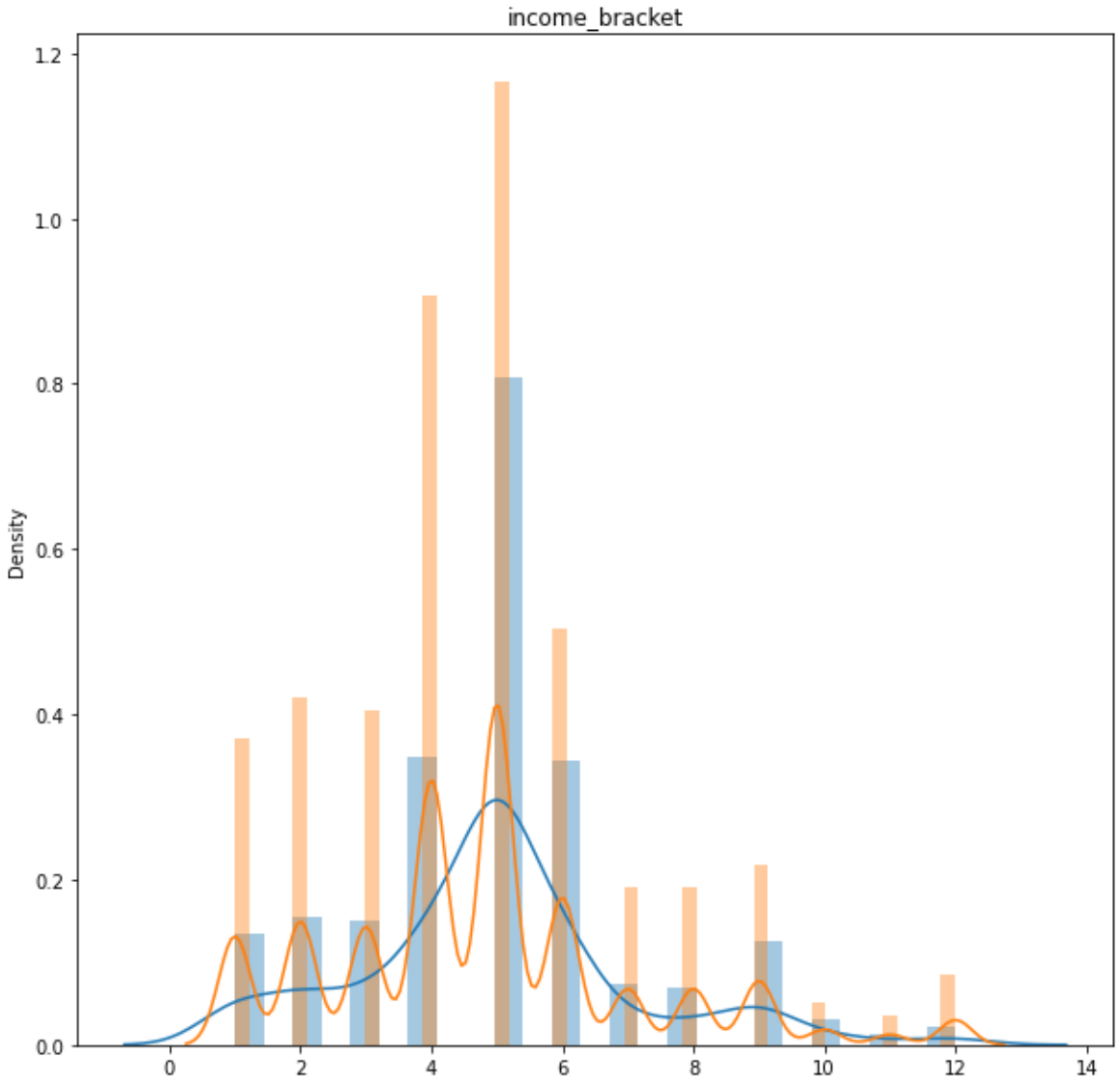



age_range
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
1
2
3
4
5
6
7


marital_status
Density
16
14
12
10
8
6


rented
Density
50
40
30
20
10
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2





AGE_RANGE

18-25 is 1

26-35 is 2

36-45 id 3

46-55 is 4

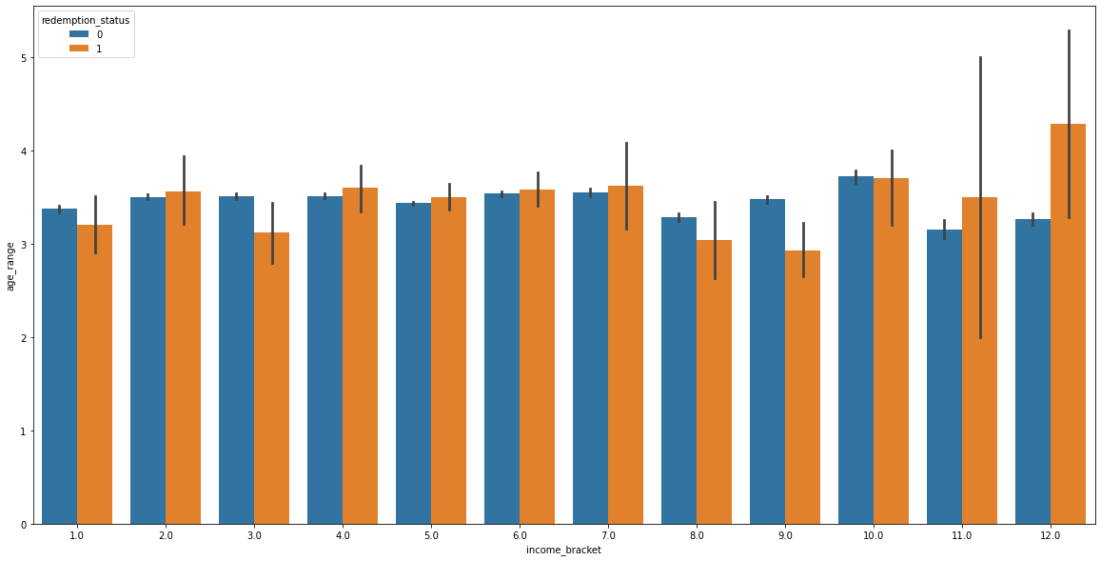
56-70 is 5

70+ os 6

In [105]:
```
figure, ax= plt.subplots(figsize = (20,10))
sns.barplot( x = 'income_bracket', y ='age_range' , hue = 'redemption_sta
                palette="tab10",  data = df)
```

executed in 4.63s, finished 12:08:53 2021-07-09

Out[105]: <AxesSubplot:xlabel='income_bracket', ylabel='age_range'>

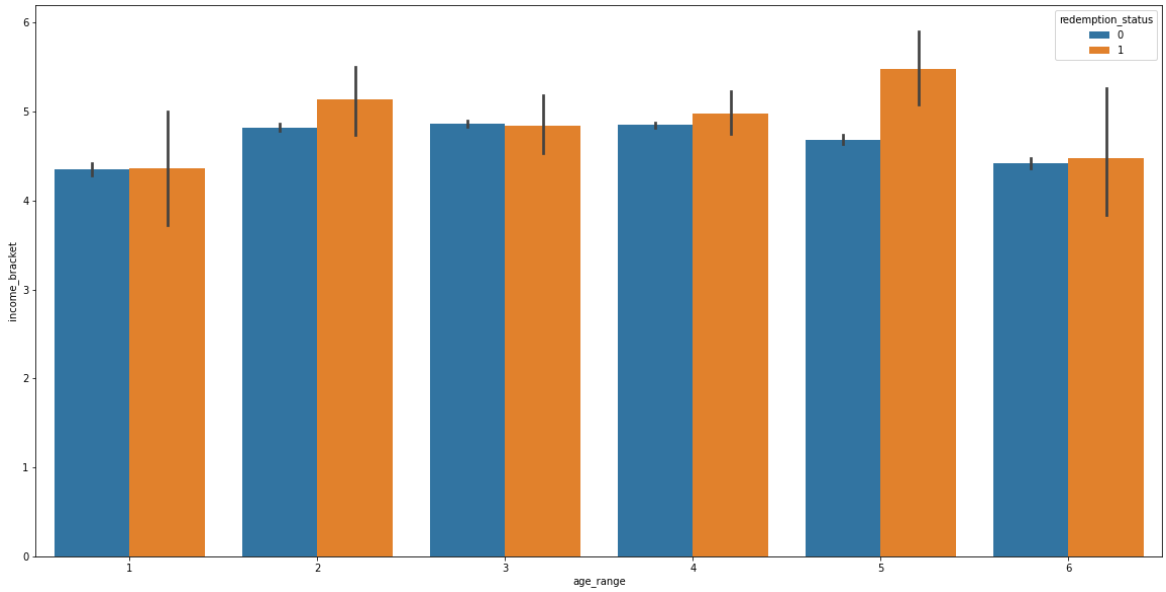


In income bracket 12 older people are more likely to use coupon

In [106]:
```
figure, ax= plt.subplots(figsize = (20,10))
sns.barplot( y = 'income_bracket', x ='age_range' , hue = 'redemption_sta
                palette="tab10",  data = df)
```

executed in 3.29s, finished 12:08:56 2021-07-09

Out[106]: <AxesSubplot:xlabel='age_range', ylabel='income_bracket'>



In [107]:
```
df['income_bracket'].value_counts(ascending=False)#(unique)
```
executed in 21ms, finished 12:08:56 2021-07-09

Out[107]:
```
5.0     20165
4.0     15590
6.0      8701
2.0      7250
3.0      6950
1.0      6379
9.0      3772
8.0      3300
7.0      3292
12.0     1458
10.0      901
11.0      611
Name: income_bracket, dtype: int64
```

In [108]:
```
df.columns
```
executed in 29ms, finished 12:08:56 2021-07-09

Out[108]:
```
Index(['redemption_status', 'age_range', 'marital_status', 'rented',
       'family_size', 'income_bracket'],
      dtype='object')
```

4 Prediction Modeling

In [109]:
```
df.columns
```
executed in 10ms, finished 12:08:56 2021-07-09

Out[109]:
```
Index(['redemption_status', 'age_range', 'marital_status', 'rented',
       'family_size', 'income_bracket'],
      dtype='object')
```

4.0.1 Changing Categorical Variables to Dummies

In [110]:
```
marital_status_dummies = pd.get_dummies(df['marital_status'],
                                        prefix = 'marital_status', drop_first = 
rented_dummies = pd.get_dummies(df['rented'], prefix = 'rented',
                                drop_first = True)
```
executed in 28ms, finished 12:08:56 2021-07-09

In [111]:
```
df_dummies=pd.concat([df.redemption_status, df['age_range'],
                      marital_status_dummies, rented_dummies,
                      df['family_size'], df['income_bracket']], axis = 1
```
executed in 26ms, finished 12:08:56 2021-07-09

4.1 Decision Tree

4.1.1 Decision Tree - Base

In [ ]:
```
1
```

In [112]:
```
1 #Split data
2 y = df_dummies['redemption_status']
3 X = df_dummies.drop('redemption_status', axis = 1)
4
5 X_train, X_test, y_train, y_test= train_test_split(X, y, train_size = .7
6                                                    test_size=.25, random_state=
```

executed in 57ms, finished 12:08:56 2021-07-09

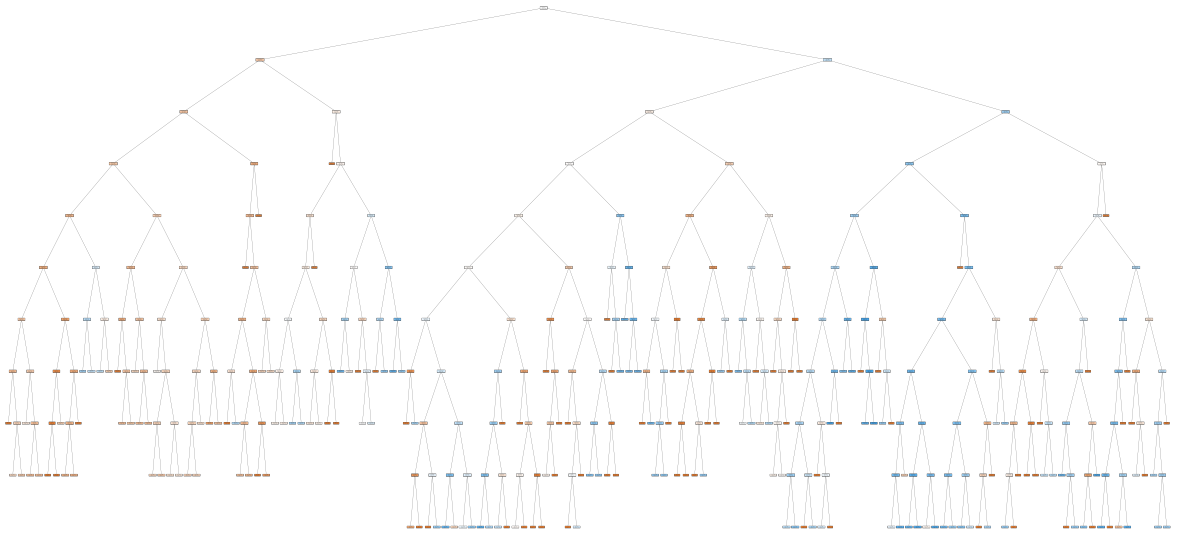
In [113]:
```
1 dt_base = DecisionTreeClassifier(criterion='gini', max_depth = 10,
2                                  class_weight = 'balanced', random_state
3 dt_base.fit(X_train, y_train)
```

executed in 156ms, finished 12:08:57 2021-07-09

Out[113]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=10, random_state=
10)
```

In [114]:
```
1 plt.figure(figsize=(150, 75))
2 plot_tree(dt_base, filled=True, rounded= True, feature_names=X_train.colu
```

executed in 1m 3.29s, finished 12:10:00 2021-07-09

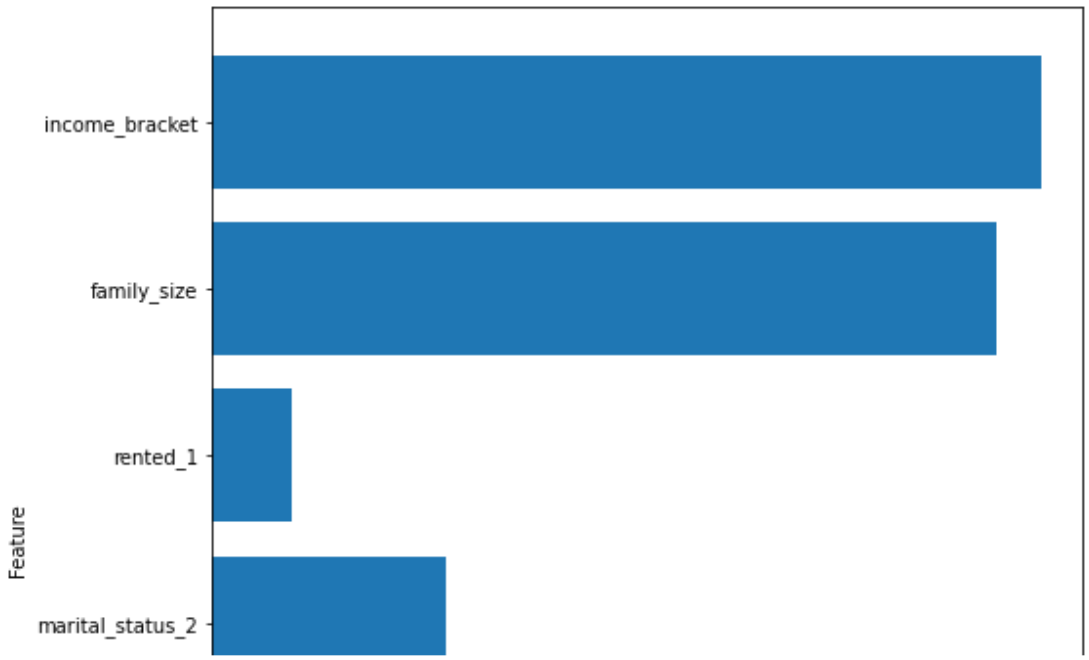


In [115]:
```
1 dt_base.feature_importances_
```
executed in 36ms, finished 12:10:00 2021-07-09

Out[115]:
```
array([0.23474387, 0.0246542 , 0.09028217, 0.03052314, 0.30119195,
       0.31860467])
```

In [116]:
```
1 def plot_feature_importances(model):
2     n_features = X_train.shape[1]
3     plt.figure(figsize=(8,10))
4     plt.barh(range(n_features), model.feature_importances_, align='center
5     plt.yticks(np.arange(n_features), X_train.columns.values,
6                horizontalalignment='right')
7     plt.xlabel('Feature Importance')
8     plt.ylabel('Feature')
```
executed in 19ms, finished 12:10:00 2021-07-09

In [117]:
```
1 plot_feature_importances(dt_base)
```
executed in 408ms, finished 12:10:00 2021-07-09



The most important factor for determining weather or not some will use a coupon, in order, are income_bracket, family_size, age_range.

In [118]:
```
# Test set predictions
pred = dt_base.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 142ms, finished 12:10:00 2021-07-09

```
[[13572  5833]
 [   84   104]]
              precision    recall  f1-score   support

           0       0.99      0.70      0.82     19405
           1       0.02      0.55      0.03       188

    accuracy                           0.70     19593
   macro avg       0.51      0.63      0.43     19593
weighted avg       0.98      0.70      0.81     19593
```

In [119]:
```
dt_cv_score1 = cross_val_score(dt_base, X_train, y_train, cv=10,
                               scoring= 'roc_auc')
mean_dt_cv_score1 = np.mean(dt_cv_score1)

dt_cv_score2 = cross_val_score(dt_base, X_train, y_train, cv=10,
                               scoring= 'f1')
mean_dt_cv_score2 = np.mean(dt_cv_score2)

dt_cv_score3 = cross_val_score(dt_base, X_train, y_train, cv=10,
                               scoring= 'recall')
mean_dt_cv_score3 = np.mean(dt_cv_score3)

dt_cv_score4 = cross_val_score(dt_base, X_train, y_train, cv=10)
mean_dt_cv_score4 = np.mean(dt_cv_score4)

print(f"Mean Cross Validation Score: {mean_dt_cv_score4 : .2%}")
print(f"Mean Cross Validation Score(roc_auc): {mean_dt_cv_score1 : .2%}")
print(f"Mean Cross Validation Score(f1): {mean_dt_cv_score2 : .2%}")
print(f"Mean Cross Validation Score(recall): {mean_dt_cv_score3 : .2%}")
```

executed in 5.64s, finished 12:10:06 2021-07-09

```
Mean Cross Validation Score:  71.43%
Mean Cross Validation Score(roc_auc):  69.36%
Mean Cross Validation Score(f1):  3.64%
Mean Cross Validation Score(recall):  58.42%
```

4.1.2 Decision Tree - GridSearch

In [120]:
```
r=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
```

executed in 24ms, finished 12:10:06 2021-07-09

In [121]:
```
dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
                  15, 16, 17, 18, 19, 20],
    'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                          13, 14, 15, 16, 17, 18, 19, 20],
    'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                         13, 14, 15, 16, 17, 18, 19, 20],
    'class_weight': ['balanced']
}
```

executed in 12ms, finished 12:10:06 2021-07-09

In [122]:
```
num_decision_trees = 3 * 2 * 4 * 2 * 2 *1
print(f"Grid Search will have to search through \
{num_decision_trees} different permutations.")

```

executed in 14ms, finished 12:10:06 2021-07-09

```
Grid Search will have to search through 96 different permutations.
```

In [123]:
```
# Instantiate GridSearchCV
dt_grid_search = GridSearchCV(dt_base, dt_param_grid, cv=3, return_train_

# Fit to the data
dt_grid_search.fit(X_train, y_train)
```

executed in 57m 27s, finished 13:07:33 2021-07-09

Out[123]:
```
GridSearchCV(cv=3,
             estimator=DecisionTreeClassifier(class_weight='balanced',
                                              max_depth=10, random_state=1
0),
             param_grid={'class_weight': ['balanced'],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1
2,
                                       13, 14, 15, 16, 17, 18, 19, 20],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 1
0, 11,
                                              12, 13, 14, 15, 16, 17, 18, 1
9,
                                              20],
                         'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 1
0,
                                               11, 12, 13, 14, 15, 16, 17,
18,
                                               19, 20]},
             return_train_score=True)
```

In [124]:
```
# Mean training score
dt_gs_training_score = np.mean(dt_grid_search.cv_results_['mean_train_sc

# Mean test score
dt_gs_testing_score = dt_grid_search.score(X_test, y_test)

print(f"Mean Training Score: {dt_gs_training_score :.2%}")
print(f"Mean Test Score: {dt_gs_testing_score :.2%}")
print("Best Parameter Combination Found During Grid Search:")
dt_grid_search.best_params_
```

executed in 16ms, finished 13:07:33 2021-07-09

```
Mean Training Score: nan%
Mean Test Score: 74.62%
Best Parameter Combination Found During Grid Search:
```

Out[124]:
```
{'class_weight': 'balanced',
 'criterion': 'entropy',
 'max_depth': 13,
 'min_samples_leaf': 10,
 'min_samples_split': 2}
```

In [125]:
```
dt_grid_search = DecisionTreeClassifier(
 class_weight= 'balanced',
 criterion= 'entropy',
 max_depth= 13,
 min_samples_leaf = 10,
 min_samples_split = 2)
dt_grid_search.fit(X_train, y_train)
```

executed in 44ms, finished 13:07:33 2021-07-09

Out[125]:
```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=13, min_samples_leaf=10)
```

In [126]:
```
# Test set predictions
pred = dt_grid_search.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

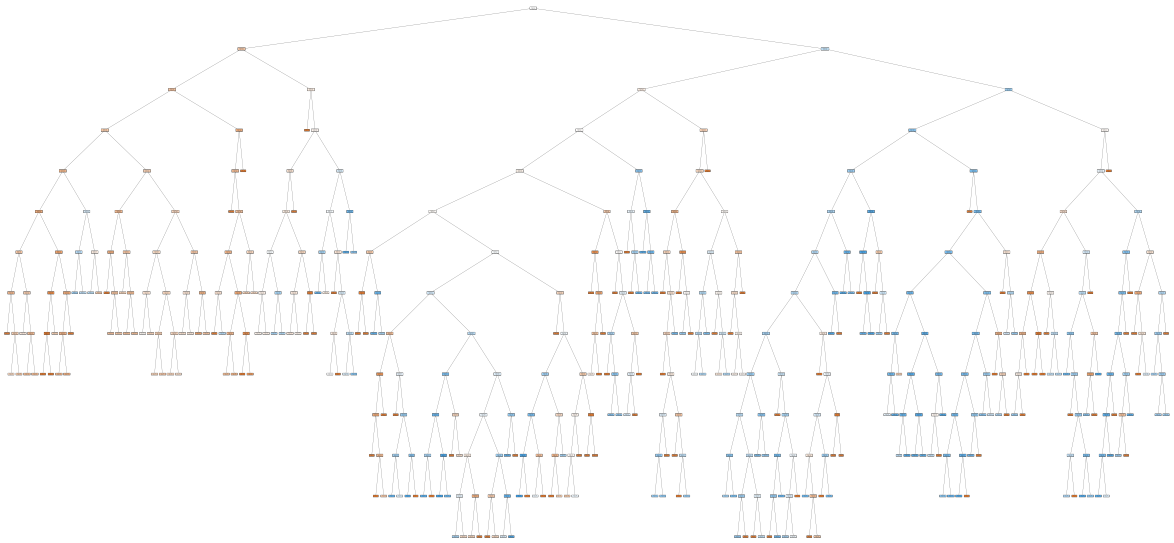
executed in 47ms, finished 13:07:33 2021-07-09

```
[[14521  4884]
 [   89    99]]
              precision    recall  f1-score   support

           0       0.99      0.75      0.85     19405
           1       0.02      0.53      0.04       188

    accuracy                           0.75     19593
   macro avg       0.51      0.64      0.45     19593
weighted avg       0.98      0.75      0.85     19593
```

In [127]:
```
1 plt.figure(figsize=(150, 75))
2 plot_tree(dt_grid_search, filled=True, rounded= True, feature_names=X_tra
```
executed in 21.2s, finished 13:07:54 2021-07-09



In [128]:
```
1 dt_grid_search.feature_importances_
```
executed in 13ms, finished 13:07:54 2021-07-09

Out[128]:
```
array([0.26379537, 0.0253312 , 0.05749168, 0.04574931, 0.27555464,
       0.3320778 ])
```

In [129]:
```
1 plot_feature_importances(dt_base)
```
executed in 142ms, finished 13:07:54 2021-07-09

The most important factor for determining weather or not some will use a coupon, in order, are

income_bracket, family_size, age_range.

In [130]:

```
dt_cv_score11 = cross_val_score(dt_grid_search, X_train, y_train,
                                cv=10, scoring= 'roc_auc')
mean_dt_cv_score11 = np.mean(dt_cv_score11)

dt_cv_score22 = cross_val_score(dt_grid_search, X_train, y_train,
                                cv=10, scoring= 'f1')
mean_dt_cv_score22 = np.mean(dt_cv_score22)

dt_cv_score33 = cross_val_score(dt_grid_search, X_train, y_train,
                                cv=10, scoring= 'recall')
mean_dt_cv_score33 = np.mean(dt_cv_score33)

dt_cv_score44 = cross_val_score(dt_grid_search, X_train, y_train, cv=10)
mean_dt_cv_score44 = np.mean(dt_cv_score44)

print(f"Mean Cross Validation Score: {mean_dt_cv_score44 : .2%}")
print(f"Mean Cross Validation Score(roc_auc): {mean_dt_cv_score11 : .2%}"
print(f"Mean Cross Validation Score(f1): {mean_dt_cv_score22 : .2%}")
print(f"Mean Cross Validation Score(recall): {mean_dt_cv_score33 : .2%}"
```

executed in 1.80s, finished 13:07:56 2021-07-09

```
Mean Cross Validation Score:  73.79%
Mean Cross Validation Score(roc_auc):  70.67%
Mean Cross Validation Score(f1):  4.06%
Mean Cross Validation Score(recall):  60.09%
```

In [131]:

```
A=mean_dt_cv_score11-mean_dt_cv_score1
B=mean_dt_cv_score22-mean_dt_cv_score2
C=mean_dt_cv_score33-mean_dt_cv_score3
D=mean_dt_cv_score44-mean_dt_cv_score4

print(f"Mean Cross Validation Score compared to base model: {D : .2%}")
print(f"Mean Cross Validation Score(roc_auc) compared to \
base model: {A : .2%}")
print(f"Mean Cross Validation Score(f1) compared to base model: {B : .2%}
print(f"Mean Cross Validation Score(recall) compared to\
base model: {C : .2%}")
```

executed in 13ms, finished 13:07:56 2021-07-09

```
Mean Cross Validation Score compared to base model:  2.36%
Mean Cross Validation Score(roc_auc) compared to base model:  1.31%
Mean Cross Validation Score(f1) compared to base model:  0.42%
Mean Cross Validation Score(recall) compared to base model:  1.67%
```

In [ ]:

```
1
```

4.1.3 Decision Tree - Cross Validation/ Pruning

In [132]:

```
path = dt_grid_search.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
ccp_alphas = ccp_alphas[:-1]

dts = []

for ccp_alpha in ccp_alphas:
    dt_prun = DecisionTreeClassifier(
         class_weight= 'balanced',
         criterion= 'entropy',
         max_depth= 13,
         min_samples_leaf = 10,
         min_samples_split = 2, ccp_alpha=ccp_alpha)
    dt_prun.fit(X_train, y_train)
    dts.append(dt_prun)
```

executed in 5.52s, finished 13:08:02 2021-07-09

In [133]:

```
# Test set predictions
pred = dt_prun.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

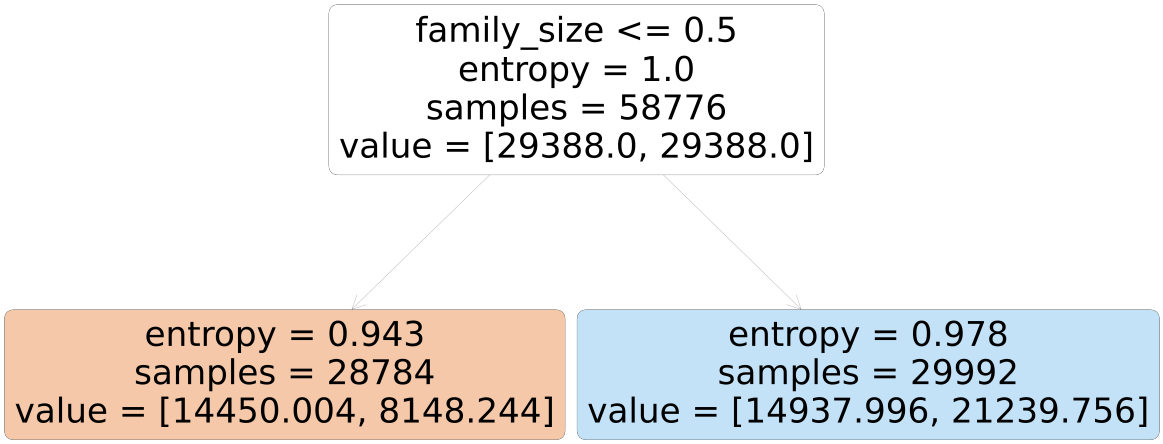
executed in 61ms, finished 13:08:02 2021-07-09

```
[[9610 9795]
 [  69  119]]
              precision    recall  f1-score   support

           0       0.99      0.50      0.66     19405
           1       0.01      0.63      0.02       188

    accuracy                           0.50     19593
   macro avg       0.50      0.56      0.34     19593
weighted avg       0.98      0.50      0.65     19593
```

In [134]:
```
1 plt.figure(figsize=(150, 75))
2 plot_tree(dt_prun, filled=True, rounded= True, feature_names=X_train.colu
```
executed in 1.97s, finished 13:08:04 2021-07-09



In [135]:
```
1 dt_prun.feature_importances_
```
executed in 16ms, finished 13:08:04 2021-07-09

Out[135]:
```
array([0., 0., 0., 0., 1., 0.])
```

In [136]:
```
1 plot_feature_importances(dt_prun)
```
executed in 173ms, finished 13:08:04 2021-07-09

The most important factor for determining weather or not some will use a coupon, in order, are income_bracket, family_size, age_range.

In [137]:

```
dt_cv_score12 = cross_val_score(dt_prun, X_train, y_train, cv=10,
                                scoring= 'roc_auc')
mean_dt_cv_score12 = np.mean(dt_cv_score12)

dt_cv_score21 = cross_val_score(dt_prun, X_train, y_train, cv=10,
                                scoring= 'f1')
mean_dt_cv_score21 = np.mean(dt_cv_score21)

dt_cv_score34 = cross_val_score(dt_prun, X_train, y_train, cv=10,
                                scoring= 'recall')
mean_dt_cv_score34 = np.mean(dt_cv_score34)

dt_cv_score43 = cross_val_score(dt_prun, X_train, y_train, cv=10)
mean_dt_cv_score43 = np.mean(dt_cv_score43)

print(f"Mean Cross Validation Score: {mean_dt_cv_score43 : .2%}")
print(f"Mean Cross Validation Score(roc_auc): {mean_dt_cv_score12 : .2%}"
print(f"Mean Cross Validation Score(f1): {mean_dt_cv_score21 : .2%}")
print(f"Mean Cross Validation Score(recall): {mean_dt_cv_score34 : .2%}"
```

executed in 1.78s, finished 13:08:06 2021-07-09

```
Mean Cross Validation Score:  61.08%
Mean Cross Validation Score(roc_auc):  61.79%
Mean Cross Validation Score(f1):  2.81%
Mean Cross Validation Score(recall):  59.13%
```

In [138]:

```
A=mean_dt_cv_score12-mean_dt_cv_score1
B=mean_dt_cv_score21-mean_dt_cv_score2
C=mean_dt_cv_score34-mean_dt_cv_score3
D=mean_dt_cv_score43-mean_dt_cv_score4

print(f"Mean Cross Validation Score compared to base model: {D : .2%}")
print(f"Mean Cross Validation Score(roc_auc) compared to base \
model: {A : .2%}")
print(f"Mean Cross Validation Score(f1) compared to base model: {B : .2%}
print(f"Mean Cross Validation Score(recall) compared to base \
      model: {C : .2%}")
```

executed in 14ms, finished 13:08:06 2021-07-09

```
Mean Cross Validation Score compared to base model: -10.35%
Mean Cross Validation Score(roc_auc) compared to base model: -7.57%
Mean Cross Validation Score(f1) compared to base model: -0.83%
Mean Cross Validation Score(recall) compared to base model:  0.71%
```

This model perform worse than the base model.

In [239]:
```
A=mean_dt_cv_score12-mean_dt_cv_score11
B=mean_dt_cv_score21-mean_dt_cv_score22
C=mean_dt_cv_score34-mean_dt_cv_score33
D=mean_dt_cv_score43-mean_dt_cv_score44

print(f"Mean Cross Validation Score compared to previous \
      model: {D : .2%}")
print(f"Mean Cross Validation Score(roc_auc) compared to \
    previous model: {A : .2%}")
print(f"Mean Cross Validation Score(f1) compared to previous \
      model: {B : .2%}")
print(f"Mean Cross Validation Score(recall) ompared to \
      previous model: {C : .2%}")
```

executed in 16ms, finished 23:05:26 2021-07-10

```
Mean Cross Validation Score compared to previous       model: -12.71%
```

This model perform worse than the previous model.

4.2 Random Forest

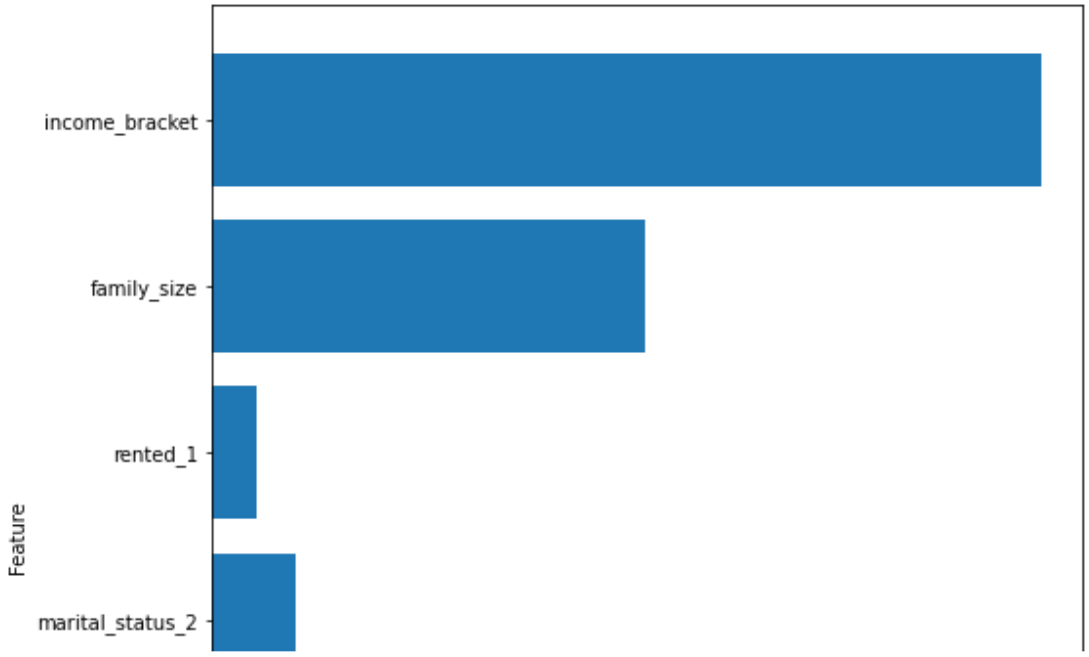
4.2.1 Random Forest

In [71]:
```
forest = RandomForestClassifier(
         class_weight= 'balanced',
         criterion= 'entropy',
         max_depth= 13,
         min_samples_leaf = 10,
         min_samples_split = 2, n_estimators=100)
forest.fit(X_train, y_train)
```

executed in 1.36s, finished 11:27:23 2021-07-08

Out[71]:
```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=13, min_samples_leaf=10)
```

In [72]:
```
1 plot_feature_importances(forest)
```
executed in 159ms, finished 11:27:23 2021-07-08



In [73]:
```
1 rf_clf = RandomForestClassifier()
2 mean_rf_cv_score = np.mean(cross_val_score(rf_clf, X_train, y_train, cv=
3
4 print(f"Mean Cross Validation Score for Random Forest Classifier: {mean_
```
executed in 3.11s, finished 11:27:26 2021-07-08

```
Mean Cross Validation Score for Random Forest Classifier: 99.08%
```

In [74]:
```
1  rf_param_grid = {
2      'criterion': ['gini', 'entropy'],
3      'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
4                    16, 17, 18, 19, 20],
5      'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
6                            14, 15, 16, 17, 18, 19, 20],
7      'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
8                           14, 15, 16, 17, 18, 19, 20],
9      'class_weight': ['balanced', 'smooth']
10 }
```
executed in 13ms, finished 11:27:26 2021-07-08

In [75]:
```
rf_grid_search = GridSearchCV(rf_clf, rf_param_grid, cv=3)
rf_grid_search.fit(X_train, y_train)

print(f"Training Accuracy: {rf_grid_search.best_score_ :.2%}")
print("")
print(f"Optimal Parameters: {rf_grid_search.best_params_}")
```

executed in 16h 27m 33s, finished 03:54:59 2021-07-09

```
Training Accuracy: 81.88%

Optimal Parameters: {'class_weight': 'balanced', 'criterion': 'entropy', 'm
ax_depth': 20, 'min_samples_leaf': 13, 'min_samples_split': 15}
```

In [76]:
```
# Test set predictions
pred = rf_grid_search.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 284ms, finished 03:54:59 2021-07-09

```
[[15440  3965]
 [   99    89]]
              precision    recall  f1-score   support

           0       0.99      0.80      0.88     19405
           1       0.02      0.47      0.04       188

    accuracy                           0.79     19593
   macro avg       0.51      0.63      0.46     19593
weighted avg       0.98      0.79      0.88     19593
```

4.3 Smote

4.3.1 Smote

Smote will be used to create a more balance data set in order to create a better random forest classification.

In [142]:
```
X_smote=X
y_smote=y
```

executed in 3ms, finished 13:18:15 2021-07-09

4.3.2 Smote - Decision Tree

In [218]:

```
# values to evaluate
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
    # define pipeline
    model = DecisionTreeClassifier()
    over = SMOTE(sampling_strategy=0.1, k_neighbors=k)
    under = RandomUnderSampler(sampling_strategy=0.5)
    steps = [('over', over), ('under', under), ('model', model)]
    pipeline = Pipeline(steps=steps)
    # evaluate pipeline
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=
    scores = cross_val_score(pipeline, X_smote, y_smote, scoring='roc_au
                             cv=cv, n_jobs=-1)
    score = np.mean(scores)
    print('> k=%d, Mean ROC AUC: %.3f' % (k, score))
```

executed in 13.4s, finished 21:54:04 2021-07-10

```
> k=1, Mean ROC AUC: 0.715
> k=2, Mean ROC AUC: 0.712
> k=3, Mean ROC AUC: 0.710
> k=4, Mean ROC AUC: 0.709
> k=5, Mean ROC AUC: 0.707
> k=6, Mean ROC AUC: 0.707
> k=7, Mean ROC AUC: 0.705
```

After checking to see weather or not changing k values will assit in creating a better ROC AUC.where the k value is 1 Mean ROC AUc produces the best vaule.

4.3.3 Smote - Random Forest

In [174]:

```
smote1 = SMOTE(sampling_strategy='not majority')
X_train_resampled, y_train_resampled = smote1.fit_sample(X_train, y_trai
smote = RandomForestClassifier()
over = SMOTE(sampling_strategy=.3, k_neighbors=1)
under = RandomUnderSampler(sampling_strategy=.7)
steps = [('over', over), ('under', under), ('model', smote)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
# cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X_smote, y_smote, scoring='roc_auc',
                         cv=10, n_jobs=-1)
score = np.mean(scores)
print('> o=%d, u=%d, Mean ROC AUC: %.3f' % (o, u, score))
```

executed in 14.7s, finished 15:57:26 2021-07-09

```
> o=0, u=0, Mean ROC AUC: 0.717
```

In [171]:
```
smote = RandomForestClassifier()
over = SMOTE(sampling_strategy=.3, k_neighbors=1)
under = RandomUnderSampler(sampling_strategy=.7)
steps = [('over', over), ('under', under), ('model', smote)]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
# cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X_smote, y_smote, scoring='roc_auc',
                         cv=100, n_jobs=-1)
score = np.mean(scores)
print('> o=%d, u=%d, Mean ROC AUC: %.3f' % (o, u, score))
```

executed in 1m 44.1s, finished 15:43:15 2021-07-09

```
> o=0, u=0, Mean ROC AUC: 0.720
```

In [172]:
```
model.fit(X_train, y_train)
```

executed in 4.47s, finished 15:53:20 2021-07-09

Out[172]:
```
RandomForestClassifier()
```

In [173]:
```
# Test set predictions
pred = model.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 844ms, finished 15:53:24 2021-07-09

```
[[19405     0]
 [  188     0]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     19405
           1       0.00      0.00      0.00       188

    accuracy                           0.99     19593
   macro avg       0.50      0.50      0.50     19593
weighted avg       0.98      0.99      0.99     19593
```

4.4 Linear Regression

In [69]:

```
# Now let's compare a few different ratios of minority class to majority
ratios = [0.1, 0.25, 0.33, 0.5, 0.7, 1]
names = ['0.1', '0.25', '0.33','0.5','0.7','even']
colors = sns.color_palette('Set2')

plt.figure(figsize=(10, 8))

for n, ratio in enumerate(ratios):
    # Fit a model
    smote = SMOTE(sampling_strategy=ratio)
    X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_t
    logreg = LogisticRegression(fit_intercept=False, C=1e20, solver ='lb
    model_log = logreg.fit(X_train_resampled, y_train_resampled)
    print(model_log)

    # Predict
    y_hat_test = logreg.predict(X_test)

    y_score = logreg.decision_function(X_test)

    fpr, tpr, thresholds = roc_curve(y_test, y_score)

    print('AUC for {}: {}'.format(names[n], auc(fpr, tpr)))
    print('----------------------------------------------------------------
    lw = 2
    plt.plot(fpr, tpr, color=colors[n],
             lw=lw, label='ROC curve {}'.format(names[n]))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

executed in 4.42s, finished 11:59:12 2021-07-09

```
LogisticRegression(C=1e+20, fit_intercept=False)
AUC for 0.1: 0.5572717878151607
----------------------------------------------------------------------
--------------
LogisticRegression(C=1e+20, fit_intercept=False)
AUC for 0.25: 0.5678119534886271
----------------------------------------------------------------------
--------------
LogisticRegression(C=1e+20, fit_intercept=False)
AUC for 0.33: 0.5636037816531164
----------------------------------------------------------------------
--------------
LogisticRegression(C=1e+20, fit_intercept=False)
```

```
AUC for 0.5: 0.5600666366970566
------------------------------------------------------------------------
-------------
LogisticRegression(C=1e+20, fit_intercept=False)
AUC for 0.7: 0.5572745289380342
------------------------------------------------------------------------
```

The graph above shows a linear regression model along with smote. Different minority and majority is where tested against the true positive and the false positive rates in hopes of finding the best balance graphicly. After graphing the data is very close the line with a slope of 1. This shows that there is not a balance that will produce precise/accurate information on either the training or test data.

```
## Final Model issues
```

In [ ]:

```
X_train, X_test, y_train, y_test= train_test_split(
    X, y, train_size = .75, test_size=.25, random_state=10)
```

In [176]:

```
# Previous original class distribution
print('Original class distribution: \n')
print(y.value_counts())
smote = SMOTE(sampling_strategy='not majority')
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train
# Preview synthetic sample class distribution
print('----------------------------------------')
print('Synthetic sample class distribution: \n')
print(pd.Series(y_train_resampled).value_counts())
```

executed in 128ms, finished 16:15:23 2021-07-09

```
Original class distribution:

0    77640
1      729
Name: redemption_status, dtype: int64
----------------------------------------
Synthetic sample class distribution:

1    58235
0    58235
Name: redemption_status, dtype: int64
```

In [227]:

```
ranfor = RandomForestClassifier(max_depth= 15, min_samples_leaf= 5,
                                min_samples_split= 10)
```

executed in 26ms, finished 22:34:43 2021-07-10

In [228]:
```
ranfor.fit(X_train_resampled, y_train_resampled)
#############
```
executed in 9.66s, finished 22:34:55 2021-07-10

Out[228]:
```
RandomForestClassifier(max_depth=15, min_samples_leaf=5, min_samples_split=
10)
```

In [229]:
```
pred11= ranfor.predict(X_test)
###############
```
executed in 741ms, finished 22:34:55 2021-07-10

In [230]:
```
# Confusion matrix and classification report
print(confusion_matrix(y_test, pred11))
print(classification_report(y_test, pred11))
###############
```
executed in 127ms, finished 22:34:56 2021-07-10

```
[[15690  3715]
 [  108    80]]
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     19405
           1       0.02      0.43      0.04       188

    accuracy                           0.80     19593
   macro avg       0.51      0.62      0.47     19593
weighted avg       0.98      0.80      0.88     19593
```

In [231]:
```
predict12=ranfor.predict(X_train_resampled)
```
executed in 4.13s, finished 22:35:00 2021-07-10

The above confusion matrix and classification report does not seem impressive, but it has a increased f1 scores and decrease. Also the below chart show that even with all this modeling the training set does not do a good job of predicting the data, it is hard to expect that the test data would perform better.

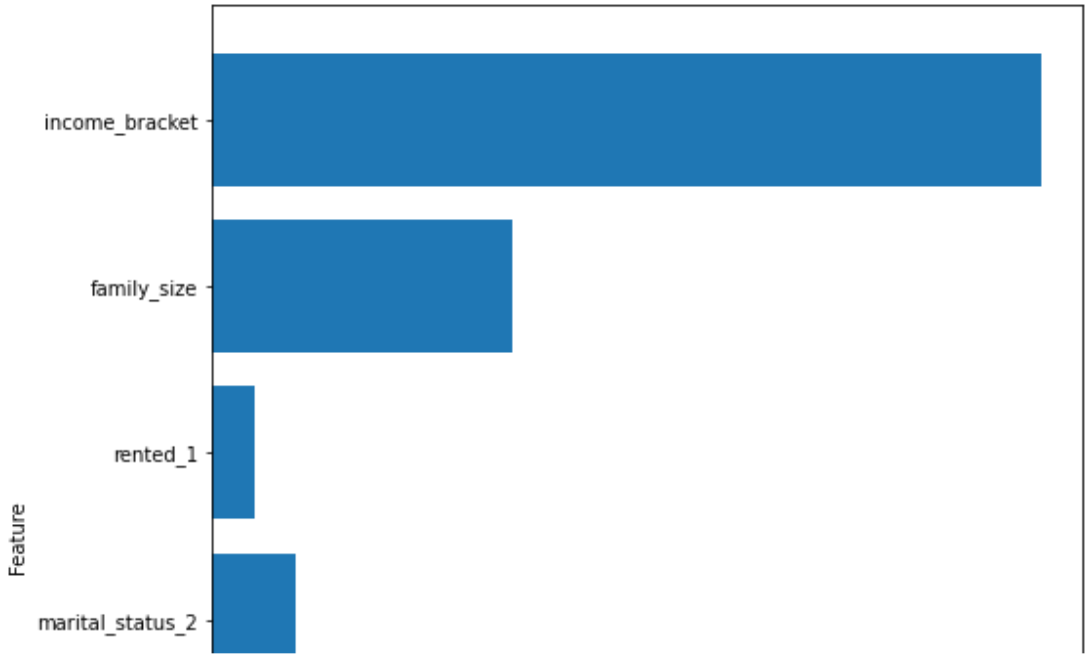
In [ ]:
```
# Confusion matrix and classification report
print(confusion_matrix(y_train_resampled, predict12))
print(classification_report(y_train_resampled, predict12))
```

In [233]:
```
from sklearn.tree import export_graphviz
```
executed in 16ms, finished 22:35:02 2021-07-10

In [221]:
```
1 ranfor.feature_importances_
```
executed in 52ms, finished 22:25:05 2021-07-10

Out[221]:
```
array([0.20690585, 0.02613524, 0.05161692, 0.02587291, 0.18355116,
       0.50591792])
```

In [222]:
```
1 plot_feature_importances(ranfor)
```
executed in 443ms, finished 22:25:07 2021-07-10



Even though the classifier did not perform we do see that across all the models the same 3 features are classified as important an in the same order.