Exploring Marketing Campaign Dataset

Step 1: Download the dataset using this link

link: https://www.kaggle.com/datasets/rodsaldanha/arketing-campaign

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

In [2]:
```
df = pd.read_csv("marketing_campaign.csv",delimiter=';')
df.head()
```

Out[2]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	R
0	5524	1957	Graduation	Single	58138.0	0	0	2012-09-04	
1	2174	1954	Graduation	Single	46344.0	1	1	2014-03-08	
2	4141	1965	Graduation	Together	71613.0	0	0	2013-08-21	
3	6182	1984	Graduation	Together	26646.0	1	0	2014-02-10	
4	5324	1981	PhD	Married	58293.0	1	0	2014-01-19	

5 rows × 29 columns

Step 2: Explore the data fields

In [3]:
```
df.columns
```

Out[3]:
```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
       'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth',
       'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1',
       'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Response'],
      dtype='object')
```

In [4]:
```
df.describe()
```

Out[4]:

	ID	Year_Birth	Income	Kidhome	Teenhome	Recency	Mr
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250	49.109375	303

std	3246.662198	11.984069	25173.076661	0.538398	0.544538	28.962453	336
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000	0
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000	24.000000	23
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000	49.000000	173
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000	74.000000	504
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493

8 rows × 26 columns

In [5]:

```
#find numerical and categorical data:
print('Numerical Features are : ')
print( "'Income', 'Kidhome', 'Teenhome','MntWines', 'MntFruits', 'MntMeatProduc
print('Categorical Features are : ')
print( "'Education','Marital_Status','AcceptedCmp3', 'AcceptedCmp4', 'AcceptedC
```

```
Numerical Features are :
'Income', 'Kidhome', 'Teenhome','MntWines', 'MntFruits', 'MntMeatProducts', 'Mn
tFishProducts', 'MntSweetProducts', 'MntGoldProds', 'NumDealsPurchases', 'NumWe
bPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth'
Categorical Features are :
'Education','Marital_Status','AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'A
cceptedCmp1', 'AcceptedCmp2', 'Complain'
```

Step 3: Check for missing values and outliers

In [6]:

```
#check the NA value
df.isna().sum()
```

Out[6]:

```
ID                      0
Year_Birth              0
Education               0
Marital_Status          0
Income                 24
Kidhome                 0
Teenhome                0
Dt_Customer             0
Recency                 0
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
AcceptedCmp3            0
AcceptedCmp4            0
AcceptedCmp5            0
AcceptedCmp1            0
```

```
AcceptedCmp2           0
Complain               0
Z_CostContact          0
Z_Revenue              0
Response               0
dtype: int64
```

In [7]:
```
df.dropna(inplace=True)
```

In [8]:
```
df.info()
```

```
Int64Index: 2216 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2216 non-null   int64  
 1   Year_Birth           2216 non-null   int64  
 2   Education            2216 non-null   object 
 3   Marital_Status       2216 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2216 non-null   int64  
 6   Teenhome             2216 non-null   int64  
 7   Dt_Customer          2216 non-null   object 
 8   Recency              2216 non-null   int64  
 9   MntWines             2216 non-null   int64  
 10  MntFruits            2216 non-null   int64  
 11  MntMeatProducts      2216 non-null   int64  
 12  MntFishProducts      2216 non-null   int64  
 13  MntSweetProducts     2216 non-null   int64  
 14  MntGoldProds         2216 non-null   int64  
 15  NumDealsPurchases    2216 non-null   int64  
 16  NumWebPurchases      2216 non-null   int64  
 17  NumCatalogPurchases  2216 non-null   int64  
 18  NumStorePurchases    2216 non-null   int64  
 19  NumWebVisitsMonth    2216 non-null   int64  
 20  AcceptedCmp3         2216 non-null   int64  
 21  AcceptedCmp4         2216 non-null   int64  
 22  AcceptedCmp5         2216 non-null   int64  
 23  AcceptedCmp1         2216 non-null   int64  
 24  AcceptedCmp2         2216 non-null   int64  
 25  Complain             2216 non-null   int64  
 26  Z_CostContact        2216 non-null   int64  
 27  Z_Revenue            2216 non-null   int64  
 28  Response             2216 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 519.4+ KB
```

In [9]:
```
#get more information with categorical data
AcceptedCmp1_rate = (df.groupby('AcceptedCmp1').size()/df['AcceptedCmp1'].count
print(AcceptedCmp1_rate)
```

```
AcceptedCmp1
0    93.592058
1     6.407942
dtype: float64
```

In [10]:
```
AcceptedCmp2_rate = (df.groupby('AcceptedCmp2').size()/df['AcceptedCmp2'].count
```

```
print(AcceptedCmp2_rate)
```

```
AcceptedCmp2
0    98.646209
1     1.353791
dtype: float64
```

In [11]:

```
AcceptedCmp3_rate = (df.groupby('AcceptedCmp3').size()/df['AcceptedCmp3'].count
print(AcceptedCmp3_rate)
```

```
AcceptedCmp3
0    92.644404
1     7.355596
dtype: float64
```

In [12]:

```
AcceptedCmp4_rate = (df.groupby('AcceptedCmp4').size()/df['AcceptedCmp4'].count
print(AcceptedCmp4_rate)
```

```
AcceptedCmp4
0    92.599278
1     7.400722
dtype: float64
```

In [13]:

```
AcceptedCmp5_rate = (df.groupby('AcceptedCmp5').size()/df['AcceptedCmp5'].count
print(AcceptedCmp5_rate)
```

```
AcceptedCmp5
0    92.689531
1     7.310469
dtype: float64
```

In [14]:

```
response_rate = (df.groupby('Response').size()/df['Response'].count())*100
print(response_rate)
```

```
Response
0    84.972924
1    15.027076
dtype: float64
```

It's not a balance dataset. The majority class is 0, means most of customers will reject the marketing compaign.

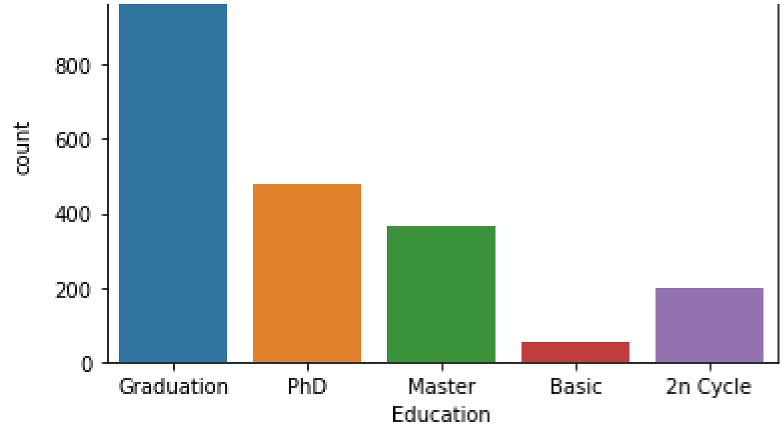
Step 4: Perform feature engineering and EDA

While performing EDA, explore the univariate,bivariate, and Multivariate variables through visualizations

In [15]:

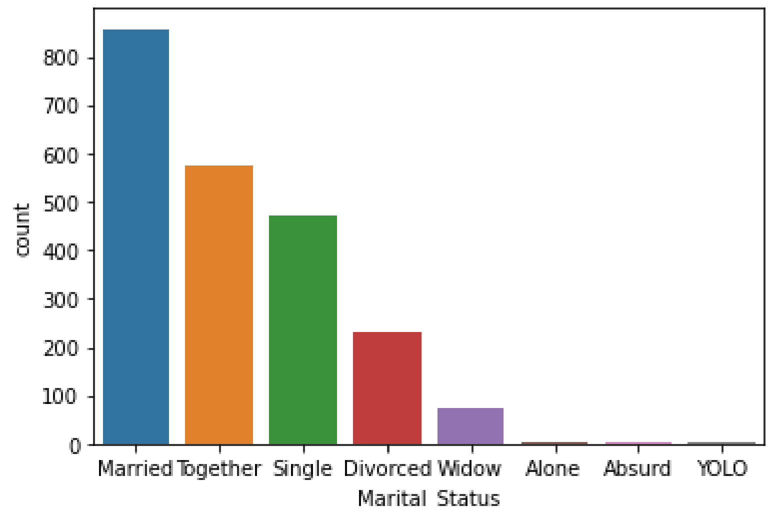
```
sns.countplot(x = 'Education',data = df)
```

Out[15]:



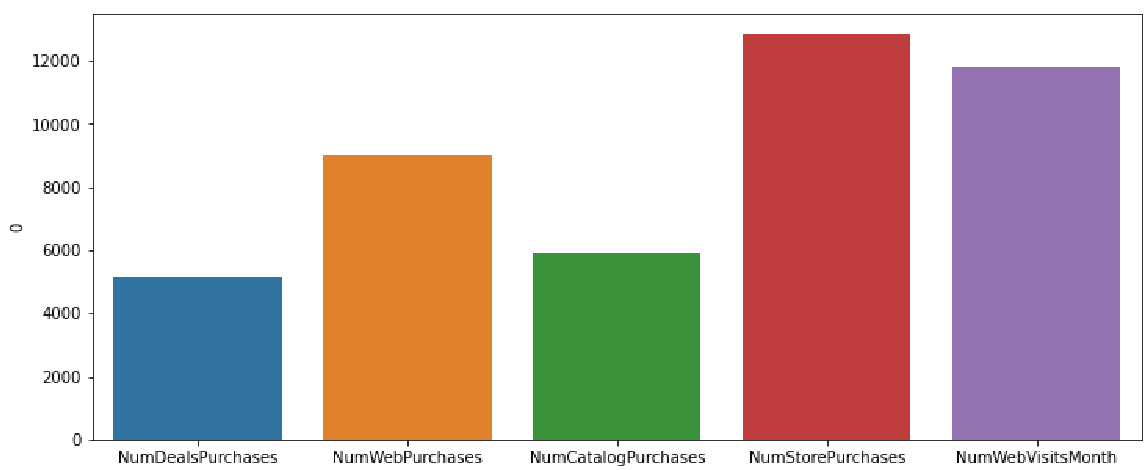
In [16]:
```
sns.countplot(x = 'Marital_Status',data = df, order = df['Marital_Status'].valu
```

Out[16]:



In [17]:
```
#which purches way is the customers' favorite
spent_tot = pd.DataFrame(df[['NumDealsPurchases', 'NumWebPurchases', 'NumCatalo
plt.figure(figsize=(12,5))
sns.barplot(x = spent_tot.index,y = spent_tot[0], data = spent_tot)
```

Out[17]:

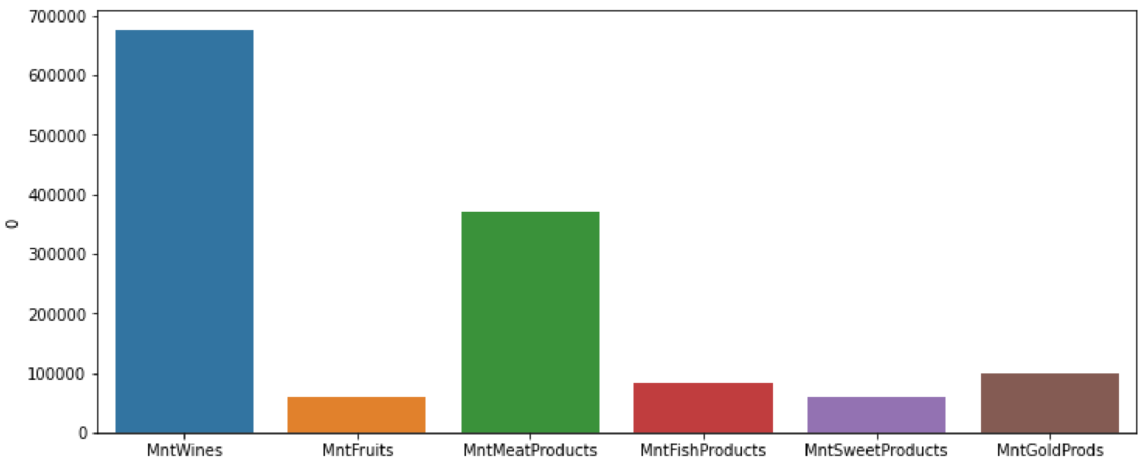


In [18]:

In [18]:

```
#What type of groceries are most popular
spent_tot = pd.DataFrame(df[[ 'MntWines', 'MntFruits',
       'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
       'MntGoldProds']].sum())
plt.figure(figsize=(12,5))
sns.barplot(x = spent_tot.index,y = spent_tot[0], data = spent_tot)
```

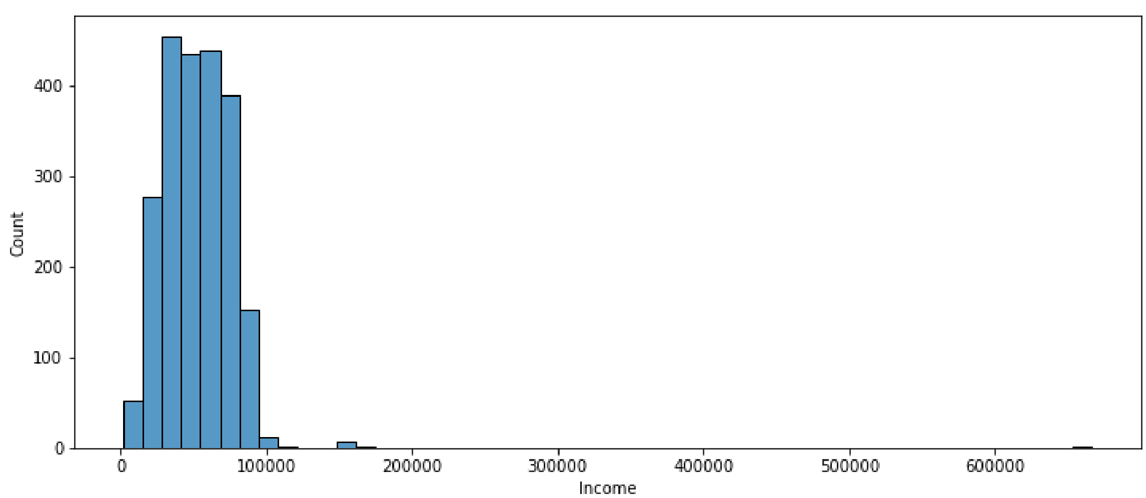
Out[18]:



In [19]:

```
#Find the data distribution
plt.figure(figsize=(12,5))
sns.histplot(x='Income',bins=50,data=df,multiple='stack')
```

Out[19]:

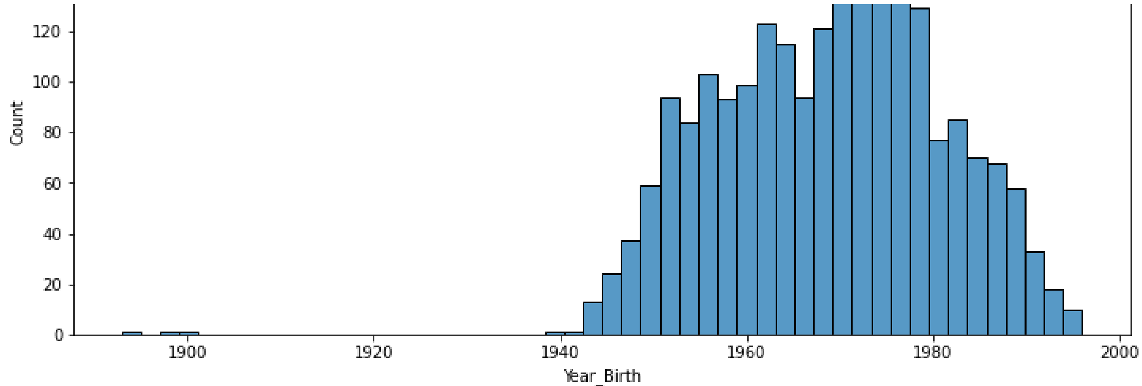


After remove some fewer outlier data, I find the Income is right skewed normal distribution.

In [20]:

```
plt.figure(figsize=(12,5))
sns.histplot(x='Year_Birth',bins=50,data=df,multiple='stack')
```

Out[20]:



After remove some fewrt outlier data, the Year_Birth is normal distribution. So I decided delete this feature.

Step 5: Apply ML models for classification

In [21]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn import svm, tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import recall_score
```

In [22]:

```
#For the object data type columns, get dummies
df1 = pd.get_dummies(df, drop_first = True, columns=['Education','Marital_Statu

df1
```

Out[22]:

	ID	Year_Birth	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	Mı
0	5524	1957	58138.0	0	0	2012-09-04	58	635	
1	2174	1954	46344.0	1	1	2014-03-08	38	11	
2	4141	1965	71613.0	0	0	2013-08-21	26	426	
3	6182	1984	26646.0	1	0	2014-02-10	26	11	
4	5324	1981	58293.0	1	0	2014-01-19	94	173	
...	...	...	...	...	...	...	...	...	
2235	10870	1967	61223.0	0	1	2013-06-13	46	709	
2236	4001	1946	64014.0	2	1	2014-06-10	56	406	
2237	7270	1981	56981.0	0	0	2014-01-25	91	908	
2238	8235	1956	69245.0	0	1	2014-01-24	8	428	

2239	9405	1954	52869.0	1	1	2012-10-15	40	84

2216 rows × 38 columns

In [23]:
```
df1.columns
```

Out[23]:
```
Index(['ID', 'Year_Birth', 'Income', 'Kidhome', 'Teenhome', 'Dt_Customer',
       'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Complain', 'Z_CostContact', 'Z_Revenue', 'Response', 'Education_Basic',
       'Education_Graduation', 'Education_Master', 'Education_PhD',
       'Marital_Status_Alone', 'Marital_Status_Divorced',
       'Marital_Status_Married', 'Marital_Status_Single',
       'Marital_Status_Together', 'Marital_Status_Widow',
       'Marital_Status_YOLO'],
      dtype='object')
```

In [24]:
```
#drop the 'Z_CostContact', 'Z_Revenue','Dt_Customer', which with no meaning
df2 = df1.drop(['Z_CostContact', 'Z_Revenue','Dt_Customer'], axis=1)
```

In [25]:
```
df2.columns
```

Out[25]:
```
Index(['ID', 'Year_Birth', 'Income', 'Kidhome', 'Teenhome', 'Recency',
       'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts',
       'MntSweetProducts', 'MntGoldProds', 'NumDealsPurchases',
       'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases',
       'NumWebVisitsMonth', 'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5',
       'AcceptedCmp1', 'AcceptedCmp2', 'Complain', 'Response',
       'Education_Basic', 'Education_Graduation', 'Education_Master',
       'Education_PhD', 'Marital_Status_Alone', 'Marital_Status_Divorced',
       'Marital_Status_Married', 'Marital_Status_Single',
       'Marital_Status_Together', 'Marital_Status_Widow',
       'Marital_Status_YOLO'],
      dtype='object')
```

In [26]:
```
X = df2[['Income', 'Kidhome', 'Teenhome', 'Recency',
        'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts',
        'MntSweetProducts', 'MntGoldProds', 'NumDealsPurchases',
        'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases',
        'NumWebVisitsMonth', 'AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5',
        'AcceptedCmp1', 'AcceptedCmp2', 'Complain',
        'Education_Basic', 'Education_Graduation', 'Education_Master',
        'Education_PhD', 'Marital_Status_Alone', 'Marital_Status_Divorced',
        'Marital_Status_Married', 'Marital_Status_Single',
        'Marital_Status_Together', 'Marital_Status_Widow',
        'Marital_Status_YOLO']]
y = df2['Response']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random
```

In [27]:

```
X_train.info()
```

```
Int64Index: 1329 entries, 1011 to 853
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Income                   1329 non-null   float64
 1   Kidhome                  1329 non-null   int64  
 2   Teenhome                 1329 non-null   int64  
 3   Recency                  1329 non-null   int64  
 4   MntWines                 1329 non-null   int64  
 5   MntFruits                1329 non-null   int64  
 6   MntMeatProducts          1329 non-null   int64  
 7   MntFishProducts          1329 non-null   int64  
 8   MntSweetProducts         1329 non-null   int64  
 9   MntGoldProds             1329 non-null   int64  
 10  NumDealsPurchases        1329 non-null   int64  
 11  NumWebPurchases          1329 non-null   int64  
 12  NumCatalogPurchases      1329 non-null   int64  
 13  NumStorePurchases        1329 non-null   int64  
 14  NumWebVisitsMonth        1329 non-null   int64  
 15  AcceptedCmp3             1329 non-null   int64  
 16  AcceptedCmp4             1329 non-null   int64  
 17  AcceptedCmp5             1329 non-null   int64  
 18  AcceptedCmp1             1329 non-null   int64  
 19  AcceptedCmp2             1329 non-null   int64  
 20  Complain                 1329 non-null   int64  
 21  Education_Basic          1329 non-null   uint8  
 22  Education_Graduation     1329 non-null   uint8  
 23  Education_Master         1329 non-null   uint8  
 24  Education_PhD            1329 non-null   uint8  
 25  Marital_Status_Alone     1329 non-null   uint8  
 26  Marital_Status_Divorced  1329 non-null   uint8  
 27  Marital_Status_Married   1329 non-null   uint8  
 28  Marital_Status_Single    1329 non-null   uint8  
 29  Marital_Status_Together  1329 non-null   uint8  
 30  Marital_Status_Widow     1329 non-null   uint8  
 31  Marital_Status_YOLO      1329 non-null   uint8  
dtypes: float64(1), int64(20), uint8(11)
memory usage: 242.7 KB
```

In [28]:

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [29]:

```
def ACC(predict, model_name):
    accuracy = np.mean(predict == y_test)
    print("Accuracy of", model_name, "is", round(float(accuracy), 3))
    #return: accuracy

def TPR(predict,model_name):
    TPR = recall_score(y_test, predict)
    print("TPR of", model_name, "is", round(float(TPR), 3))
    #return:True positive rate
```

In [32]:

```
def Logistic_Reg():
    lg_classifier = LogisticRegression(solver = "lbfgs")
```

```
    lg_classifier.fit(X_train, y_train)
    pred_Y = lg_classifier.predict(X_test)
    return ACC(pred_Y,'logistic_regression :'), TPR(pred_Y,'logistic_regression

def SVM():
    svm_classifier = svm.SVC(kernel='linear')
    svm_classifier.fit(X_train, y_train)
    pred_Y = svm_classifier.predict(X_test)
    return ACC(pred_Y, 'SVM :'), TPR(pred_Y, 'SVM :')

def forest():
    rf_classifier=RandomForestClassifier()
    rf_classifier.fit(X_train, y_train)
    pred_Y = rf_classifier.predict(X_test)
    return ACC(pred_Y, 'random_forest :'), TPR(pred_Y, 'random_forest :')

def knn():
    knn_classifier = KNeighborsClassifier(n_neighbors = 7)
    knn_classifier.fit(X_train, y_train)
    pred_Y = knn_classifier.predict(X_test)
    return ACC(pred_Y, 'knn :'), TPR(pred_Y, 'knn :')

def NB():
    nb_classifier = GaussianNB()
    nb_classifier.fit(X_train, y_train)
    pred_Y = nb_classifier.predict(X_test)
    return ACC(pred_Y, 'Naive bayes :'), TPR(pred_Y, 'Naive bayes :')

def TREE():
    tree_classifire = tree.DecisionTreeClassifier(criterion = 'entropy').fit(X_
    pred_Y = tree_classifire.predict(X_test)
    return ACC(pred_Y, 'Decision tree :'), TPR(pred_Y, 'Decision tree :')
```

In [34]:

```
print(Logistic_Reg(),
SVM(),
forest(),
knn(),
NB(),
TREE())
```

```
Accuracy of logistic_regression : is 0.886
TPR of logistic_regression : is 0.438
Accuracy of SVM : is 0.885
TPR of SVM : is 0.383
Accuracy of random_forest : is 0.869
TPR of random_forest : is 0.227
Accuracy of knn : is 0.875
TPR of knn : is 0.234
Accuracy of Naive bayes : is 0.834
TPR of Naive bayes : is 0.477
Accuracy of Decision tree : is 0.846
TPR of Decision tree : is 0.469
(None, None) (None, None) (None, None) (None, None) (None, None) (None, None)
```

Based on the accuracy and TPR, Logistic Regression performed best. So I decided to find the feature importance through Logistic Regression.
But all of them have low TPR.That's make sence, because it's a imbalance dataset, so the classification model tend to predict to the majority class. But that's not I want, because in

this research scenario, the company don't want to miss any one who will accept. So It's better that we mistakenly predict a customer will accept than we mistakenly predict a customer won't accept but actually he will.

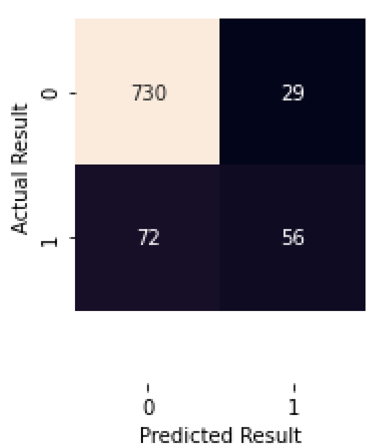
In [35]:
```
# analyse the logestic regression first
from sklearn.metrics import confusion_matrix
from sklearn import metrics
clf = LogisticRegression(solver = "lbfgs")
clf.fit(X_train, y_train)
pred_Y = clf.predict(X_test)
print(metrics.classification_report(y_test, pred_Y))
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.94       759
           1       0.66      0.44      0.53       128

    accuracy                           0.89       887
   macro avg       0.78      0.70      0.73       887
weighted avg       0.87      0.89      0.88       887
```

In [36]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(y_test, pred_Y)
sns.heatmap(mat, fmt='g', square=True, annot=True, cbar=False)
plt.xlabel("Predicted Result")
plt.ylabel("Actual Result")
a, b = plt.ylim()
a += 0.5
b -= 0.5
plt.ylim(a, b)
plt.show()
```

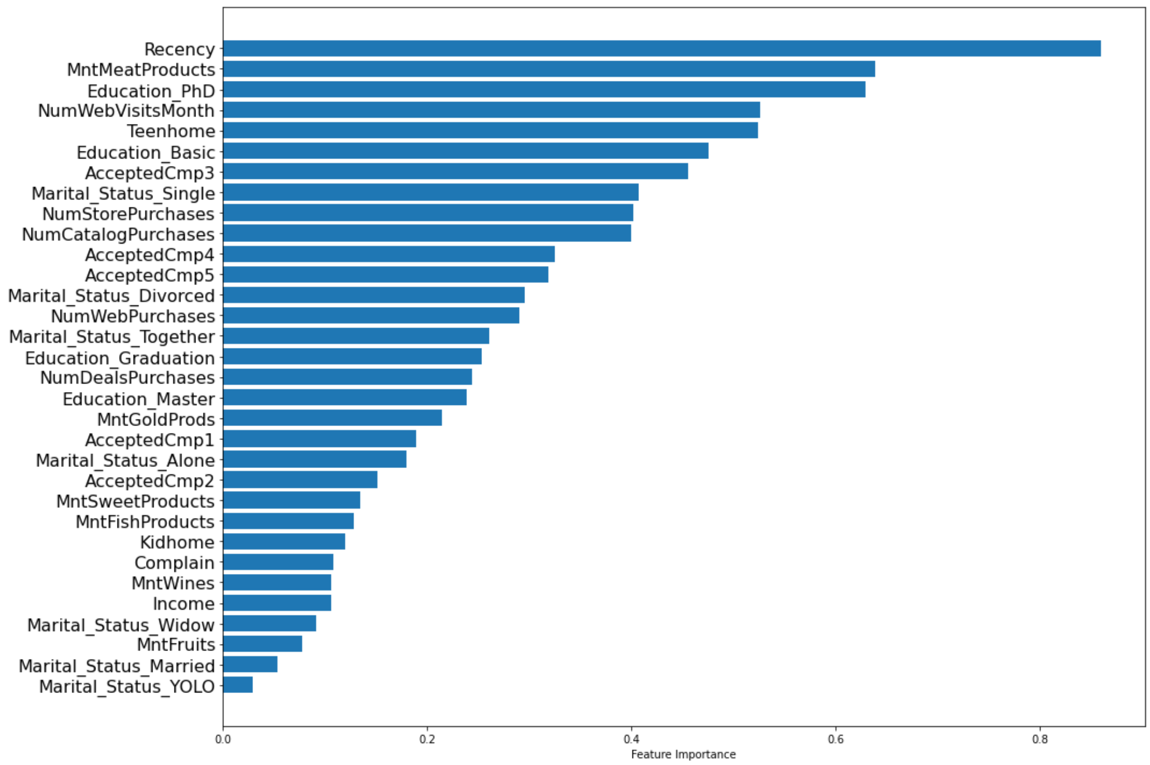


In [37]:
```
feature_importance = abs(clf.coef_[0])
#feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
```

```
pos = np.arange(sorted_idx.shape[0]) + .5

featfig = plt.figure(figsize = (15,10))
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
featax.set_yticks(pos)
featax.set_yticklabels(np.array(X.columns)[sorted_idx], fontsize=16)
featax.set_xlabel('Feature Importance')

plt.tight_layout()
plt.show()
```



'Recency','MntMeatProducts','Education_phD' are the three most important feature to predict the response.

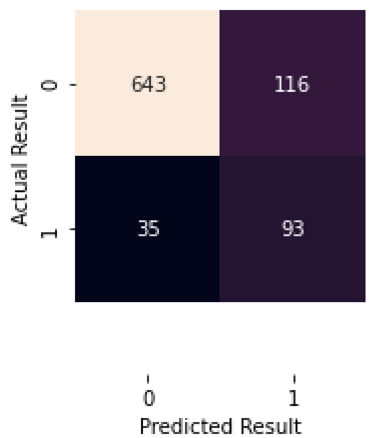
In [38]:

```
#Use Penalized-SVM on the original imbalanced dataset
# we can add class_weight='balanced' to add panalize mistake
svm_classifier = svm.SVC(class_weight='balanced', probability=True)
svm_classifier.fit(X_train, y_train)
pred_Y = svm_classifier.predict(X_test)
```

In [39]:

```
#The performence of penalized SVM
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(y_test, pred_Y)
sns.heatmap(mat, fmt='g', square=True, annot=True, cbar=False)
plt.xlabel("Predicted Result")
plt.ylabel("Actual Result")
a, b = plt.ylim()
a += 0.5
b -= 0.5
```

```
b -= 0.5
plt.ylim(a, b)
plt.show()
```



In [40]:

```
print(metrics.classification_report(y_test, pred_Y))
```

```
              precision    recall  f1-score   support

           0       0.95      0.85      0.89       759
           1       0.44      0.73      0.55       128

    accuracy                           0.83       887
   macro avg       0.70      0.79      0.72       887
weighted avg       0.88      0.83      0.85       887
```

accuracy rate: 0.83

sensitivity rate: 0.73

specificity rate: 0.85

precision: 0.44

balanced accuracy:0.79