Snehal Yadav

Analytics Challenge Solution

Penn National Gaming

Contact Details:

714-908-6503

ysnehal28@gmail.com

GitHub: https://github.com/snehal-y

LinkedIn: https://www.linkedin.com/in/snehal287/

Contents

Challenge:

Recommend a course of action regarding the mystery spin experiment described below. Should we implement Variant 1, Variant 2, revert to the control, or wait longer?

Objective: Find which variant is impactful to gain more revenue. Revenue is gained when the player buys something from the Buy Page of Mystery Spin game. This means either we need to find what all factors make players to purchase during their experiment or what are the factors which are making more views on the Buy Page.

Target (Dependent variable) => bc (if player has made a purchase or not)

Analysis:

1. I would prefer either Variant 0 or Variant 1. Because with Variant0 or Variant 2, the player should make number of spins greater than 73068 but less than 110351, then only player will possibly make purchase. With 91% confidence, 78 players out of 79 players will possibly make a purchase during their experiment.
2. A player will mostly likely (73%) make a purchase if he/she visits the Buy Page for 5 to 7 times in week from the day the player has entered the experiment. This is possible only if the player makes number of spins more than 8,946.
3. Irrespective of the type of the Variant, a player will most likely make a purchase during the experiment if the player finds something engaging in the game. In this case, number of spins followed by heartbeats seemed to be engaging factors for any player.

Recommendations:

1. Engaging factors such as Spins and heartbeats are required to acquire and sustain existing of players.
2. Adding more engaging features would be beneficial.
3. Rewarding existing players would be beneficial to keep them engaged and attracted towards the game.

Data Analysis:

I used Python version 3 and Tableau for doing the data analysis of given dataset. To achieve statistical analysis using Python, I have used *SciKit-learn* library along with *numpy, pandas, Seaborn, matplotlib* libraries. I have also used *graphviz* suit to visualize the data in graphical way.

Scikit-learn is a free machine learning library. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

Importing Data Set

```
#import required libraries

import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mp
import matplotlib.pyplot as plt
import pydotplus
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from sklearn.metrics import classification_report
import researchpy as rp
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.stats.multicomp
```

```
os.getcwd()
os.chdir(r"C:\Users\url\to\target\folder")
```

```
df = pd.read_csv('expByuser.csv')
```

Read_csv() method provided by *pandas* module to read *csv* file and saved the data in *df* named DataFrame.

Data Exploration

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 433514 entries, 0 to 433513
Data columns (total 17 columns):
variant              433514 non-null int64
entry_date           433514 non-null object
u_custom_platform    433514 non-null object
version_name         433514 non-null object
install_date         433514 non-null object
first_pay_date       14511 non-null object
playertype           433514 non-null object
revenue              433514 non-null float64
bc                   433514 non-null int64
txns                 433514 non-null int64
heartbeats           433514 non-null int64
spins                433514 non-null int64
buypageviews         433514 non-null int64
d1                   433514 non-null int64
d7                   433514 non-null int64
d14                  433514 non-null int64
revenue_before       433514 non-null float64
dtypes: float64(2), int64(9), object(6)
memory usage: 56.2+ MB
```

In any kind of data analysis, knowing the datatype of the variables is important. It helps us in knowing which operation to be performed. Df.infor() method is used to know the data types of the variables.

The columns with *'object'* dtype are the possible categorical features in df dataset. Because categorical features are 'possible' values and they do not have certainty. So, we should not completely rely on .info() to get the real data type of the values of a feature, as some missing values that are represented as strings in a continuous feature can coerce it to read them as *object* dtypes.

- In this case, *variant*, *bc*, *d1, d7 and d14* are supposed to be the categorical variables. However, they are identified as *interger* variables.

```
import datetime
df['entry_date'] = pd.to_datetime(df.entry_date)
df['install_date'] = pd.to_datetime(df.install_date)
df['first_pay_date'] = pd.to_datetime(df.first_pay_date)
```

```
df.dtypes
```

```
variant                    int64
entry_date          datetime64[ns]
u_custom_platform          object
version_name               object
install_date        datetime64[ns]
first_pay_date      datetime64[ns]
playertype                 object
revenue                   float64
bc                          int64
txns                        int64
heartbeats                  int64
spins                       int64
buypageviews                int64
d1                          int64
d7                          int64
d14                         int64
revenue_before            float64
dtype: object
```

- Another problem is that some variables indicating date values have the data type as *Object.* I converted them to *datetime64[ns]* data type.

Data Visualization

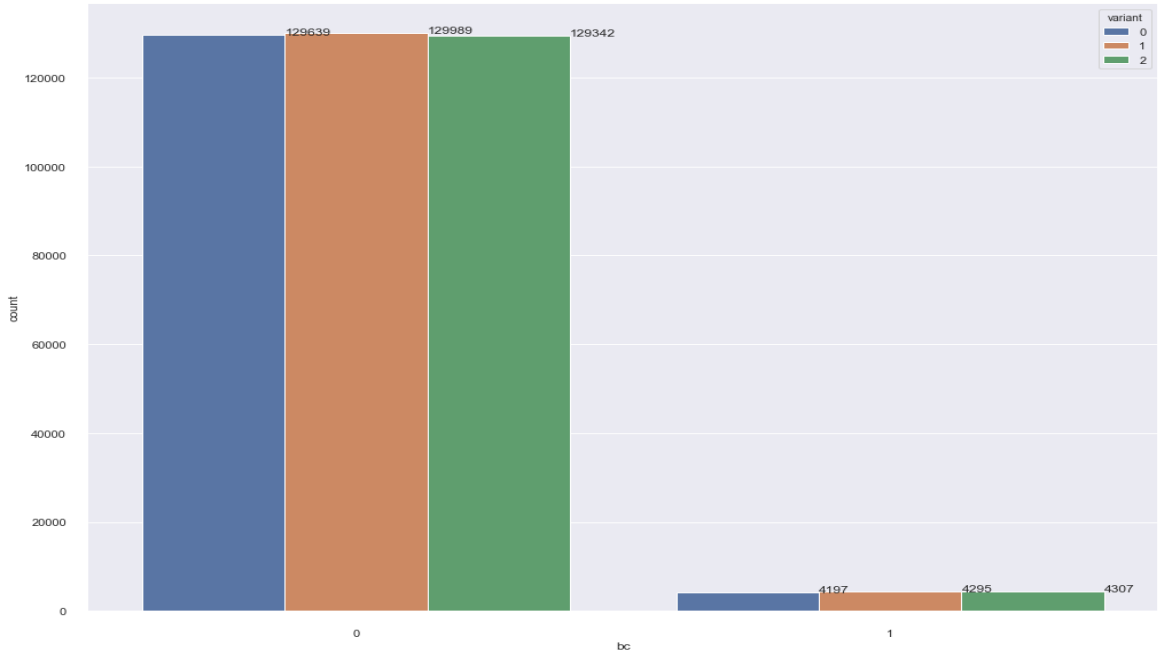
To uncover more insights from the given data set, I visualized the data by creating plots. Since I need to find the type of variants used in the experiment for better revenue generation, I plotted all graphs against type of variants.

For visualization, I used Seaborn library.

i. Purchase (bc) Vs. Variant

```
# set the background colour of the plot to white
sns.set(style="whitegrid", color_codes=True)
# setting the plot size for all plots
sns.set(rc={'figure.figsize':(16.7,12.27)})
# create a countplot
ax = sns.countplot('bc',data=df,hue = 'variant')

for p in ax.patches:
        length = p.get_height()
        x = p.get_x() + p.get_width()
        y = p.get_y() + p.get_height()
        ax.annotate(length, (x, y))
# Remove the top and down margin
sns.despine(offset=10, trim=True)
# display the plot
plt.show()
```

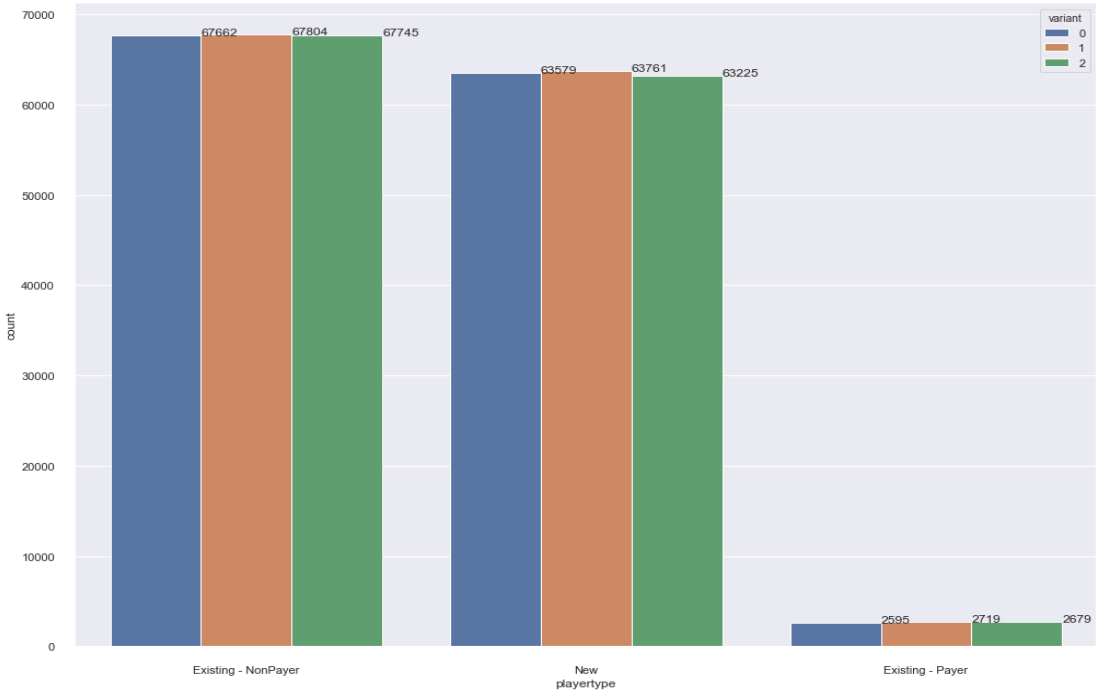
- Below bar-chart implies that the data set has MORE records of (*bc=0*)players who DID NOT made a purchase during the experiment than records of (*bc=1*)players who made purchase during the experiment.
- We can also see that Variant 1 is most frequently used option by all the players.
- However, more players tend to make purchase when they use Variant 2.
- This barchart also provides an idea that the data is imbalaced. Most of the players are not prefering to make a purchase during the experiment.
- Therefore, most of the data is accumulated near y-axis.



ii. ***Player Type Vs. Variant Type***

```
# create a countplot
ax = sns.countplot('playertype',data=df,hue = 'variant')

for p in ax.patches:
        length = p.get_height()
        x = p.get_x() + p.get_width()
        y = p.get_y() + p.get_height()
        ax.annotate(length, (x, y))
# Remove the top and down margin
sns.despine(offset=10, trim=True)
# display the plot
plt.show()
```

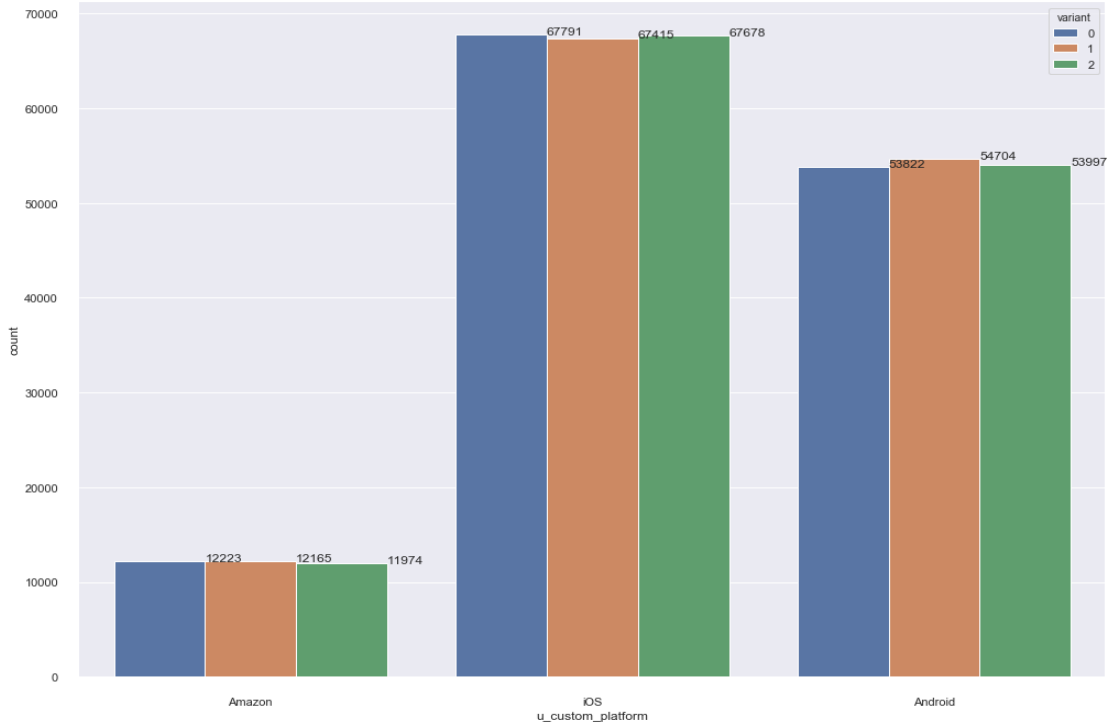
- Below bar-chart implies that most of the *Existing-Payers* are NOT making a purchase in the experiment followed by a *New Players*.
- This shows that the Mystery Spin game certainly has some engaging characteristics which are attracting new players and making them purchase during the experiment.



iii. Platform Vs Variant

```
# create a countplot
ax = sns.countplot('u_custom_platform',data=df,hue = 'variant')

for p in ax.patches:
        length = p.get_height()
        x = p.get_x() + p.get_width()
        y = p.get_y() + p.get_height()
        ax.annotate(length, (x, y))
# Remove the top and down margin
sns.despine(offset=10, trim=True)
# display the plot
plt.show()
```

- From below bar-chart it is clear that iOS users are making a MORE purchase in the experiment followed by the Android users.



Data Pre-processing

i. *Handling data types*

From initial data exploration, I found that most of the variables are *Object* data type. Generally, numerical data types are easier to understand for any machine learning algorithm than categorical variables. Using LabelEncoder() method of Scikit-learn python library, I converted categorical labels from the given data set into numerical variables.

```
#Finding category types of each categorical variable
print("variant : ",df['variant'].unique())
print("u_custom_platform : ",df['u_custom_platform'].unique())
print("playertype : ",df['playertype'].unique())
print("version_name : ",df['version_name'].unique())
```

```
variant :  [0 1 2]
u_custom_platform :  ['Amazon' 'iOS' 'Android']
playertype :  ['Existing - NonPayer' 'New' 'Existing - Payer']
version_name :  ['1.16.0' '1.15.0' '1.16.5']
```

```
#import the necessary module
from sklearn import preprocessing
# create the Labelencoder object
le = preprocessing.LabelEncoder()
#convert the categorical columns into numeric
df['u_custom_platform'] = le.fit_transform(df['u_custom_platform'])
df['playertype'] = le.fit_transform(df['playertype'])
df['version_name'] = le.fit_transform(df['version_name'])
#display the initial records
df.head()
```

	variant	entry_date	u_custom_platform	version_name	install_date	first_pay_date	playertype	revenue	bc	txns	heartbeats	spins	buypageviews	d1
0	0	2016-01-30	0	1	2015-08-27	NaT	0	0.0	0	0	19	496	0	1
1	0	2016-01-31	0	1	2016-01-01	NaT	0	0.0	0	0	5	106	0	0
2	1	2016-01-25	2	0	2016-01-25	NaT	2	0.0	0	0	2	6	0	0
3	1	2016-01-27	2	0	2016-01-27	NaT	2	0.0	0	0	11	422	0	0
4	1	2016-01-16	1	0	2015-11-05	NaT	0	0.0	0	0	11	176	0	0

ii. *Handling imbalanced dataset*

Imbalanced classes give incorrect “accuracy” of the models. To overcome this problem, I performed up-sampling on the minority class.

In our data set, variable *bc* is imbalanced. This variable is one of our target variables. In existing data set, variable *bc* has following count of records for each value.

```
df['bc'].value_counts()
```

```
0    388970
1     12799
Name: bc, dtype: int64
```

Out of total 401,769 records, 12,799 are showing that players have made purchase during the of their experiment. This constitutes only 3.18% of total data related to purchase.

Because of such imbalanced data set, the chances of bias analysis increases. To overcome his problem, I performed up-sampling on bc=1. Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce its signal.

```
from sklearn.utils import resample
```

```
# Separate majority and minority classes
df_majority = df[df.bc==0]
df_minority = df[df.bc==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=388970,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled.bc.value_counts()
```

```
1    388970
0    388970
Name: bc, dtype: int64
```

The new DataFrame has more observations than the original, and the ratio of the two classes is now 1:1.

Relationship between independent and dependent variables:

Before partitioning the data, I checked data types of the all variables in the data set. And as per data exploration step, I found that the dataset contains categorical as well as numerical data types. After doing required conversion, I looked for other parameters before building a model.

Since the given data set is imbalanced data set, I preferred doing re-sampling of the given data set instead of using original data set.

```
df_anova2 = df_upsampled[['variant','entry_date','version_name','install_date','first_pay_date','u_custom_platform','playerty
```

Then I also checked if there is any significant relationship between independent and dependent variables. I performed 2-way ANOVA to see the relationship between independent and dependent variables.

2-way ANOVA:

The **two-way ANOVA** test compares the mean differences between groups that have been split on **two** independent variables (called factors). The primary purpose of a **two-way ANOVA** is to understand if there is an interaction between the **two** independent variables on the dependent variable.

```
model = ols('buypageviews ~ C(variant)*C(bc)', df_anova2).fit()
print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f_pvalue: .4f}")
```

```
Overall model F( 5, 777934) =  2470.750, p =  0.0000
```

```
res = sm.stats.anova_lm(model, typ= 2)
res
```

	sum_sq	df	F	PR(>F)
C(variant)	7.163342e+04	2.0	34.921897	6.827999e-16
C(bc)	1.247632e+07	1.0	12164.620019	0.000000e+00
C(variant):C(bc)	1.335585e+05	2.0	65.110904	5.309578e-29
Residual	7.978672e+08	777934.0	NaN	NaN

Above model find the relationship between two categorical independent variables against '*buypageviews*' numerical dependent variable. I found that this model is overall significant. I also checked the assumptions of the ANOVA, normality and homogeneity of variance. *Statsmodels* already provides model diagnostics in the model summary table.

From above ANOVA table, I concluded that there is a significant relation between independent and dependent variables. The p-value for interaction between *variant* and *bc* is negligible.

```
model.summary()
```

OLS Regression Results

Dep. Variable:	buypageviews	**R-squared:**	0.016
Model:	OLS	**Adj. R-squared:**	0.016
Method:	Least Squares	**F-statistic:**	2471.
Date:	Thu, 18 Jul 2019	**Prob (F-statistic):**	0.00
Time:	13:30:53	**Log-Likelihood:**	-3.8006e+06
No. Observations:	777940	**AIC:**	7.601e+06
Df Residuals:	777934	**BIC:**	7.601e+06
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.9112	0.089	21.487	0.000	1.737	2.085
C(variant)[T.1]	0.2746	0.126	2.185	0.029	0.028	0.521
C(variant)[T.2]	0.1948	0.126	1.548	0.122	-0.052	0.442
C(bc)[T.1]	9.1893	0.126	72.733	0.000	8.942	9.437
C(variant)[T.1]:C(bc)[T.1]	-1.7710	0.178	-9.949	0.000	-2.120	-1.422
C(variant)[T.2]:C(bc)[T.1]	-1.7527	0.178	-9.837	0.000	-2.102	-1.403

Omnibus:	2867467.683	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1889837555891.662
Skew:	79.754	**Prob(JB):**	0.00
Kurtosis:	7636.963	**Cond. No.**	9.80

Condition Number assess multicollinearity. Condition Number values over 20 are indicative of multicollinearity. The model passes the assumption check, which is excellent.

Similarly, I checked the relationship between other categorical variables from the data set against numerical variables.

I found below with a better R-square value as compared to other R-square values.

```
model = ols('txns ~ C(variant)*C(bc)*C(d1)*C(d7)*C(d14)*C(playertype)', df_anova2).fit()

# Seeing if the overall model is significant
print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f_pvalue: .4f}")
```

```
Overall model F( 143, 777796) =  687.297, p =  0.0000
```

```
model.summary()
```

OLS Regression Results

Dep. Variable:	txns	**R-squared:**	0.112
Model:	OLS	**Adj. R-squared:**	0.112
Method:	Least Squares	**F-statistic:**	687.3
Date:	Thu, 18 Jul 2019	**Prob (F-statistic):**	0.00
Time:	10:25:41	**Log-Likelihood:**	-2.7604e+06
No. Observations:	777940	**AIC:**	5.521e+06
Df Residuals:	777796	**BIC:**	5.523e+06
Df Model:	143		
Covariance Type:	nonrobust		

This model is also overall significant. I also checked the assumptions of the ANOVA, normality and homogeneity of variance. *Statsmodels* already provides model diagnostics in the model summary table.

From above ANOVA table, I concluded that there is a significant relation between independent and dependent variables. The p-value for interaction between *txns* and other variables is negligible. Using these variables for building a model for the prediction of purchase action by the players.

Partitioning data into Training and Testing data sets:

A Machine Learning algorithm needs to be trained on a set of data to learn the relationships between different features and how these features affect the target variable. For this we need to divide the entire data set into two sets. One is the training set on which we are going to train our algorithm to build a model. The other is the testing set on which we will test our model to see how accurate its predictions are.

However, I have already selected features variables based on ANOVA test analysis.

```
#getting the features values as your taining set
data = df[['variant','d1','d7','d14','spins','buypageviews']]

#assigning the 'bc' column as target
#because we are checkig which all factors are attracting players to make a purchase
target = df.iloc[:,8]
```

Imported *train_test_split* library to separate the data set into training and testing data sets.

```
#import the necessary module
from sklearn.model_selection import train_test_split
```

```
#split data set into train and test sets. 20% data is used for testing the model.
data_train, data_test, target_train, target_test = train_test_split(data,target, test_size = 0.20, random_state = 10)
```

Building a Model

I have built a prediction model. Since our data has multiple classes and also our target variable a categorical variable, I preferred below algorithms. However, before going ahead with any algorithms I checked few things.

Such as:

- More than 50 samples – Check
- Are we predicting a category – Check
- We have labeled data? (data with clear names like opportunity amount etc.) – Check
- Check Less than 100k samples – Check

Based on the checklist that I used below algorithms.

1. Naive Bayes
2. Decision Tree Analysis

We can also use following data algorithm for prediction and classification of this data set.

1. Linear SVC
2. K-Neighbours Classifier
3. Multinomial Regression

Naïve Bayes

On a very high level a Naive-Bayes algorithm calculates the probability of the connection of a feature with a target variable and then it selects the feature with the highest probability.

```
# import the necessary module
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

```
#create an object of the type GaussianNB
gnb = GaussianNB()
```

```
#train the algorithm on training data and predict using the testing data
pred = gnb.fit(data_train, target_train).predict(data_test)
```

```
#print the accuracy score of the model
print("Naive-Bayes accuracy : ",accuracy_score(target_test, pred, normalize = True))
```

```
Naive-Bayes accuracy :  0.9552729173407671
```

This model can predict 95% correctly if a player is going to make a purchase or not based on the features (independent variables) provided to this model.

```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix:')
unique_label = np.unique(target_test)
print(pd.DataFrame(confusion_matrix(target_test, pred,labels = unique_label)))
```

```
Confusion Matrix:
       0     1
0  76304  1471
1   2123   456
```

As we can see, the amount of errors is pretty balanced between purchase made and purchase was not made.
1471 records whhich show that the user has NOT made purchase is classified as those records where users have made purchase. 2123 purchase records were classified as purchase NOT made.

```
from sklearn.metrics import classification_report
print(classification_report(target_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     77775
           1       0.24      0.18      0.20      2579

   micro avg       0.96      0.96      0.96     80354
   macro avg       0.60      0.58      0.59     80354
weighted avg       0.95      0.96      0.95     80354
```

From the classification report it is clear that we have received excellent precision for the records having no purchase made by the players and very bad precision for the records having evidence of making purchase by the players. This is because of the imbalanced dataset.

Naïve Bayes on Resampled data

```
#getting the data
data = df[['playertype','variant','d1','d7','d14','spins','buypageviews']]

#assigning the 'bc' column as target
#because we are checkig which all factors are attracting players to make a purchase
target = df.iloc[:,8]
```

```
#split data set into train and test sets
data_train_up, data_test_up, target_train_up, target_test_up = train_test_split(data_up,target_up, test_size = 0.20, random_s
```

```
#create an object of the type GaussianNB
gnb_up = GaussianNB()
```

```
#train the algorithm on training data and predict using the testing data
pred_up = gnb_up.fit(data_train_up, target_train_up).predict(data_test_up)
```

```
#print the accuracy score of the model
print("Naive-Bayes accuracy_after resampling : ",accuracy_score(target_test_up, pred_up, normalize = True))
```

```
Naive-Bayes accuracy_after resampling :  0.8900493611332494
```

```
print(classification_report(target_test_up, pred_up))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.90     77813
           1       0.98      0.79      0.88     77775

   micro avg       0.89      0.89      0.89    155588
   macro avg       0.91      0.89      0.89    155588
weighted avg       0.91      0.89      0.89    155588
```

Here, we get less accuracy as compared to the accuracy of the original data set. However, we get a better precision and f1-score. The F1-score for both variables, is more than 85% which is a great score. This means that with resampled data set the model is becoming more accurate.

Decision Tree Analysis

Decision trees are capable of fitting complex data sets while allowing the user to see how a decision was taken.

```
# Importing libraries

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.externals import joblib
```

```
#Splitting the data into independent and dependent variables
X = df[['variant','d1','d7','d14','spins','buypageviews']] # independent features set
y = df[['bc']]
```

```
# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1) # 80% training and 30% test
```

```
# Create Decision Tree classifer object
from sklearn.tree import DecisionTreeClassifier


clf = DecisionTreeClassifier(criterion="entropy", max_depth=7)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

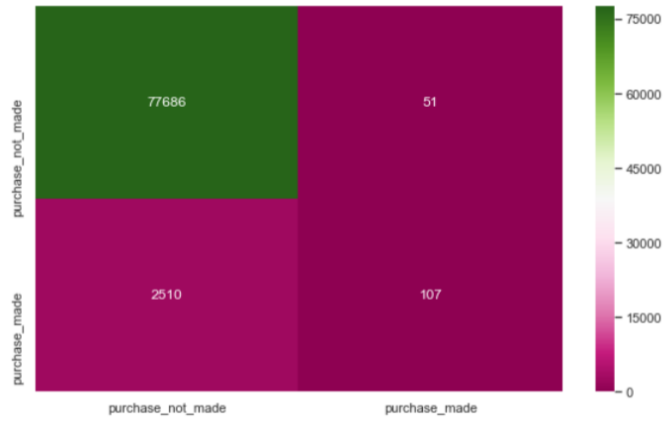
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9681285312492222
```

The accuracy is 96% which great. However, accuracy is not the only factor one should consider understanding the efficiency of the model. Because accuracy for the imbalanced data set could be deceiving. We should other parameters as well. Therefore, I checked the classification matrix as well. We can decide the efficiency of the model using precision, recall and f1-score parameters.

```
# Model confusion matrix, how often is the classifier correct?
cm = confusion_matrix(y_test, y_pred)

index = ['purchase_not_made','purchase_made']
cols1 = ['purchase_not_made','purchase_made']
cm_df = pd.DataFrame(cm,cols1,index)
plt.figure(figsize=(10,6))
sns.heatmap(cm_df, annot=True,cmap='PiYG', fmt='g')


#print(cm)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2942305aa90>
```



I used Seaborn library to visualize the confusion matrix.

There are 107 records which are correctly predicted for the purchase made by the players.

```
#Classification report

print(classification_report(y_test, y_pred))
```

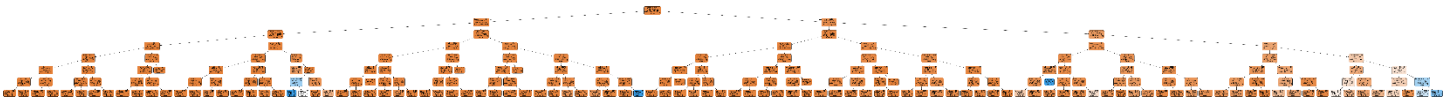
```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     77737
           1       0.68      0.04      0.08      2617

    micro avg       0.97      0.97      0.97     80354
    macro avg       0.82      0.52      0.53     80354
 weighted avg       0.96      0.97      0.95     80354
```

Precision of this model is largely towards bc=0. This makes analysis biased.

We can expect to have biased visualization of this tree. We can expect more of class = 0, representing purchase NOT made by the user.

Therefore, doing analysis on resampled data set.

I completely understand from below image you cannot se the numbers properly. However, the color of the decision leaves tell everything. Most of the part of the decision tree is in orange, indicating class =0. Meaning, we have a biased analysis which will predict bc=0 most of the time.



Decision Tree Analysis on Resampled data

```
#getting the data
data_up_dec = df_upsampled[['variant','d1','d7','d14','spins','buypageviews']]

#assigning the 'variant' column as target because we are checkig which all factors are attracting players to make a purchase
target_up_dec = df_upsampled.iloc[:,8]
```

```
#split data set into train and test sets
data_train_up_dec, data_test_up_dec, target_train_up_dec, target_test_up_dec = train_test_split(data_up_dec,target_up_dec, te
```

```
# Create Decision Tree classifer object

clf_up = DecisionTreeClassifier(criterion="entropy",max_depth=7)

# Train Decision Tree Classifer
clf_up = clf_up.fit(data_train_up_dec,target_train_up_dec)

#Predict the response for test dataset
y_pred_dec = clf_up.predict(data_test_up_dec)
```

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",accuracy_score(target_test_up_dec, y_pred_dec))
```

```
Accuracy: 0.8076651155616115
```

```
print(classification_report(target_test_up_dec, y_pred_dec))
```

```
              precision    recall  f1-score   support

           0       0.83      0.78      0.80     77813
           1       0.79      0.84      0.81     77775

    micro avg       0.81      0.81      0.81    155588
    macro avg       0.81      0.81      0.81    155588
 weighted avg       0.81      0.81      0.81    155588
```

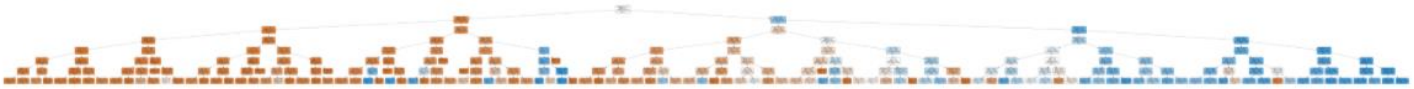
Here, I received less accuracy as compared to the accuracy of the original data set. However, I received a better precision and f1-score. With this resampled data set, precision is not biased. F1-score for both variable, is more than 80% which is a great score. This means that with resampled data set the model is becoming more accurate for prediction.

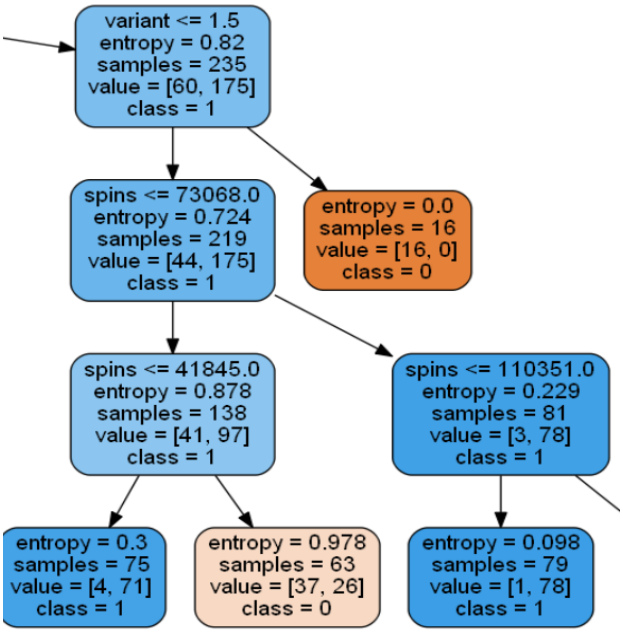
Visualizing the decision tree:

```
dot_data_up = StringIO()

export_graphviz(clf_up, out_file=dot_data_up,  feature_names = col_names, filled=True, rounded=True, class_names=['0','1'])
```

```
graph_up = pydotplus.graph_from_dot_data(dot_data_up.getvalue())
```

```
Image(graph_up.create_png())
```





In this visualization you get to see two colors. I have visualized this decision tree based on entropy parameter. Since, the accuracy, precision and F1-score are better for this decision tree, it is good for prediction.

For re-sampled data, Variant 0 or 1 can be used to predict if the player is going to make purchase or not. It also depends on the number spins.

Along with variant0 or Variant 2, the player should make number of spins greater than 73068 but less than 110351, then only player will possibly make purchase. With entropy of 9.8% or say, with 91% confidence, 78 players out of 79 will possibly make a purchase during their experiment.

With an observation of above decision tree, it is found that the purchase is mostly dependent on the number of spins a player doing on slot machine.

Another observation is that number frequency of coming back after entering the experiment is most likely cause of the purchase. This implicitly tells us that if number of engaging features increases the number of purchases increases.

I even re-sampled the number of variants of each type to check the accuracy of the model.

Taking Variant 2 = 60% of overall Variant1 data and rest of the data will be equally distributed

```
# Sampling class
df_v0_upsampled20 = resample(df_v0,
                                replace=True,     # sample with replacement
                                n_samples=156180,    # to match majority class
                                random_state=123) # reproducible results

df_v1_upsampled20 = resample(df_v1,
                                replace=True,     # sample with replacement
                                n_samples=156180,    # to match majority class
                                random_state=123) # reproducible results

df_v2_upsampled60 = resample(df_v2,
                                replace=True,     # sample with replacement
                                n_samples=468541,    # to match majority class
                                random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled_variants2_6040 = pd.concat([df_v0_upsampled20, df_v1_upsampled20,df_v2_upsampled60])

# Display new class counts
print(df_upsampled_variants2_6040.variant.value_counts())
```

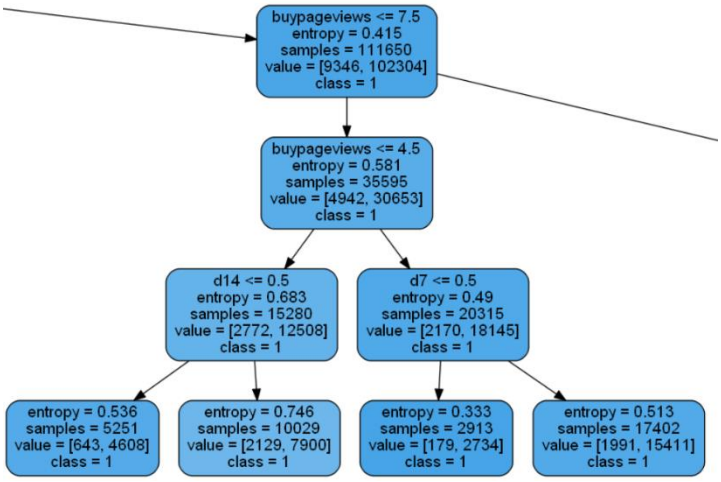
```
2    468541
1    156180
0    156180
Name: variant, dtype: int64
```

```
#getting the data
data_up_dec_var2_6040 = df_upsampled_variants[['variant','d1','d7','d14','spins','buypageviews']]

#assigning the 'variant' column as target because we are checkig which all factors are attracting players to make a purchase
target_up_dec_var2_6040 = df_upsampled_variants.iloc[:,8]

#split data set into train and test sets
data_train_up_dec_v2_6040, data_test_up_dec_v2_6040, target_train_up_dec_v2_6040, target_test_up_dec_v2_6040 = train_test_spl
```

```
# Create Decision Tree classifer object

clf_up_var2_6040 = DecisionTreeClassifier(criterion="entropy",max_depth=7)

# Train Decision Tree Classifer
clf_up_var2_6040 = clf_up_var2_6040.fit(data_train_up_dec_v2_6040,target_train_up_dec_v2_6040)

#Predict the response for test dataset
y_pred_dec_v2_6040 = clf_up_var2_6040.predict(data_test_up_dec_v2_6040)
```

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy: after resampling Variants",accuracy_score(target_test_up_dec_v2_6040, y_pred_dec_v2_6040))

print(classification_report(target_test_up_dec_v2_6040, y_pred_dec_v2_6040))
```

```
Accuracy: after resampling Variants 0.8066474154986842
              precision    recall  f1-score   support

           0       0.83      0.77      0.80     78116
           1       0.79      0.84      0.81     78065

   micro avg       0.81      0.81      0.81    156181
   macro avg       0.81      0.81      0.81    156181
weighted avg       0.81      0.81      0.81    156181
```



Irrespective of the variant, a player will mostly likely (73%) make a purchase if he/she visits the Buy Page for 5 to 7 times in week from the day the player has entered the experiment. However, a good spin of more than 8,946 is required.

Conclusion:

		Original Imbalanced Data Set	Re-sampled Data Set
Naïve Beys	**Accuracy**	95.52%	89%
	Precision	0.97|0.24	0.83|0.98
	F1-score	0.98|0.20	0.90|0.88
Decision tree	**Accuracy**	96.12%	80.76%
	Precision	0.97|0.68	0.83|0.79
	F1-score	0.98|0.08	0.80|0.81

Both models are giving great accuracy and after re-sampling both are giving great precision to predict the purchase made by the player during experiment.

I also observed that type of variant is not only factor to have a purchase from the player, rather keeping them engaged would be beneficial.

Recommendations:

1. Engaging factors such as Spins and heartbeats are required to acquire and sustain existing of players.
2. Adding more engaging features would be beneficial.
3. Rewarding existing players would be beneficial to keep them engaged and attracted towards the game.