In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
NYC = pd.read_csv('nyc_historical.csv')
```

# A. Bring the dataset into your environment, and use the head() function to explore the variables.

In [2]:
```
NYC.head()
```

Out[2]:

| | householdID | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perperson | avgfoo |
|---|---|---|---|---|---|---|
| **0** | 44 | 20 | 9.8 | 32.4 | 27.2 | |
| **1** | 57 | 20 | 11.7 | 71.8 | 40.8 | |
| **2** | 63 | 20 | 9.8 | 27.4 | 25.7 | |
| **3** | 159 | 17 | 2.2 | 1.5 | 91.1 | |
| **4** | 162 | 19 | 3.4 | 5.0 | 12.0 | |

# B. Which of the variables here are categorical? Which are numerical?

visits, avgrides_perperson,avgmerch_perperson,avggoldzone_perperson, and avgfood_perperson are numerical. goldzone_playersclub,own_car, homestate,FB_Like,renew,and householdID are categorical.

In [3]:
```
NYC['renew'].value_counts()
```

Out[3]:
```
1    2126
0    1074
Name: renew, dtype: int64
```

In [4]:
```
NYC['renew'].value_counts(normalize=True)
```

Out[4]:
```
1    0.664375
0    0.335625
Name: renew, dtype: float64
```

# C.what are the different outcome classes here, and how common are each of them in the dataset? What was different here about the second time you ran this function?

The first "value_counts" counts the amount of 1s and 2s in "renew",1 appears 2126 times,meaning 2126 people renew their membership and 0 appears 1074 times, meaning 1074 people doesn't renew the membership. The second counts the percentage of each one. 66.4375% of people renew their membership, and 33.5625% of people does not.

# D. For your categorical input variables, do you need to take any steps to convert them into dummies, in order to build a logistic regression model? Why or why not?

No, as all category variables are binary in nature. Dummify Household IDs are pointless, as each ID represents a category. Home states are difficult to dummify and are best substituted by latitudes and longitudes; however, because we only have three home states in this case, those who live outside of the three cannot be accounted, it can hurt our estimates.

In [5]:

```
NYC.shape
```

Out[5]: (3200, 11)

In [6]:

```
NYC.head()
```

Out[6]:

| | householdID | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perperson | avgfoc |
|---|---|---|---|---|---|---|
| **0** | 44 | 20 | 9.8 | 32.4 | 27.2 | |
| **1** | 57 | 20 | 11.7 | 71.8 | 40.8 | |
| **2** | 63 | 20 | 9.8 | 27.4 | 25.7 | |
| **3** | 159 | 17 | 2.2 | 1.5 | 91.1 | |
| **4** | 162 | 19 | 3.4 | 5.0 | 12.0 | |

In [7]:
```
NYC.describe()
```

Out[7]:

| | householdID | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perpers |
|---|---|---|---|---|---|
| **count** | 3200.000000 | 3200.000000 | 3200.000000 | 3200.000000 | 3200.000( |
| **mean** | 1600.500000 | 6.691562 | 9.484344 | 35.007719 | 79.9017 |
| **std** | 923.904757 | 6.198614 | 2.813355 | 23.928679 | 48.0428 |
| **min** | 1.000000 | 1.000000 | 0.100000 | 0.100000 | 0.100( |
| **25%** | 800.750000 | 2.000000 | 8.400000 | 16.600000 | 40.500( |
| **50%** | 1600.500000 | 4.000000 | 10.000000 | 29.800000 | 73.600( |
| **75%** | 2400.250000 | 7.000000 | 11.400000 | 48.725000 | 116.525( |
| **max** | 3200.000000 | 20.000000 | 13.400000 | 89.100000 | 172.800( |

In [8]:
```
NYC=NYC.drop(columns='householdID')
NYC=NYC.drop(columns='homestate')
```

In [9]:
```
NYC.describe()
```

Out[9]:

| | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perperson | avgfood_ |
|---|---|---|---|---|---|
| **count** | 3200.000000 | 3200.000000 | 3200.000000 | 3200.000000 | 32 |
| **mean** | 6.691562 | 9.484344 | 35.007719 | 79.901781 | |
| **std** | 6.198614 | 2.813355 | 23.928679 | 48.042896 | |
| **min** | 1.000000 | 0.100000 | 0.100000 | 0.100000 | |
| **25%** | 2.000000 | 8.400000 | 16.600000 | 40.500000 | |
| **50%** | 4.000000 | 10.000000 | 29.800000 | 73.600000 | |
| **75%** | 7.000000 | 11.400000 | 48.725000 | 116.525000 | |
| **max** | 20.000000 | 13.400000 | 89.100000 | 172.800000 | |

In [10]:
```
NYC.columns
```

Out[10]:
```
Index(['visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'FB_Like', 'renew'],
      dtype='object')
```

In [11]:
```
t=NYC[['visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'FB_Like', 'renew']]
```

In [12]:
```
t.corr()
```

Out[12]:

| | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_per |
|---|---|---|---|---|
| **visits** | 1.000000 | 0.007144 | -0.001957 | 0.( |
| **avgrides_perperson** | 0.007144 | 1.000000 | -0.037131 | 0.( |
| **avgmerch_perperson** | -0.001957 | -0.037131 | 1.000000 | -0.( |
| **avggoldzone_perperson** | 0.014787 | 0.002382 | -0.013894 | 1.( |
| **avgfood_perperson** | 0.011928 | -0.012097 | -0.007268 | -0.( |
| **goldzone_playersclub** | -0.002978 | -0.011219 | 0.017161 | -0.( |
| **own_car** | 0.008785 | -0.001298 | -0.016104 | 0.( |
| **FB_Like** | -0.004384 | -0.015478 | -0.006174 | -0.( |
| **renew** | 0.271513 | 0.068153 | 0.037206 | 0.( |

In [ ]:
```

```

# E. Determine the correlations among your potential input variables. If any correlations appear to be very high, remove one variable from any highly-correlated pair.

The highest corelation is between visits and renew, and i believe none of the variables should be droped.

In [13]:
```
x=NYC[['visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'FB_Like']]
y=NYC['renew']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.6,random_state=600
```

# F.a. How did you pick your seed value?

6 is a good number in China, it means everything will be alright.

In [14]:
```
x.corr()
```

Out[14]:

| | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perper |
|---|---|---|---|---|
| **visits** | 1.000000 | 0.007144 | -0.001957 | 0.014 |
| **avgrides_perperson** | 0.007144 | 1.000000 | -0.037131 | 0.002 |
| **avgmerch_perperson** | -0.001957 | -0.037131 | 1.000000 | -0.013 |
| **avggoldzone_perperson** | 0.014787 | 0.002382 | -0.013894 | 1.000 |
| **avgfood_perperson** | 0.011928 | -0.012097 | -0.007268 | -0.003 |
| **goldzone_playersclub** | -0.002978 | -0.011219 | 0.017161 | -0.007 |
| **own_car** | 0.008785 | -0.001298 | -0.016104 | 0.018 |
| **FB_Like** | -0.004384 | -0.015478 | -0.006174 | -0.035 |

# G. Build a logistic regression model using Python

In [15]:
```
logmodel = LogisticRegression()
logmodel.fit(x_train, y_train)
predictions=logmodel.predict(x_test)
accuracy_score(y_test,predictions)
```

```
C:\Users\41223\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.
org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (ht
tps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[15]:
```
0.696875
```

In [16]:
```
logmodel.intercept_
```

Out[16]:
```
array([-1.73343698])
```

In [17]:
```
logmodel.coef_
```

Out[17]:
```
array([[ 1.24343555e-01,  6.13551322e-02,  7.01610062e-03,
         2.90850568e-03,  3.18347802e-04,  3.20084648e-01,
         7.91766292e-01, -1.95298044e-01]])
```

In [18]:

```
pd.DataFrame(data=logmodel.coef_.transpose(),columns=['Coef'])
```

Out[18]:

| | Coef |
|---|---|
| 0 | 0.124344 |
| 1 | 0.061355 |
| 2 | 0.007016 |
| 3 | 0.002909 |
| 4 | 0.000318 |
| 5 | 0.320085 |
| 6 | 0.791766 |
| 7 | -0.195298 |

# a. Which of your numeric variables appear to influence the outcome variable the most? Write a paragraph for Lobster Land management that indicates the direction, and strength, of the impact that these numeric variables have on the outcome. For each one, speculate a bit (one sentence is okay) about why it might be impacting the model this way

"Own car" has the greatest influence on the outcome. Consumers will consider renewing their memberships only when they get the mobility to visit the park more frequently. For members who do not own a car, we may plan shuttle buses to transport them back and forth, which will encourage more "non-vehicle" members to renew their memberships more frequently.

Visits also have an effect on the outcome, since the more visits consumers make, the more likely they are to be lobster land fans, which means they will continue to visit lobster land in the future, and so renew their membership. However, the majority of individuals elect not to renew. The explanation for this may be that they have visited lobster land so frequently that they have grown weary of it and have decided to cancel their memberships. The explanation might also possibly be that people believe memberships do not add enough value to them and hence decline to renew them. For existing renewing members, we should provide additional value to ensure a greater retention rate. We can establish membership lounges across the park where members can consume snacks and relax without being disturbed by crowds. We may also send them holiday presents; they do not have to be extravagant, but they should feel our appreciation. For individuals who did not renew their membership due to our low-value activities, we should determine what we did wrong and make it right. To entice visitors who have grown bored of our park, we should continue to update rides and events; perhaps we might invite bands to perform at the park, such as Guns & Roses.

The average number of rides taken and the average amount spent on items also have a positive effect on the outcome. Customers who like the park's attractions and merchandise may return in the future and may consider renewing their membership,despite the fact that the correlation is small. The most of consumers would not renew their membership because they ride a lot of rides or spend a lot of money at the park's merchandise. Gold Zone spending also has positive influence to the outcome, but it has the least postive influence. The awesomeness of the Gold zone may bring customers to renew their membership, but people do not renew their membership just because of the Gold zone.

Participants in the Gold Zone Players' Club are die-hard fans of the Gold Zone, and they can only access the club after visiting the park. If they want to join the Gold Zone Players' Club, they will be required to visit the park more frequently, necessitating the renewal of their membership.

Average food expenditure has the least positive effect on output. no matter food is expensive or not, people are not making decision of renewing membership based on food.

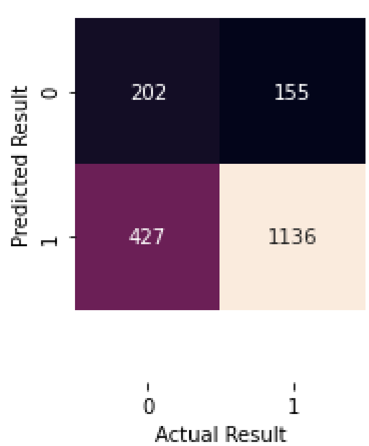
The number of "likes" on a Facebook page has a significant negative effect on the outcome. Most people do not like lobsterland facebook page. We need to revamp our page and hire a professional to manage the account.

In [ ]:

```
 
```

In [19]:

```
mat = confusion_matrix(predictions, y_test)
sns.heatmap(mat, square=True, fmt = 'g', annot=True, cbar=False)
plt.xlabel("Actual Result")
plt.ylabel("Predicted Result")
a, b = plt.ylim()
a += 0.5
b -= 0.5
plt.ylim(a, b)
plt.show()
```




In [20]:

```
accuracy_score(y_test, predictions)
```

Out[20]: 0.696875

a. What is your model’s accuracy rate?

0.696875

In [21]:
```
sensitivity=1136/(1136+155)
sensitivity
```

Out[21]: 0.8799380325329202

b. What is your model’s sensitivity rate?

0.8799380325329202

In [22]:
```
speficity=202/(202+427)
speficity
```

Out[22]: 0.32114467408585057

c. What is your model’s specificity rate?

0.32114467408585057

In [23]:
```
precision=1136/(1136+427)
precision
```

Out[23]: 0.7268074216250799

d. What is your model’s precision?

0.7268074216250799

In [24]:
```
Balanced_accuracy=(sensitivity+speficity)/2
Balanced_accuracy
```

Out[24]: 0.6005413533093854

e. What is your model’s balanced accuracy?

0.6005413533093854

In [25]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.57      0.32      0.41       629
           1       0.73      0.88      0.80      1291

    accuracy                           0.70      1920
   macro avg       0.65      0.60      0.60      1920
weighted avg       0.67      0.70      0.67      1920
```

In [26]:
```
preds_train = logmodel.predict(x_train)
accuracy_score(y_train,preds_train)
```

Out[26]: 0.6859375

In [27]:
```
preds_test = logmodel.predict(x_test)
accuracy_score(y_test,preds_test)
```

Out[27]: 0.696875

a. What is the purpose of comparing those two values? b. In this case, what does the comparison of those values suggest about the model that you have built?

The method's objective is to generalize the trend in the data, and we want a model to predict the data we never seen before. If two figures are significantly different, this indicates that the model was well-built only for the data used to build it and is not suitable for any other data. In this case, the model is perfect, and it is not overfitting the current data, it is suitable for predicting new data.

In [28]:
```
NYC.head()
```

Out[28]:

| | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perperson | avgfood_perperson |
|---|---|---|---|---|---|
| **0** | 20 | 9.8 | 32.4 | 27.2 | 70.7 |
| **1** | 20 | 11.7 | 71.8 | 40.8 | 1.6 |
| **2** | 20 | 9.8 | 27.4 | 25.7 | 74.9 |
| **3** | 17 | 2.2 | 1.5 | 91.1 | 28.9 |
| **4** | 19 | 3.4 | 5.0 | 12.0 | 9.2 |

In [29]:
```
NYC.columns
```

Out[29]:
```
Index(['visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'FB_Like', 'renew'],
      dtype='object')
```

# K. Make up a household.

In [30]:
```
Di_Ha = pd.DataFrame([{'visits':17, 'avgrides_perperson':36, 'avgmerch_perperson'
        'avggoldzone_perperson':24, 'avgfood_perperson':68, 'goldzone_playersclub'
        'own_car':1, 'FB_Like':1}])

newprediction = logmodel.predict(Di_Ha)
newprediction

```

Out[30]: array([1], dtype=int64)

a. What did your model predict -- will this household renew?

Yes

In [31]:
```
logmodel.predict_proba(Di_Ha)
```

Out[31]: array([[0.01932295, 0.98067705]])

b. According to your model, what is the probability that this household will renew?

98.07%

In [32]:
```
Di_Hahahaha = pd.DataFrame([{'visits':859, 'avgrides_perperson':582, 'avgmerch_pe
       'avggoldzone_perperson':9865, 'avgfood_perperson':4668, 'goldzone_playersc
       'own_car':7, 'FB_Like':9}])

newprediction = logmodel.predict(Di_Hahahaha)
newprediction
```

Out[32]: array([1], dtype=int64)

In [33]:
```
logmodel.predict_proba(Di_Hahahaha)
```

Out[33]: array([[0., 1.]])

Di_Hahahaha is lobsterland ghost,who lives in a time loop and can never get out. The probability of him being a membership is 100%. This caused by outrange input number, the outcome can only have extreme probability 0 or 1.

# Part II: Random Forest Model

In [58]:
```
RFM = pd.read_csv('nyc_historical.csv')
```

In [59]:
```
RFM.head()
```

Out[59]:

| | householdID | visits | avgrides_perperson | avgmerch_perperson | avggoldzone_perperson | avgfoc |
|---|---|---|---|---|---|---|
| **0** | 44 | 20 | 9.8 | 32.4 | 27.2 | |
| **1** | 57 | 20 | 11.7 | 71.8 | 40.8 | |
| **2** | 63 | 20 | 9.8 | 27.4 | 25.7 | |
| **3** | 159 | 17 | 2.2 | 1.5 | 91.1 | |
| **4** | 162 | 19 | 3.4 | 5.0 | 12.0 | |

In [60]:
```
RFM.columns
```

Out[60]:
```
Index(['householdID', 'visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'homestate', 'FB_Like', 'renew'],
      dtype='object')
```

In [61]:
```
RFM=pd.get_dummies(RFM,columns=['homestate'])
RFM
```

Out[61]:

| vgmerch_perperson | avggoldzone_perperson | avgfood_perperson | goldzone_playersclub | own_car | Fl |
|---|---|---|---|---|---|
| 32.4 | 27.2 | 70.7 | 0 | 1 | |
| 71.8 | 40.8 | 1.6 | 0 | 1 | |
| 27.4 | 25.7 | 74.9 | 0 | 1 | |
| 1.5 | 91.1 | 28.9 | 1 | 1 | |
| 5.0 | 12.0 | 9.2 | 0 | 1 | |
| ... | ... | ... | ... | ... | |
| 37.7 | 72.1 | 17.1 | 0 | 1 | |
| 29.3 | 85.7 | 8.9 | 0 | 1 | |
| 49.3 | 16.4 | 30.3 | 0 | 1 | |
| 21.2 | 85.7 | 38.4 | 0 | 0 | |
| 39.4 | 88.3 | 1.7 | 0 | 1 | |

In [62]:
```
RFM.columns
```

Out[62]:
```
Index(['householdID', 'visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'FB_Like', 'renew', 'homestate_CT', 'homestate_NJ',
       'homestate_NY'],
      dtype='object')
```

In [63]:
```
x=RFM[['householdID', 'visits', 'avgrides_perperson', 'avgmerch_perperson',
       'avggoldzone_perperson', 'avgfood_perperson', 'goldzone_playersclub',
       'own_car', 'FB_Like',  'homestate_CT','homestate_NJ', 'homestate_NY']]
y=RFM['renew']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.6,random_state=600
```

In [ ]:
```

```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier()
clf.fit(x_train,y_train)
clf
```

In [77]:
```
param_grid1= {
 'n_estimators': [50, 100, 150],
 'max_depth': [2, 4, 6, 8],
 'max_features': [1, 2, 3, 4, 5],
 'min_samples_leaf':[2,4,6,10]
}

```

In [78]:
```
from sklearn.model_selection import GridSearchCV
CV_rfc = GridSearchCV(estimator=clf, param_grid=param_grid1, cv= 5)
CV_rfc.fit(x_train, y_train)
print(CV_rfc.best_params_)

```

```
{'max_depth': 6, 'max_features': 5, 'min_samples_leaf': 6, 'n_estimators': 50}
```

In [79]:
```
clf=RandomForestClassifier(
    n_estimators=50, max_depth=6, max_features=5,min_samples_leaf=6, random_state
clf.fit(x_train,y_train)
```

Out[79]:
```
RandomForestClassifier(max_depth=6, max_features=5, min_samples_leaf=6,
                       n_estimators=50, random_state=600)
```

In [80]:
```
feature_imp_df = pd.DataFrame(list(zip(clf.feature_importances_, x_train)))
feature_imp_df.columns = ['feature importance', 'feature']
feature_imp_df = feature_imp_df.sort_values(by='feature importance', ascending=Fa
feature_imp_df
```

Out[80]:

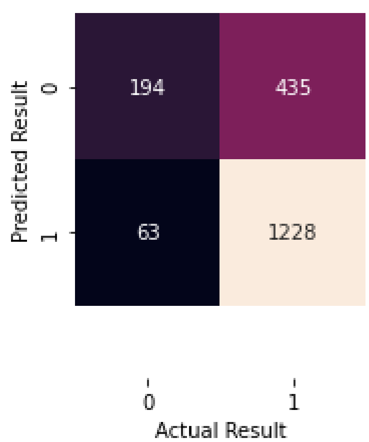
| | feature importance | feature |
|---|---|---|
| 1 | 0.301526 | visits |
| 10 | 0.123032 | homestate_NJ |
| 2 | 0.098273 | avgrides_perperson |
| 3 | 0.095067 | avgmerch_perperson |
| 7 | 0.085599 | own_car |
| 4 | 0.080133 | avggoldzone_perperson |
| 5 | 0.072787 | avgfood_perperson |
| 0 | 0.066703 | householdID |
| 9 | 0.028832 | homestate_CT |
| 11 | 0.020927 | homestate_NY |
| 6 | 0.018579 | goldzone_playersclub |
| 8 | 0.008544 | FB_Like |

# How did your random forest model rank the variables in order of importance,from highest to

# lowest? For a random forest model, how can you interpretfeature importance?

The table rates the variables in ascending order of relevance, with the top row being the most significant and the bottom row being the least significant. The sum of all feature values should equal one. Visits have the most impact on membership renewing. Maybe NJ is close to Lobersterland, so it affects membership renewing, and it is the second most varible affecting the outcome. People do not like our facebook, which makes the FB like the least important varible.

In [81]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
predictions = clf.predict(x_test)
mat = confusion_matrix(y_test, predictions)
sns.heatmap(mat, fmt='g', square=True, annot=True, cbar=False)
plt.xlabel("Actual Result")
plt.ylabel("Predicted Result")
a, b = plt.ylim()
a += 0.5
b -= 0.5
plt.ylim(a, b)
plt.show()
```




In [82]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
```

Out[82]: 0.740625

In [84]:
```

```

In [86]:
```
sensitivity=1228/(1228+435)
sensitivity
```

Out[86]: 0.7384245339747444

sensitivity is 0.7384245339747444

In [87]:
```
speficity=194/(63+194)
speficity
```
Out[87]: 0.754863813229572

speficity is 0.754863813229572

In [88]:
```
precision=1228/(1228+63)
precision
```
Out[88]: 0.9512006196746708

precision is 0.9512006196746708

In [89]:
```
balanced_accuracy=(sensitivity+speficity)/2
balanced_accuracy
```
Out[89]: 0.7466441736021582

balanced_accuracy is 0.7466441736021582

In [ ]:
```

```

In [102]:
```
logmodel = LogisticRegression()
logmodel.fit(x_train,y_train)
preds_train = logmodel.predict(x_train)
accuracy_score(y_train,preds_train)
```
```
C:\Users\41223\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```
Out[102]: 0.690625

In [103]:
```
preds_train = logmodel.predict(x_test)
accuracy_score(y_test,preds_test)
```
Out[103]: 0.696875

The results are similar, the model can be used for predicting new data.

In [ ]:
```

```

In [105]:
```
Di_Ha = pd.DataFrame([{'householdID':3555,'visits':17, 'avgrides_perperson':36, '
        'avggoldzone_perperson':24, 'avgfood_perperson':68, 'goldzone_playersclub'
        'own_car':1, 'FB_Like':1,'homestate_CT':1,'homestate_NJ':0, 'homestate_NY'

newprediction = logmodel.predict(Di_Ha)
newprediction
```

Out[105]:
```
array([1], dtype=int64)
```

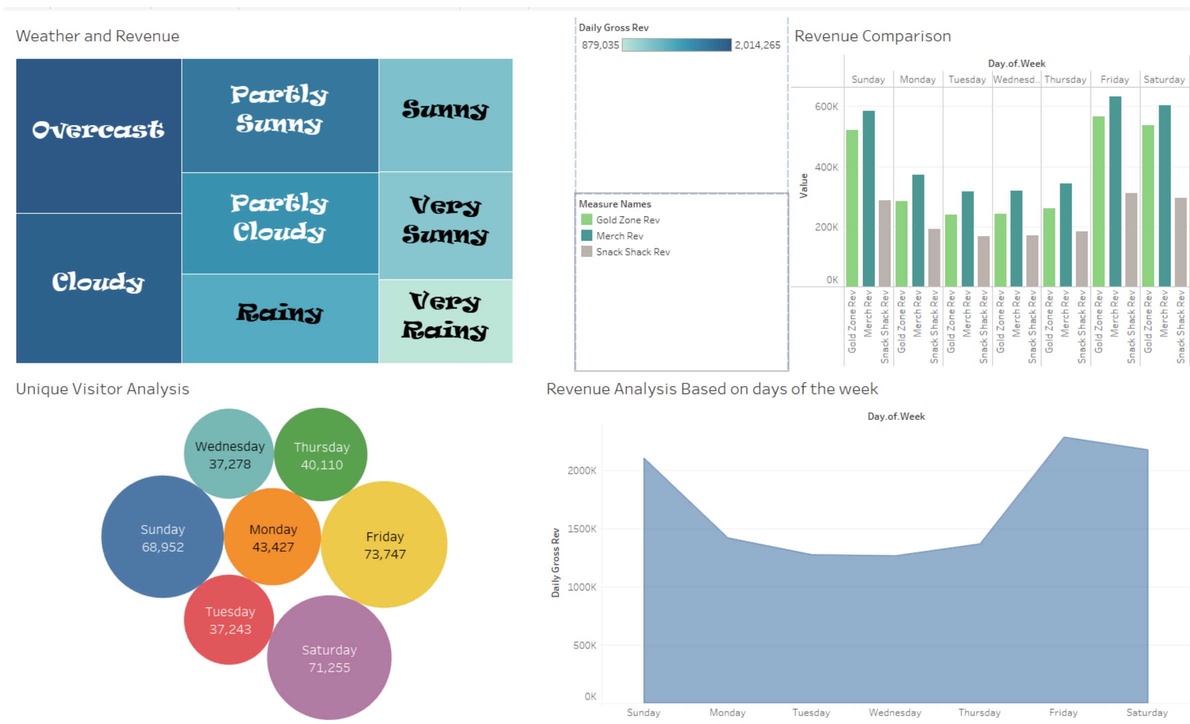
Yes, the model think Di_Ha will renew

Lobsterland makes use of the technology to forecast future clients. By analyzing our clients, we can target the appropriate customer segments and enhance conversion rates, rather than sending advertisements to random people and receiving no response. Additionally, each time a new customer visits the park, we can forecast if the client will join membership or not, therefore adding value to the customer and increasing conversion rate. By comprehending our clients, we can also provide value to existing customers, identify areas for improvement, and increase conversion rates.

# Part III

In [108]:
```
from IPython.display import Image
Image("WeChat Screenshot_20211024201436.png")
```

Out[108]:




I created a comparison between weather and overall revenue using Tree map. The outcome is the total opposite of what I anticipated. I assumed that the sunnier the weather, the more revenue Lobsterland might earn, but the results indicate otherwise. We get the greatest income when the weather is overcast, and we earn the least revenue when the weather is really sunny.

Additionally, I developed side-by-side bars to compare gold zone, merchandise, and snack shack earnings by day of the week. I discovered that merchandise generates more revenue than the other two, and that sanck sahck generates the least revenue. Friday, Saturday, and Sunday produce significantly more money than weekdays and Friday alone. In columns, I list the days of the week and the measure's name; in rows, I provide the measure's value.

Then, I utilized bubble maps to compare the number of unique visitors on various weekdays. Friday has the most unique visitors, and the weekend has more than twice the amount of unique visitors compared to weekdays. Unique visitors are more likely to visit lobsterland during weekend.

For the final graphic, I utilized an area chart to depict the gross income for seven week days. Friday generates the most revenue, while Tuesday generates the least. This is because many tourists come to lobsterland on weekends and spend money, but on Tuesdays, most people are at work and cannot visit the park. For this dragram, I put days of week in columns and sum of daily revenue in rows.

In [ ]:

```

```