Predicting the likelihood of the outcome

November 20, 2020

[356]:
```
#Description: I have implemented Random Forest Classifier. The Outcome classes␣
↪are imbalanced, which I found out
#after running a few statistical analysis. Therefore, I balanced the Data using␣
↪SMOTE algorithm and ran the Random
#Forest Classifier which has the followwing characteristics:

#The Auc Value: 0.853323949609402

#Training accuracy: 0.7487827145465612
#Testing accuracy: 0.7373612823674476

#Recall Score: 0.8137651821862348
```

[357]:
```
#Reading the data and displaying the first five observations

import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
df = pd.read_csv('/Users/harshith/Downloads/data/Train.csv')
df.head(5)
```

[357]:
```
   age  cost_of_ad device_type gender  in_initial_launch_location  income  \
0   56    0.005737      iPhone      M                           0   62717
1   50    0.004733     desktop      F                           0   64328
2   54    0.004129      laptop      M                           0   83439
3   16    0.005117     Android      F                           0   30110
4   37    0.003635     desktop      M                           0   76565

   n_drivers  n_vehicles  prior_ins_tenure  outcome
0          2           1                 4        0
1          2           3                 2        0
2          1           3                 7        0
3          2           3                 0        0
4          2           1                 5        0
```

[358]:
```python
#Displaying the last five observations of the data

df.tail(5)
```

[358]:
```
      age  cost_of_ad device_type gender  in_initial_launch_location  income  \
9995   41    0.004225     desktop      M                           0   64489
9996   50    0.004751       other      F                           0   88643
9997   60    0.003804       other      M                           0   87870
9998   18    0.003838      laptop      M                           0   56468
9999   33    0.005250      iPhone    NaN                           0   59935

      n_drivers  n_vehicles  prior_ins_tenure  outcome
9995          2           3                 8        0
9996          1           3                 0        0
9997          2           2                 9        0
9998          2           2                 0        0
9999          2           1                 6        0
```

[359]:
```python
#Knowing the data type of the attributes

df.dtypes
```

[359]:
```
age                             int64
cost_of_ad                    float64
device_type                    object
gender                         object
in_initial_launch_location      int64
income                          int64
n_drivers                       int64
n_vehicles                      int64
prior_ins_tenure                int64
outcome                         int64
dtype: object
```

[360]:
```python
# Printing the shape of the data and finding out if the data contains any␣
↪duplicate values

print(df.shape)
duplicate_rows_df = df[df.duplicated()]
print('number of duplicate rows:', duplicate_rows_df.shape)
```

```
(10000, 10)
number of duplicate rows: (0, 10)
```

[361]:
```python
#Printing the null values according to each attribute, we find that there are␣
↪269 NULL values in gender
```

```
print(df.isnull().sum())
```

```
age                           0
cost_of_ad                    0
device_type                   0
gender                      269
in_initial_launch_location    0
income                        0
n_drivers                     0
n_vehicles                    0
prior_ins_tenure              0
outcome                       0
dtype: int64
```

[362]:
```
#Dropping the NULL values from the dataset. Imputing values for categorical␣
↪attribute might not be a good approach

df = df.dropna()
```

[363]:
```
# Printing the shape of the data set after dropping the NULL values

df.shape
```

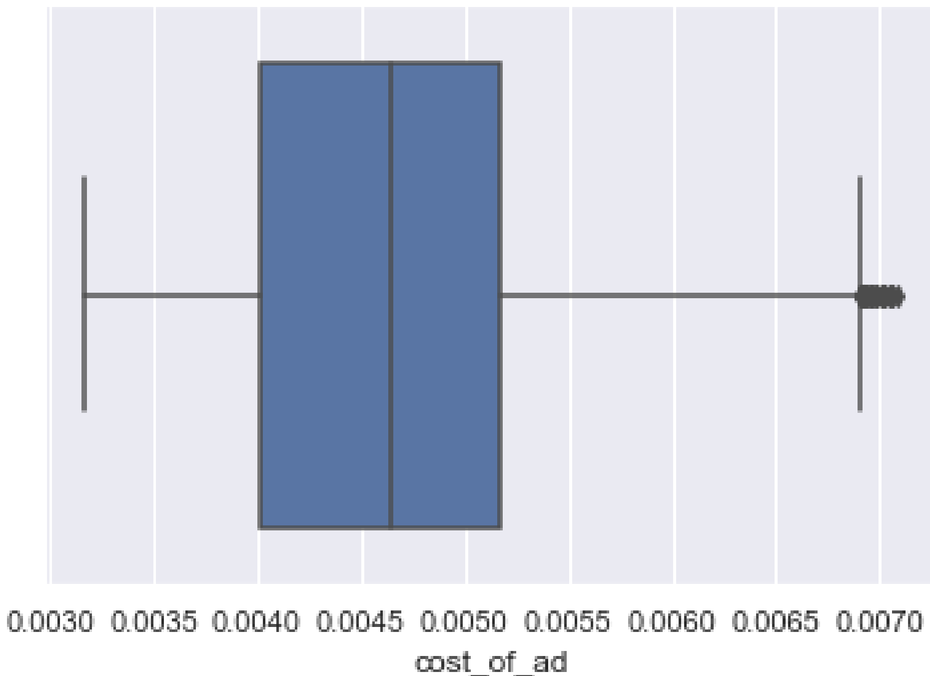
[363]: (9731, 10)

[364]:
```
#Finding out if there are any NULL values after dropping

print(df.isnull().sum())
```

```
age                           0
cost_of_ad                    0
device_type                   0
gender                        0
in_initial_launch_location    0
income                        0
n_drivers                     0
n_vehicles                    0
prior_ins_tenure              0
outcome                       0
dtype: int64
```

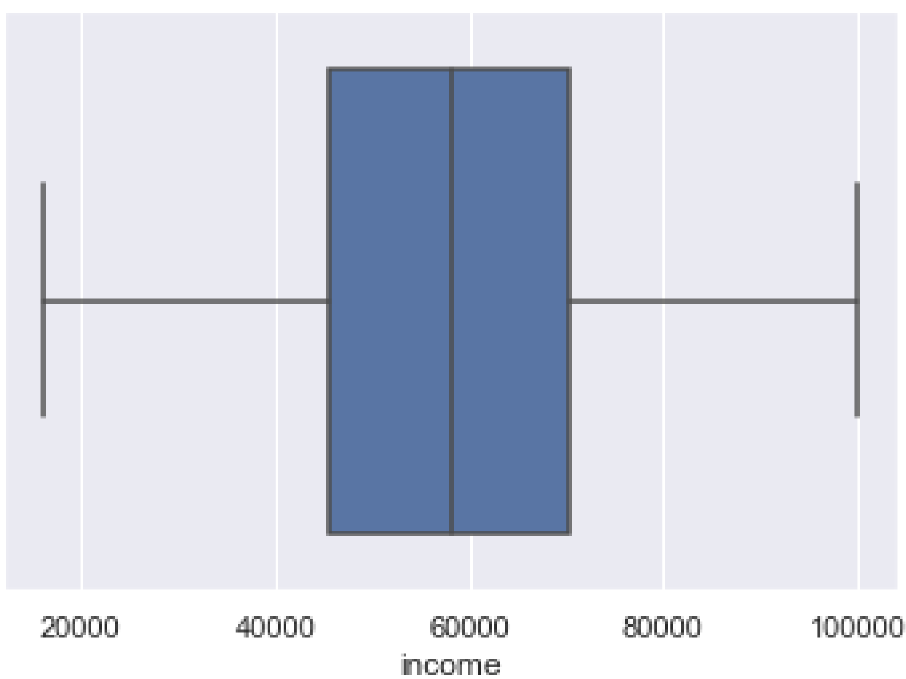
[365]:
```
# Using box plots to find the outliers according to each attribute

sns.boxplot(x=df['cost_of_ad'])
```

[365]: <matplotlib.axes._subplots.AxesSubplot at 0x1a39421ad0>



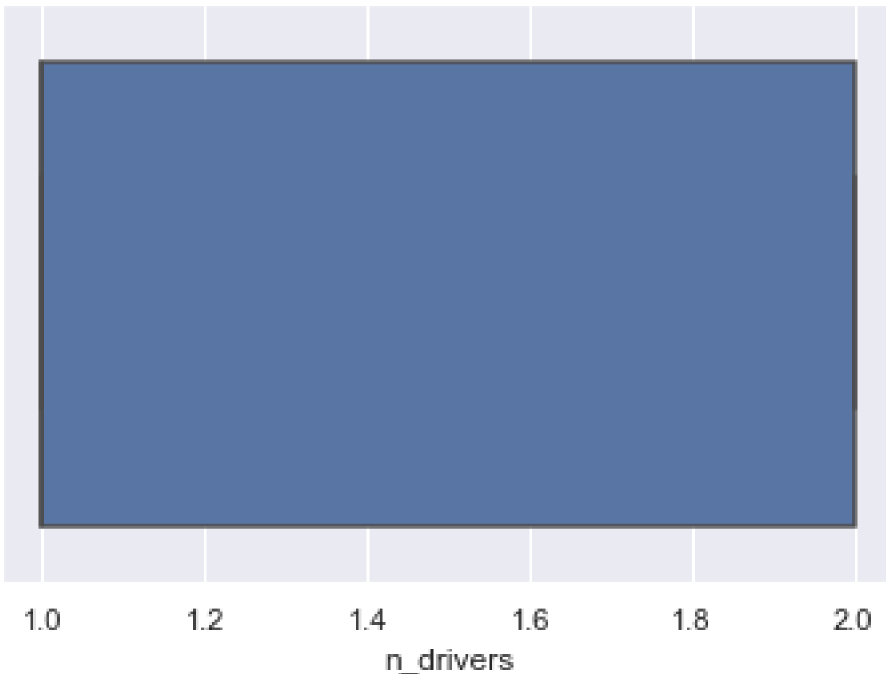
[366]:
```
sns.boxplot(x=df['income'])
```

[366]: <matplotlib.axes._subplots.AxesSubplot at 0x1a46812950>



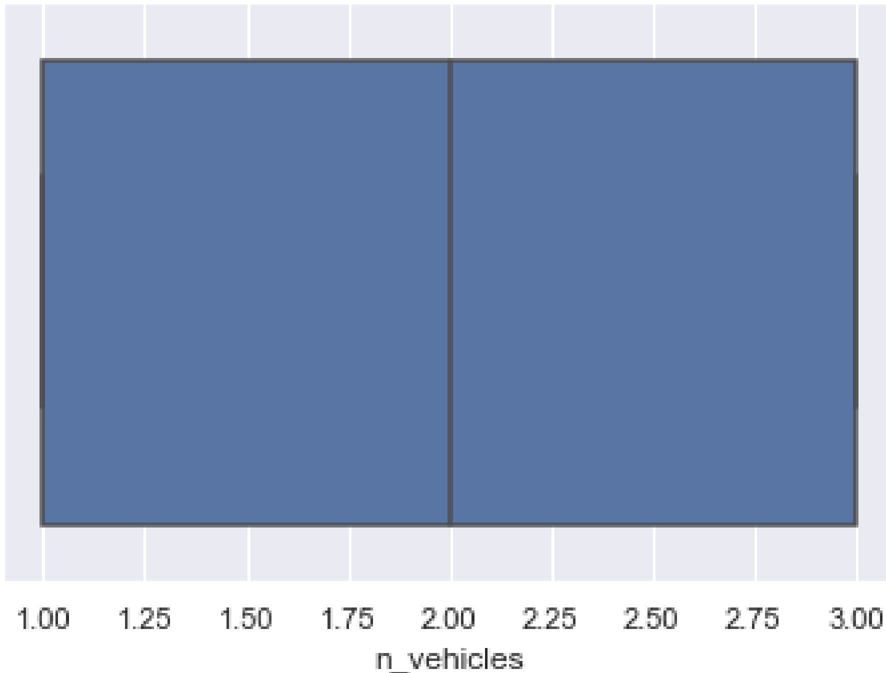
[367]:
```
sns.boxplot(x=df['n_drivers'])
```

[367]: <matplotlib.axes._subplots.AxesSubplot at 0x1a473d6250>



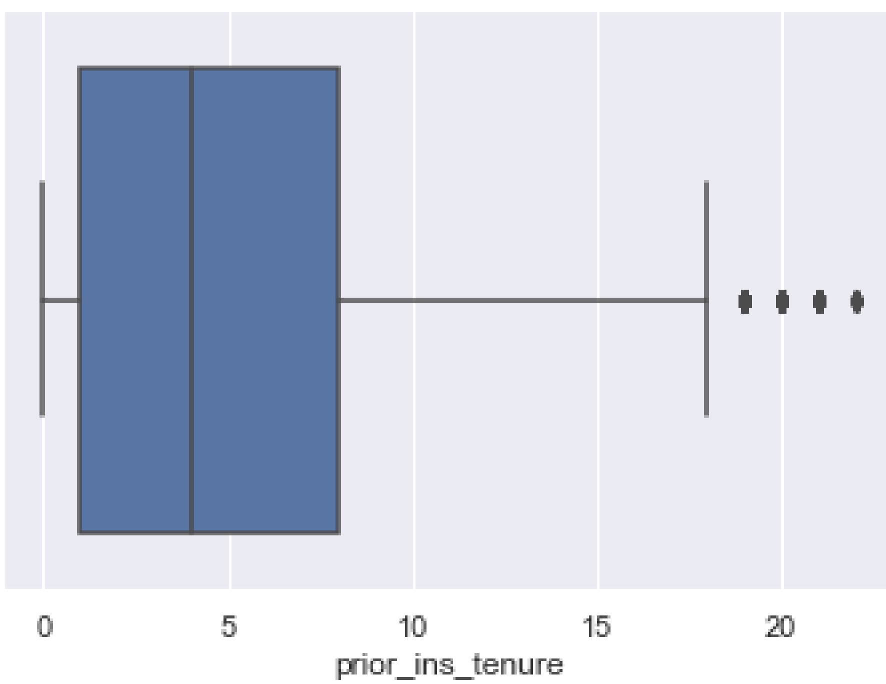
[368]:
```
sns.boxplot(x=df['n_vehicles'])
```

[368]: <matplotlib.axes._subplots.AxesSubplot at 0x1a478032d0>



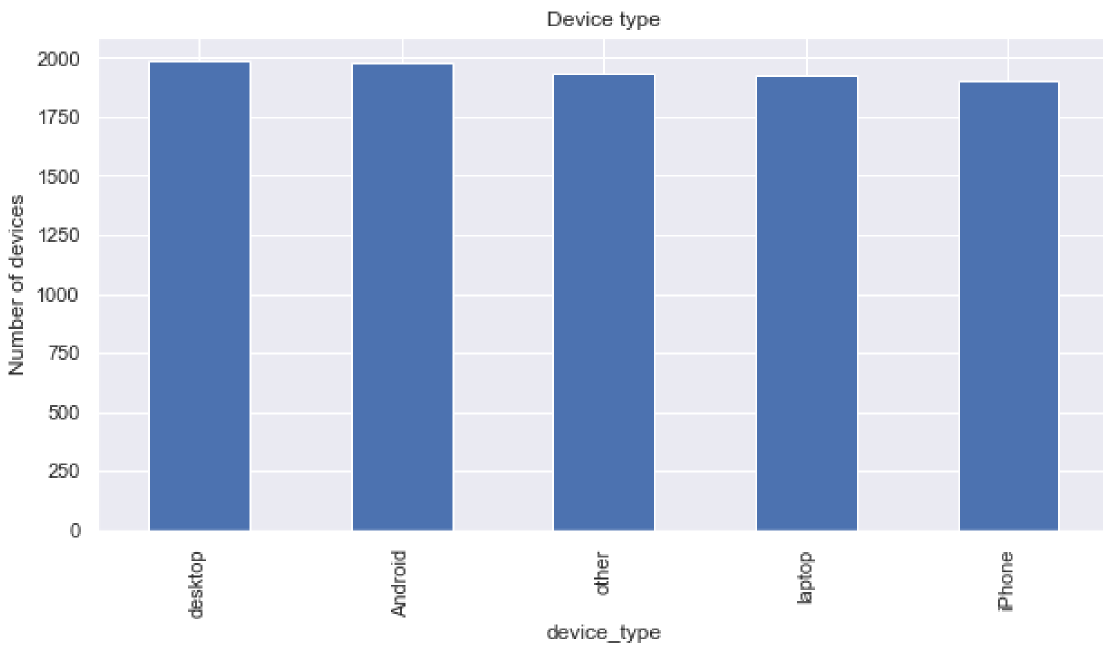
[369]:
```
sns.boxplot(x=df['prior_ins_tenure'])
```

[369]: <matplotlib.axes._subplots.AxesSubplot at 0x1a458c56d0>

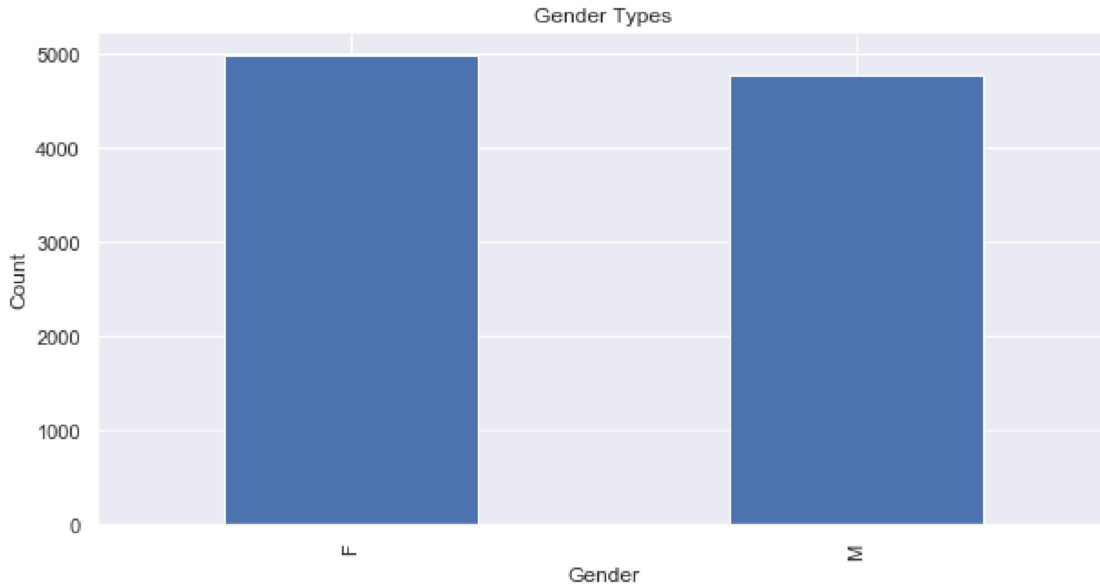


[370]:
```
# I could Eliminate the Outliers but the all Outliers are the 1's Outcome which␣
↪I found out after constructing few
# other plots which I have not included.

# Since, 1's are needed in the Outcome attribute, I didn't delete any Outlier␣
↪data.
```

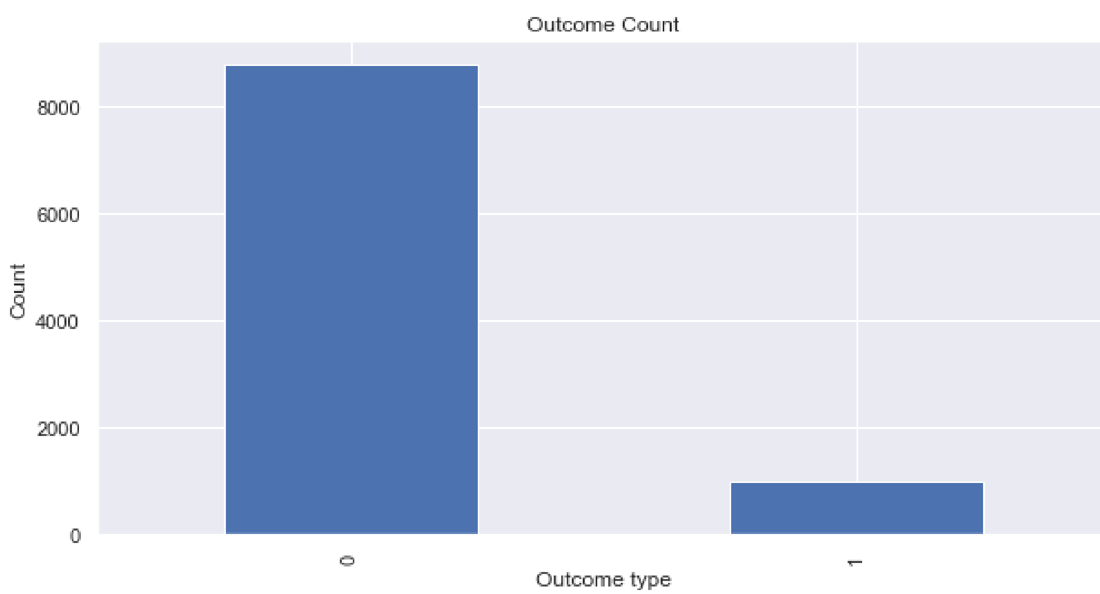
[371]:
```
# Plotting a Histogram for device type categorical variable

df.device_type.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title('Device type')
plt.ylabel('Number of devices')
plt.xlabel('device_type');
```



[372]:
```
# Plotting a Histogram for the gender attribute categorical value

df.gender.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title('Gender Types')
plt.ylabel('Count')
plt.xlabel('Gender');
```




```
[373]: # Plotting a Histogram for the Outcome attribute, where I found that the␣
       ↪classes are imbalnced and the data needs
       # to be balanced for proper training of the Machine Learning ALgorithm or else␣
       ↪the model would be baised and the
       #performance metrics would be no good to analyze

       df.outcome.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
       plt.title('Outcome Count')
       plt.ylabel('Count')
       plt.xlabel('Outcome type');
```

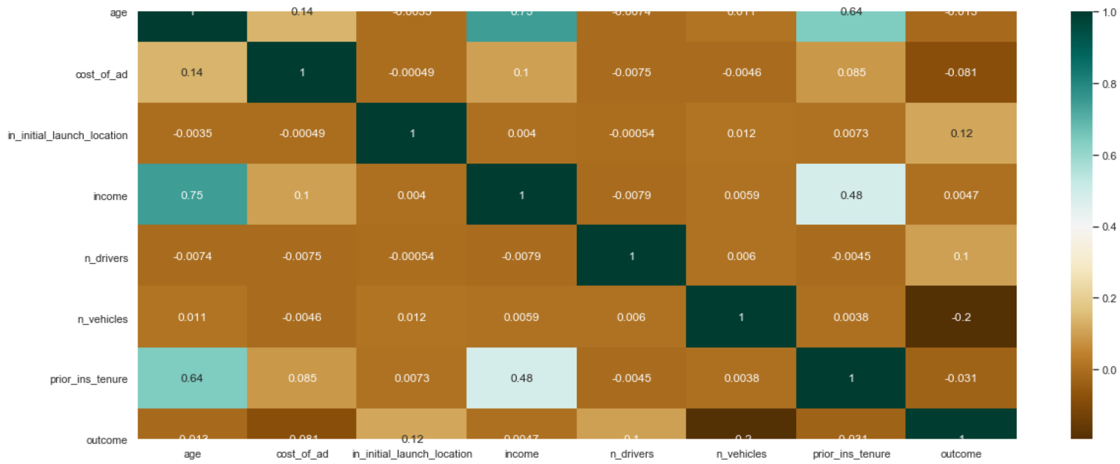


[374]:

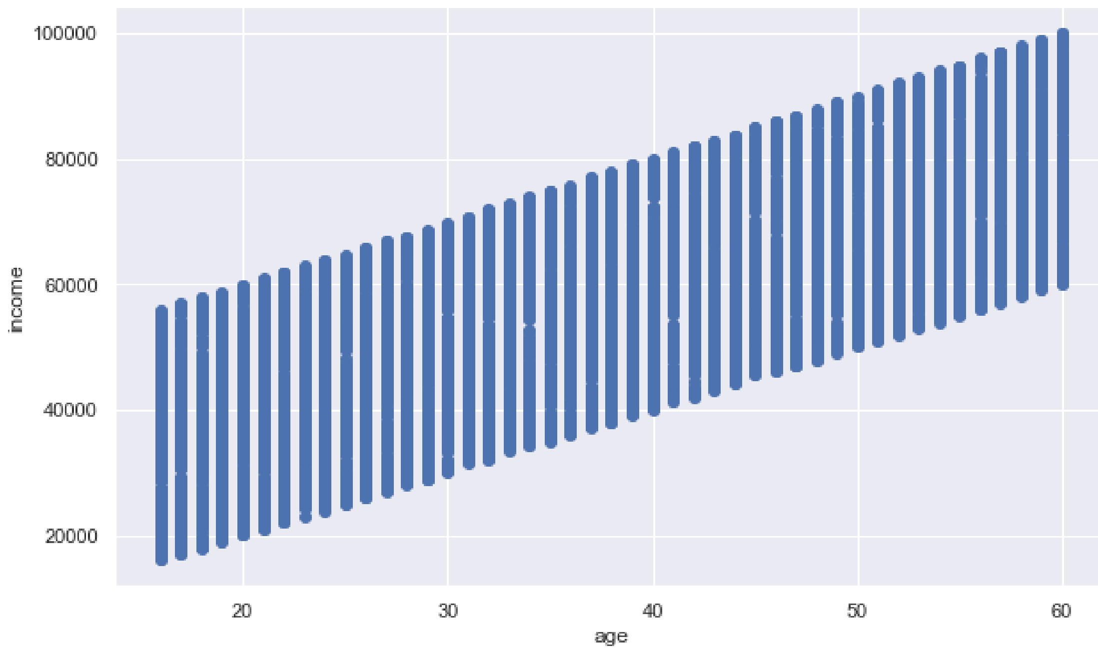
```python
# Finding the relations between the attributes

plt.figure(figsize=(20,8))
c= df.corr()
sns.heatmap(c,cmap='BrBG',annot=True)
c
```

[374]:

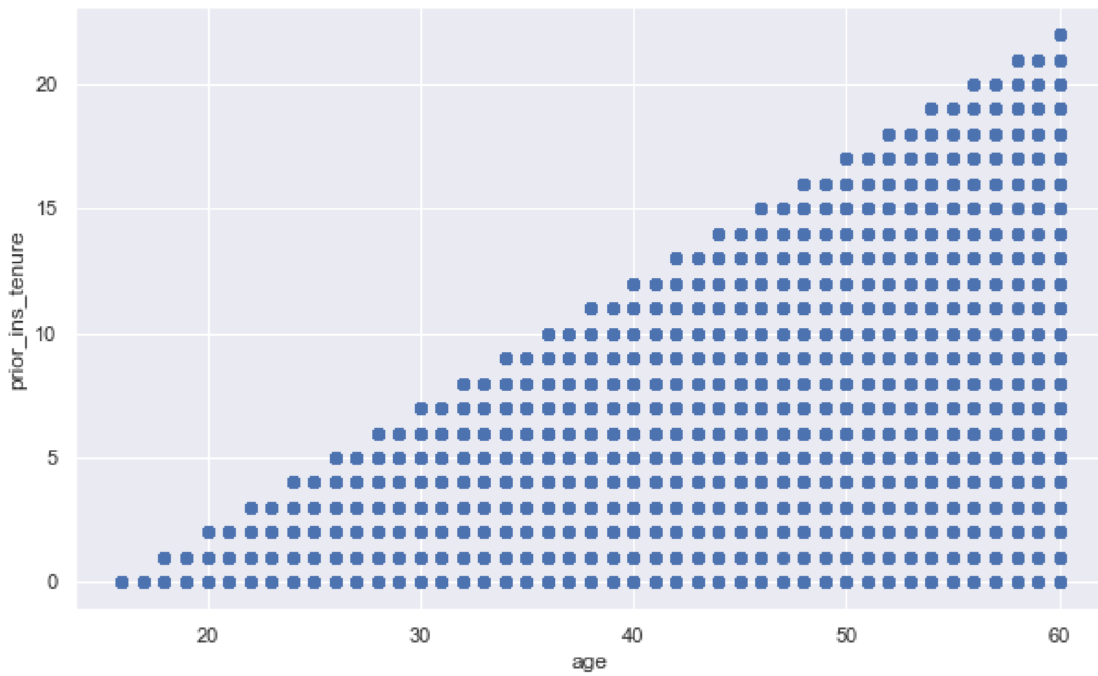
```
                                age  cost_of_ad  in_initial_launch_location  \
age                        1.000000    0.139465                   -0.003498
cost_of_ad                 0.139465    1.000000                   -0.000495
in_initial_launch_location -0.003498  -0.000495                    1.000000
income                     0.746357    0.100585                    0.004028
n_drivers                 -0.007366   -0.007465                   -0.000539
n_vehicles                 0.011223   -0.004628                    0.012063
prior_ins_tenure           0.643853    0.084788                    0.007318
outcome                   -0.013205   -0.080626                    0.116644

                             income  n_drivers  n_vehicles  prior_ins_tenure  \
age                        0.746357  -0.007366    0.011223          0.643853
cost_of_ad                 0.100585  -0.007465   -0.004628          0.084788
in_initial_launch_location  0.004028  -0.000539    0.012063          0.007318
income                     1.000000  -0.007855    0.005868          0.484850
n_drivers                 -0.007855   1.000000    0.006020         -0.004526
n_vehicles                 0.005868   0.006020    1.000000          0.003821
prior_ins_tenure           0.484850  -0.004526    0.003821          1.000000
outcome                    0.004659   0.101983   -0.195405         -0.031377

                            outcome
age                        -0.013205
cost_of_ad                 -0.080626
in_initial_launch_location  0.116644
income                      0.004659
n_drivers                   0.101983
n_vehicles                 -0.195405
prior_ins_tenure           -0.031377
outcome                     1.000000
```



[375]:
```python
# Plotting a scatter plot for more detail analysis between age and income

fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['age'], df['income'])
ax.set_xlabel('age')
ax.set_ylabel('income')
plt.show()
```



[376]:
```
# Plotting a scatter plot for more detail analysis  between age and
 ↪prior_ins_tenure

fig, ax = plt.subplots(figsize=(10,6))
ax.scatter(df['age'], df['prior_ins_tenure'])
ax.set_xlabel('age')
ax.set_ylabel('prior_ins_tenure')
plt.show()
```



[377]:
```
#Simple calculation to find the percentage of outcome classes

count_no_outcome = len(df[df['outcome']==0])
count_outcome = len(df[df['outcome']==1])
pct_of_no_outcome = count_no_outcome/(count_no_outcome+count_outcome)
print("percentage of no outcome is", pct_of_no_outcome*100)
pct_of_outcome = count_outcome/(count_no_outcome+count_outcome)
print("percentage of outcome", pct_of_outcome*100)
```

```
percentage of no outcome is 90.0010276436132
percentage of outcome 9.998972356386805
```

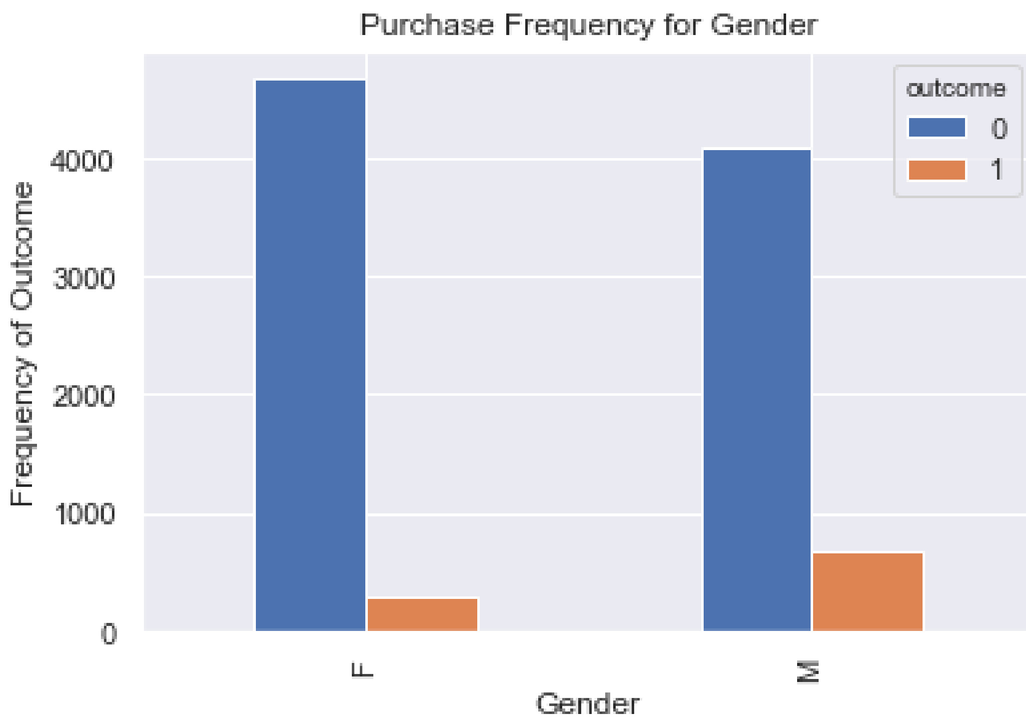
[378]:
```
#Further analysis of the data based on Device Type

df.groupby('device_type').mean()
```

[378]:
```
                   age  cost_of_ad  in_initial_launch_location        income  \
device_type
Android      38.413131    0.004394                    0.500505  58837.974242
desktop      37.586016    0.004392                    0.502515  57517.659960
iPhone       37.517585    0.005882                    0.500262  57489.867717
laptop       38.435484    0.004393                    0.485952  58146.458897
other        37.939050    0.004362                    0.501550  58071.095041

             n_drivers  n_vehicles  prior_ins_tenure   outcome
device_type
Android       1.482828    1.993939          5.468687  0.152525
desktop       1.504527    2.010563          5.431087  0.172535
iPhone        1.488189    1.985302          5.295538  0.086614
laptop        1.518210    1.986993          5.505203  0.045786
other         1.489153    2.013430          5.304752  0.038740
```

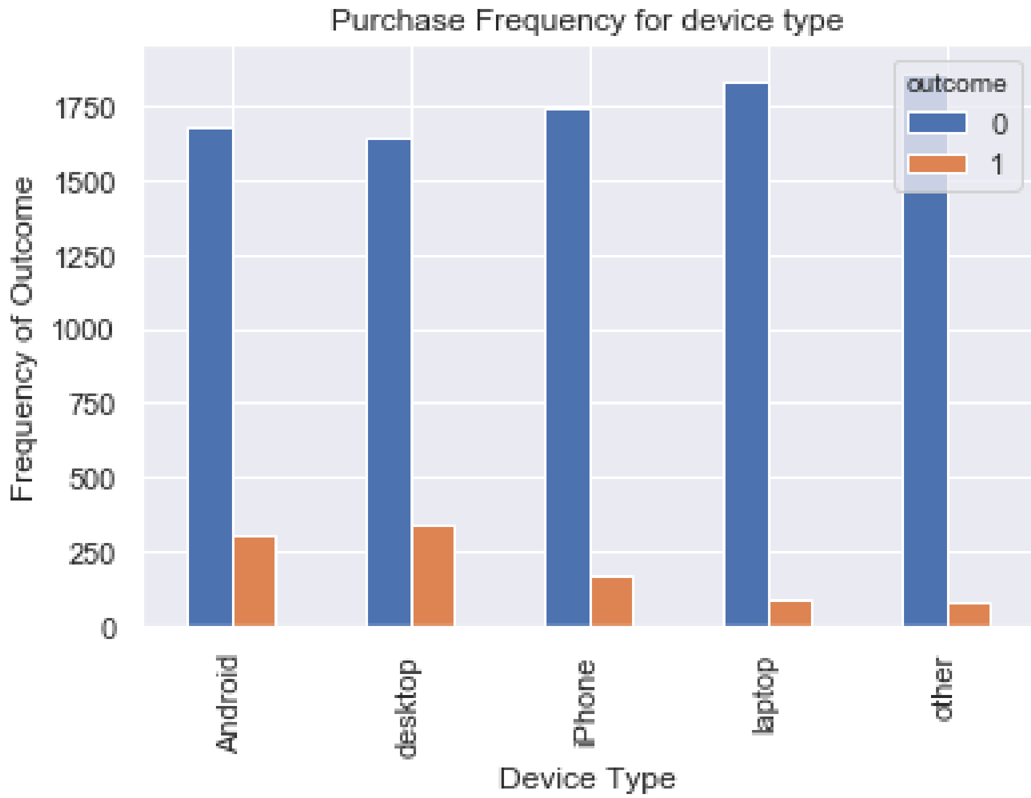
[379]:
```python
#Further analysis of the data based on Gender Type

df.groupby('gender').mean()
```

[379]:
```
              age  cost_of_ad  in_initial_launch_location        income  \
gender
F       38.051308    0.005174                    0.497183  58012.976459
M       37.903382    0.004160                    0.499265  58017.461878

        n_drivers  n_vehicles  prior_ins_tenure   outcome
gender
F        1.497787    1.995372          5.416901  0.058954
M        1.495274    2.001050          5.385843  0.142827
```

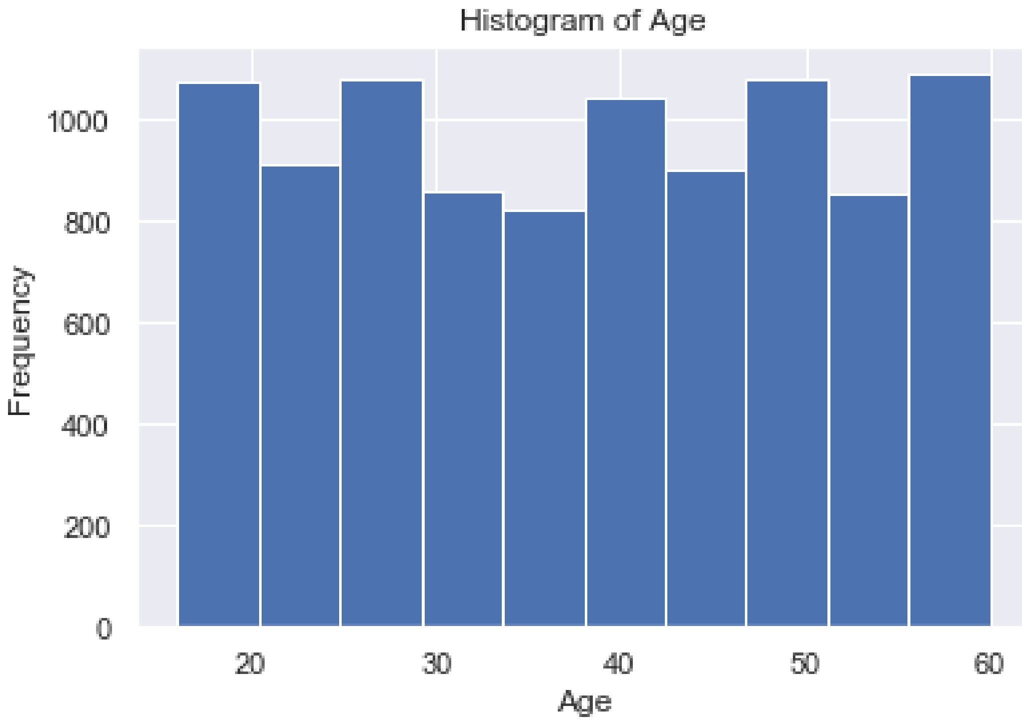
[380]:
```python
#Plotting a frequency graph for determing the outcome againist gender

%matplotlib inline
pd.crosstab(df.gender,df.outcome).plot(kind='bar')
plt.title('Purchase Frequency for Gender')
plt.xlabel('Gender')
plt.ylabel('Frequency of Outcome')
plt.savefig('purchase_fre_gender')
```



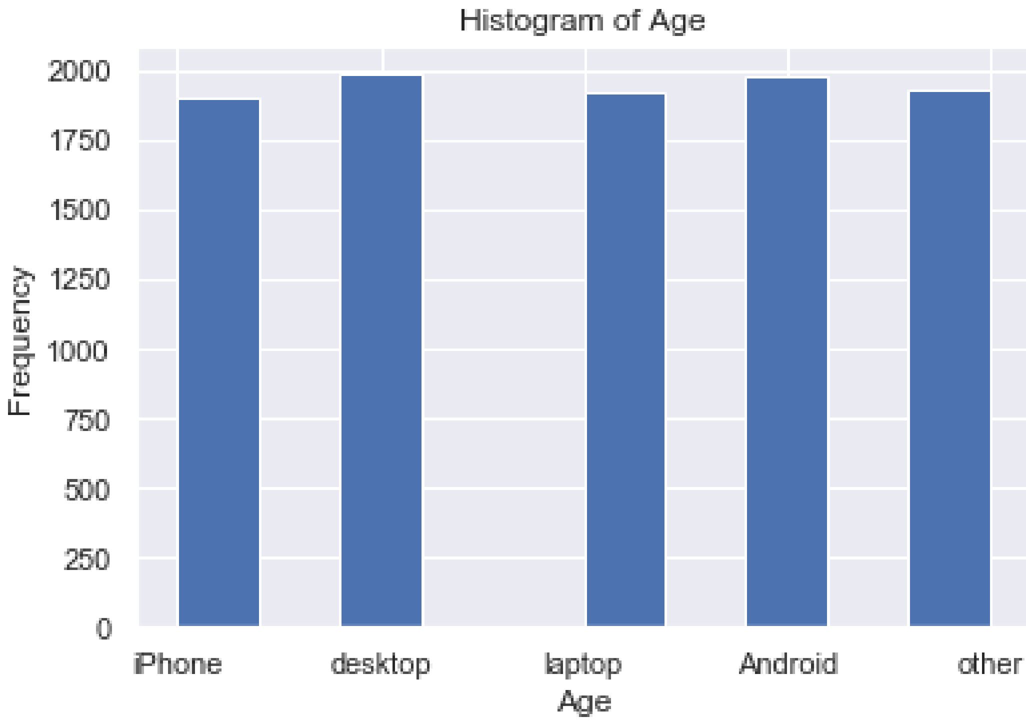

```
[381]: #Plotting a frequency graph for determing the outcome againist device type

%matplotlib inline
pd.crosstab(df.device_type,df.outcome).plot(kind='bar')
plt.title('Purchase Frequency for device type')
plt.xlabel('Device Type')
plt.ylabel('Frequency of Outcome')
plt.savefig('purchase_fre_device')
```




```
[382]: #Histogram distribution for age

df.age.hist()
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('hist_age')
```



[383]:
```
#Histogram distribution for device type

df.device_type.hist()
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('hist_age')
```



[384]:
```
#One hot encoding is performed to convert categorical attributes into numerical␣
↪data

# One hot encoding of device_type
one_hot = pd.get_dummies(df['device_type'])
df = df.drop('device_type',axis = 1)
df = df.join(one_hot)

# One hot encoding of gender
one_hot = pd.get_dummies(df['gender'])
df = df.drop('gender',axis = 1)
df = df.join(one_hot)
```

[385]:
```
#Displaying the data after one hot encoding

df
```

[385]:

	age	cost_of_ad	in_initial_launch_location	income	n_drivers	\
0	56	0.005737	0	62717	2	
1	50	0.004733	0	64328	2	
2	54	0.004129	0	83439	1	
3	16	0.005117	0	30110	2	
4	37	0.003635	0	76565	2	

```
…    …          …                             …      …        …
9994    58     0.003941                             0   95916          1
9995    41     0.004225                             0   64489          2
9996    50     0.004751                             0   88643          1
9997    60     0.003804                             0   87870          2
9998    18     0.003838                             0   56468          2

      n_vehicles  prior_ins_tenure  outcome  Android  desktop  iPhone  laptop  \
0              1                 4        0        0        0       1       0
1              3                 2        0        0        1       0       0
2              3                 7        0        0        0       0       1
3              3                 0        0        1        0       0       0
4              1                 5        0        0        1       0       0
…           …                …      …      …      …      …      …
9994           1                18        0        1        0       0       0
9995           3                 8        0        0        1       0       0
9996           3                 0        0        0        0       0       0
9997           2                 9        0        0        0       0       0
9998           2                 0        0        0        0       0       1

      other  F  M
0         0  0  1
1         0  1  0
2         0  0  1
3         0  1  0
4         0  0  1
…       … .. ..
9994      0  0  1
9995      0  0  1
9996      1  1  0
9997      1  0  1
9998      0  0  1

[9731 rows x 15 columns]
```

[386]:

```python
#Since the outcome classesa are imbalanced, I have performed upsampling of the␣
↪minority classes, i.e, 1
#But before upsampling the data, I'm splitting the data into Training and␣
↪Testing sets and perfomed upsampling only
#on the Training set. This will ensure that the sample are not repeted in the␣
↪Training and Testing sets

from sklearn.utils import resample

# Separating the input features and target
X = df.drop('outcome', axis=1)
y = df.outcome
```

```
# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=27)

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
no_outcome = X[X.outcome==0]
outcome = X[X.outcome==1]

#upsample minority
outcome_upsampled = resample(outcome,
                          replace=True, # sample with replacement
                          n_samples=len(no_outcome), # match number in majority␣
↪class
                          random_state=27) # reproducible results

#combine majority and upsampled minority
upsampled = pd.concat([no_outcome, outcome_upsampled])

# we can Check the numbers of our data
print("Length of oversampled data is ",len(upsampled))
print("Number of no outcome in oversampled␣
↪data",len(upsampled[upsampled['outcome']==0]))
print("Number of outcome",len(upsampled[upsampled['outcome']==1]))
print("Proportion of no outfome data in oversampled data is␣
↪",len(upsampled[upsampled['outcome']==0])/len(upsampled))
print("Proportion of outcome data in oversampled data is␣
↪",len(upsampled[upsampled['outcome']==1])/len(upsampled))
```

```
Length of oversampled data is  13144
Number of no outcome in oversampled data 6572
Number of outcome 6572
Proportion of no outfome data in oversampled data is  0.5
Proportion of outcome data in oversampled data is  0.5
```

[387]:
```
#Storing the upsampled training data into X and displaying
X = upsampled
```

[388]:
```
X
```

[388]:

	age	cost_of_ad	in_initial_launch_location	income	n_drivers	\
9070	50	0.004715	0	75615	2	
983	39	0.005185	0	47571	2	
9075	27	0.004737	0	59864	2	

```
6697    22   0.003926                          0   27698          1
141     44   0.004715                          0   52270          2
…      …          …                   …     …       …
3557    56   0.004089                          1   78904          1
883     24   0.003317                          1   60109          2
9008    44   0.005916                          1   49624          1
8896    33   0.004886                          1   59309          2
1271    31   0.003835                          1   45961          1

      n_vehicles  prior_ins_tenure  Android  desktop  iPhone  laptop  other  \
9070           3                 0        0        0       0       1      0
983            3                 2        0        1       0       0      0
9075           1                 2        0        1       0       0      0
6697           3                 1        0        0       0       1      0
141            1                 1        0        0       0       1      0
…            …                …      …      …      …      …      …
3557           1                 4        0        1       0       0      0
883            1                 0        1        0       0       0      0
9008           1                 6        0        0       1       0      0
8896           1                 3        0        0       1       0      0
1271           1                 4        0        1       0       0      0

      F  M  outcome
9070  1  0        0
983   1  0        0
9075  1  0        0
6697  0  1        0
141   1  0        0
…    .. ..      …
3557  0  1        1
883   0  1        1
9008  0  1        1
8896  0  1        1
1271  0  1        1

[13144 rows x 15 columns]
```

[389]:

```python
#Counting the number of Outcome values, we can see that the training data␣
↪contains equal amount of classes

X.outcome.value_counts()
```

[389]:

```
1    6572
0    6572
Name: outcome, dtype: int64
```

[390]:
```
#Seperating the attributes and labels from the Training Data
import numpy as np

# Labels are the values that we want to predict
labels = np.array(X['outcome'])

df_list = list(X.columns)

X = X.drop('outcome', axis =1)
df = np.array(X)
```

[391]:
```
#Storing the data into specific values, so that we can directly feed into the␣
↪Random Forest Model.
#X_test and y_test are the testing data attributes and their labels. Remember␣
↪how we split the data before Upsampling

train_features = df
train_labels = labels
test_features = X_test
test_labels = y_test
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (13144, 14)
Training Labels Shape: (13144,)
Testing Features Shape: (2433, 14)
Testing Labels Shape: (2433,)
```

[392]:
```
#Implementing the Random Forest Classifier and Calculating it's performance␣
↪metrics

from sklearn.ensemble import RandomForestClassifier

# Creating the model with 100 trees
rf = RandomForestClassifier(n_estimators = 50, max_leaf_nodes = 10,␣
↪class_weight = 'balanced', random_state = 42)

# Fit on training data
rf.fit(train_features, train_labels);

# Use the random forest's predict method on the test data
predictions = rf.predict(test_features)

accuracy = rf.score(test_features, test_labels)
```

```
# Probabilities for each class
rf_probs = rf.predict_proba(test_features)[:, 1]

from sklearn.metrics import roc_auc_score

# Calculate roc auc
auc_value = roc_auc_score(test_labels, rf_probs)
print("The Auc Value:" , auc_value)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_labels, predictions)
print("The confusion Matrix:\n",cm)

from sklearn.metrics import f1_score
f1_score = f1_score(test_labels, predictions)
print("F1 score:", f1_score)

from sklearn.metrics import accuracy_score
Training_accuracy = accuracy_score(train_labels, rf.predict(train_features))
Testing_accuracy = accuracy_score(test_labels, predictions)
print("Training accuracy:", Training_accuracy)
print("Testing accuracy:", Testing_accuracy)

from sklearn.metrics import precision_score, recall_score
precision_score = precision_score(test_labels, predictions)

recall_score = recall_score(test_labels, predictions)

print("Precision Score:", precision_score)
print("Recall Score:", recall_score)
```

```
The Auc Value: 0.8553224235195631
The confusion Matrix:
 [[1550  636]
 [  40  207]]
F1 score: 0.3798165137614679
Training accuracy: 0.7442939744370055
Testing accuracy: 0.7221537196876284
Precision Score: 0.24555160142348753
Recall Score: 0.8380566801619433
```

[393]:
```
# Getting  numerical feature importances
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance␣
↪in zip(df_list, importances)]
```

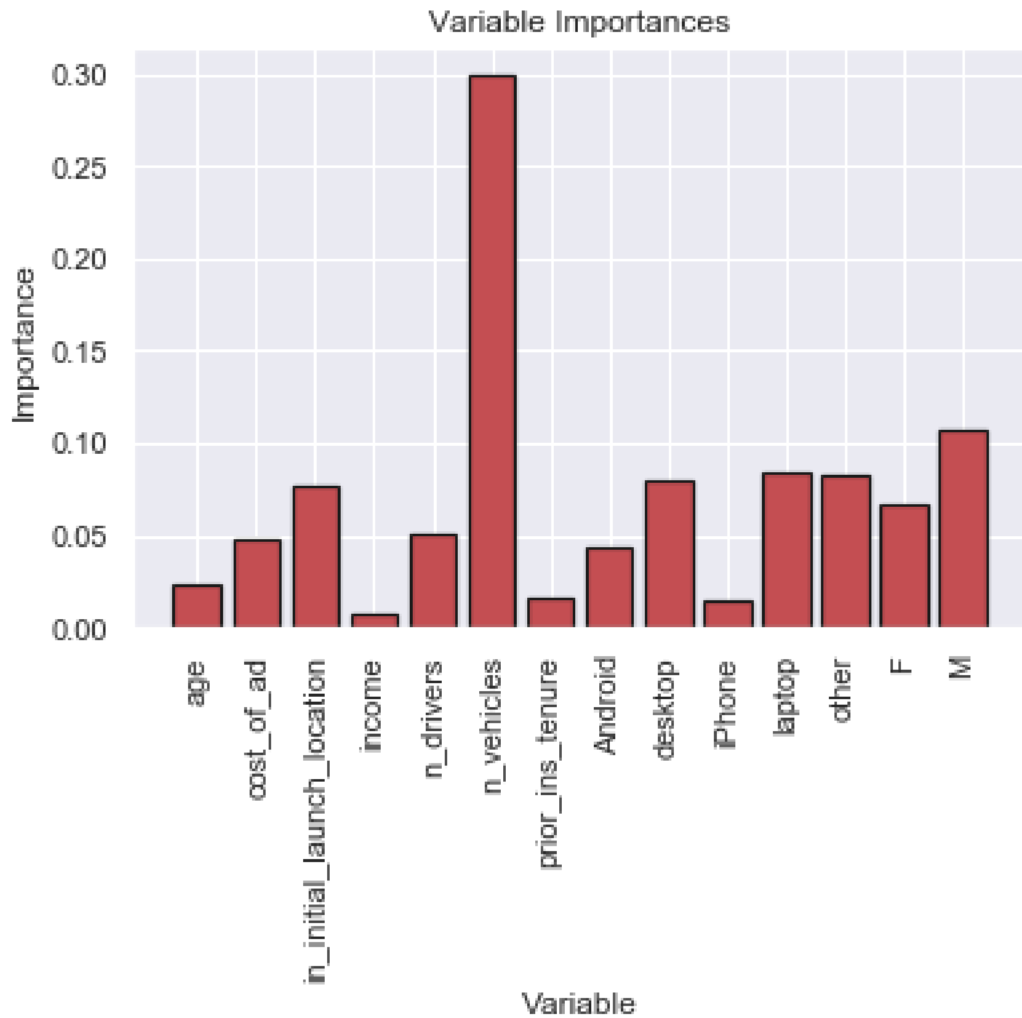
```
# Sorting the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse␣
↪= True)

# Printing out the feature and it's importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in␣
↪feature_importances]
```

```
Variable: n_vehicles           Importance: 0.3
Variable: M                    Importance: 0.11
Variable: in_initial_launch_location Importance: 0.08
Variable: desktop              Importance: 0.08
Variable: laptop               Importance: 0.08
Variable: other                Importance: 0.08
Variable: F                    Importance: 0.07
Variable: cost_of_ad           Importance: 0.05
Variable: n_drivers            Importance: 0.05
Variable: Android              Importance: 0.04
Variable: age                  Importance: 0.02
Variable: prior_ins_tenure     Importance: 0.02
Variable: income               Importance: 0.01
Variable: iPhone               Importance: 0.01
```

[393]:
```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None]
```

[394]:
```
#Plotting the feature importance in a much more visual way

import matplotlib.pyplot as plt
x_values = list(range(len(importances)))
# Make a bar chart
plt.bar(x_values, importances, orientation = 'vertical', color = 'r', edgecolor␣
↪= 'k', linewidth = 1.2)
# Tick labels for x axis
```

```
plt.xticks(x_values, df_list, rotation='vertical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable␣
↪Importances');
```



[395]:
```
#Importing the Test Data from the local machine

import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
df_test = pd.read_csv('/Users/harshith/Downloads/data/Test.csv')
df_test.head(5)
```

[395]:

```
   age  cost_of_ad device_type gender  in_initial_launch_location  income  \
0   34    0.005134     Android      F                           1   40376
1   53    0.005223     desktop      F                           1   84511
2   46    0.004939      laptop      F                           0   79322
3   36    0.004924     Android      F                           0   63295
4   28    0.005146       other      F                           1   36170

   n_drivers  n_vehicles  prior_ins_tenure
0          1           3                 7
1          1           1                11
2          1           1                 4
3          1           2                 0
4          1           3                 3
```

[398]:

```python
#Dropping the NA values from the test data file

print(df_test.isnull().sum())
```

```
age                             0
cost_of_ad                      0
device_type                     0
gender                        249
in_initial_launch_location      0
income                          0
n_drivers                       0
n_vehicles                      0
prior_ins_tenure                0
dtype: int64
```

[400]:

```python
df_test = df_test.dropna()
```

[401]:

```python
#One hot encoding the Test Data's Categorical Values

# One hot encoding of device_type
one_hot = pd.get_dummies(df_test['device_type'])
df_test = df_test.drop('device_type',axis = 1)
df_test = df_test.join(one_hot)

# One hot encoding of gender
one_hot = pd.get_dummies(df_test['gender'])
df_test = df_test.drop('gender',axis = 1)
df_test = df_test.join(one_hot)
```

[402]:

```python
df_test = np.array(df_test)
```

[403]:

```python
#Making the predictions on the Test Data
```

```python
predictions = rf.predict(df_test)
```

[404]:
```python
#Displaying the Results and the shape

predictions
```

[404]: array([0, 1, 0, …, 0, 0, 0])

[405]:
```python
predictions.shape
```

[405]: (9751,)

[406]:
```python
#Converting the predictions to a data frame so that it can be used for future␣
 ↪purposes
import pandas as pd
predictions = pd.DataFrame(predictions)
```

[407]:
```python
#Printing the first 5 predictions of the Test Data

print(predictions.head(5))
```

```
   0
0  0
1  1
2  0
3  0
4  0
```

[408]:
```python
#Optional: Saving the predictions to a CVS file into our local machine
predictions.to_csv('/Users/harshith/Downloads/data/predictions.csv',␣
 ↪index=False)
```

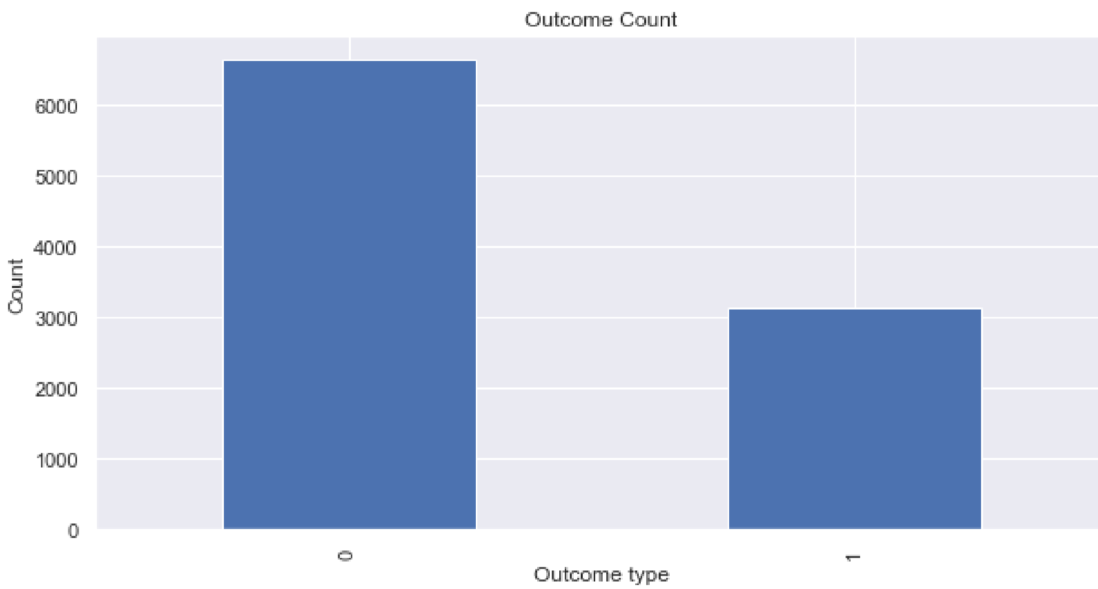
[409]:
```python
#Optional: Reading the CSV file of predictions

import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
df = pd.read_csv('/Users/harshith/Downloads/data/predictions.csv',␣
 ↪names=['Outcome'])
df.head(5)
```

[409]:

	Outcome
0	0
1	0

```
2        1
3        0
4        0
```

```
[410]: # Optional: Plotting a Histogram for getting an understanding of the Outcome␣
        ↪distribution

       df.Outcome.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
       plt.title('Outcome Count')
       plt.ylabel('Count')
       plt.xlabel('Outcome type');
```




```
[411]: #Optional: Calculating the percentage of Outcome Classes

       count_no_outcome = len(df[df['Outcome']==0])
       count_outcome = len(df[df['Outcome']==1])
       pct_of_no_outcome = count_no_outcome/(count_no_outcome+count_outcome)
       print("percentage of no outcome is", pct_of_no_outcome*100)
       pct_of_outcome = count_outcome/(count_no_outcome+count_outcome)
       print("percentage of outcome", pct_of_outcome*100)
```

```
percentage of no outcome is 67.95529122231338
percentage of outcome 32.04470877768662
```

```
[412]: #Lastly, printing the first 5 predictions

       print(predictions.head(5))
```

```
   0
0  0
1  1
2  0
3  0
4  0
```

[413]:
```python
#Optional: Printing the last 5 predictions

print(predictions.tail(5))
```

```
      0
9746  0
9747  0
9748  0
9749  0
9750  0
```