DC1B_Amy_Wen

October 17, 2019

1 Background

Company XYZ is interested in whether increasing the price of their software from $39 to $59 would increase their revenue. To answer this question, an experiment was performed where a random sample of 2/3 of the users saw $39 as the software price while the remaining 1/3 saw $59.

2 Executive Summary

For company XYZ to increase their revenue, it is recommended that the company sells their software for **$59**. Although there was a 1.25-fold reduction in conversion with the higher price, the price itself is 1.5-fold higher. Therefore, the higher price successfully counterbalances the reduction in conversion, with a net 1.2-fold increase in revenue.

Some additional insights from exploring the data reveal that the company should encourage friend referrals, focus on Google and Facebook advertisements, and determine why Mac and iOS operating systems result in higher. Friend referrals had the highest conversion rate proportion, followed by Google and Facebook advertisements. Similarly, Mac and iOS operating systems also had the highest conversion rates. However, Windows users were the highest proportion of visitors and there was no conversion observed from Linux users, so this indicates that the website or software may not work as well on these operating systems. Access to these customer bases would drive up revenue.

All these recommendations are assuming that the data collection process was correct. However, there were some concerning findings that undermine this assumption. 0.1% of the data had mismatches between the 'test' column and the 'price' column, which should match since the control group should all see a price of $39 and the test group should all see a price of $59. Also, 0.2% of the users were from "St. Petersburg, USA". It may not be an issue for the analysis, but about 13% of users that were in the test results were not in the users table. It would be good to know why these users did not exist, whether it was a benign reason, such as purchases can be made without being a logged in, or whether there was a period of time where the data was not being recorded or was recorded incorrectly. Finally, about 3% of the entries had unusual timestamps, where 60 was the value for seconds or minutes. Again, it would be important to determine the underlying reason, whether some systems have a delay between changing 60 to 0 or whether the timestamping process is completely erroneous. While reasonable results were obtained regardless, it would be important to check the data collection process to make sure these were clerical errors for select datapoints and not, for example, a shift in the data where the results in the rows are not aligned correctly.

[1]:
```
# Import necessary packages
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

# Resampling
from sklearn.utils import resample

# Model selection
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report, roc_curve,␣
↪roc_auc_score

# Classification model
from sklearn.linear_model import LogisticRegression

# Statistics
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
```

3 Exploratory Data Analysis and Data Cleaning

Some initial exploratory analysis and descriptive statistics of the data was performed.

Some questionable data that only represented a small proportion of the overall data were removed since they shouldn't affect the overall analysis: - There were mismatches in 365 out of the 316,800 test data, where the test column indicated that the user was in the control group (or test group) but the price column showed the new price of \$59 (or \$39). Since this was only 0.1% of the data and there is no way of knowing which column has the correct information, these rows were removed. - There were 705 unusual datapoints from St. Petersburg, USA, representing only 0.2% of the data, so they were also removed.

There were also some other significant errors, but since they represent a larger proportion of the overall data, they were not immediately removed. Further exploration of whether or not removal will bias the data will be performed first. - There were 41,184 users that were in the test results but not the users table (such as user_id==501672). - There were 10,225 entries with unusual timestamps, where 60 was input for seconds or minutes.

[2]:
```
# Load data
test = pd.read_csv("test_results.csv")
users = pd.read_csv("user_table.csv")

# Merge the two datasets using user_id
data = test.merge(users, left_on='user_id', right_on='user_id', how='left')

# Examine data
```

```python
print(data.describe()) # 316,800 rows from test, but only 275,616 rows from
 ↪users
data.head()
```

```
              user_id           test          price      converted  \
count   316800.000000  316800.000000  316800.000000  316800.000000
mean    499281.341840       0.360079      46.205051       0.018333
std     288591.154044       0.480024       9.601487       0.134154
min          3.000000       0.000000      39.000000       0.000000
25%     249525.750000       0.000000      39.000000       0.000000
50%     499021.500000       0.000000      39.000000       0.000000
75%     749025.500000       1.000000      59.000000       0.000000
max    1000000.000000       1.000000      59.000000       1.000000

                 lat           long
count  275616.000000  275616.000000
mean       37.111680     -93.981772
std         5.209627      18.086486
min        19.700000    -157.800000
25%        33.660000    -112.200000
50%        37.740000     -88.930000
75%        40.700000     -78.910000
max        61.180000      30.310000
```

[2]:

```
   user_id            timestamp          source  device operative_system  \
0   604839  2015-05-08 03:38:34     ads_facebook  mobile              iOS
1   624057  2015-05-10 21:08:46       seo-google  mobile          android
2   317970  2015-04-04 15:01:23         ads-bing  mobile          android
3   685636  2015-05-07 07:26:01   direct_traffic  mobile              iOS
4   820854  2015-05-24 11:04:40     ads_facebook     web              mac

   test  price  converted          city country    lat   long
0     0     39          0       Buffalo     USA  42.89 -78.86
1     0     39          0     Lakeville     USA  44.68 -93.24
2     0     39          0         Parma     USA  41.38 -81.73
3     1     59          0  Fayetteville     USA  35.07 -78.90
4     0     39          0       Fishers     USA  39.95 -86.02
```

Mismatches between test and price columns were identified below.

[3]:

```python
# Test if test and price columns match correctly
mismatches = data[(data.price-data.test*20) != 39] # price=(39, 59) should
 ↪correspond to test=(0, 1)
print(mismatches.shape) # 365 mismatches
mismatches.tail()
```

```
(365, 12)
```

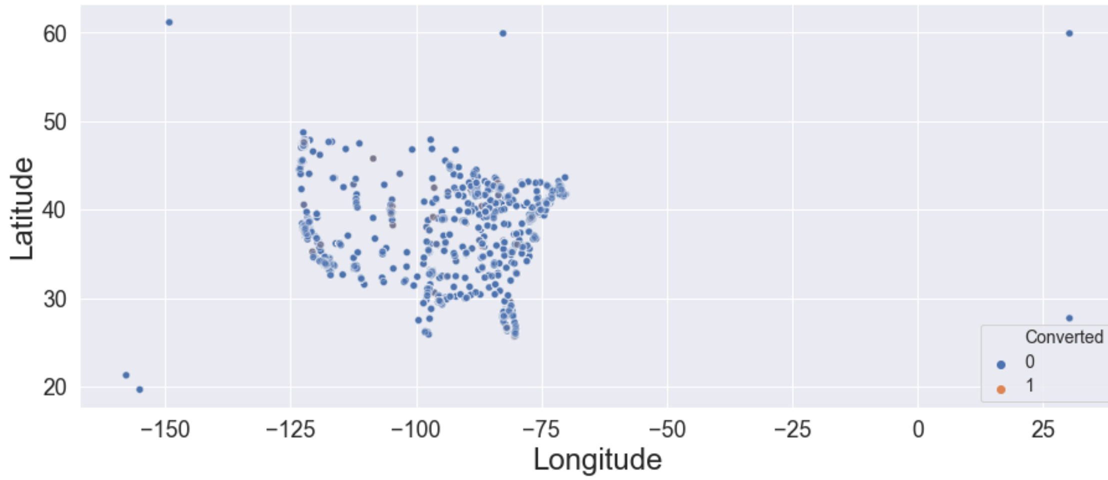
[3]:
```
        user_id            timestamp          source  device operative_system  \
314402   191130  2015-04-10 15:45:42  direct_traffic  mobile          android
314696   237644  2015-05-15 11:41:49  direct_traffic  mobile          android
315529   590389  2015-04-14 04:07:41  direct_traffic  mobile              iOS
315864   748425  2015-05-25 19:15:05      ads-google     web          windows
316663   501672  2015-04-01 08:55:41    seo_facebook     web          windows

        test  price  converted         city country    lat    long
314402     1     39          0    Placentia     USA  33.88 -117.85
314696     1     39          0  Los Angeles     USA  34.11 -118.41
315529     0     59          0    Livermore     USA  37.69 -121.76
315864     1     39          0     Mesquite     USA  32.77  -96.60
316663     0     59          0          NaN     NaN    NaN     NaN
```

[4]:
```
# Remove points where 'test' and 'price' do not match
data = data[(data.price-data.test*20) == 39]
```

Latitude vs. longitude was plotted to see if there were any observable differences in conversion due to location. While no clear differences were observed, points outside of the US (from Saint Petersburg) were discovered.

[5]:
```
# Function for legible graphs
def legible_graph(x, y):
    sns.set(rc={'figure.figsize':(x,y)})
    plt.rc('axes', labelsize=24) # Fontsize of x and y labels
    plt.rc('xtick', labelsize=18) # Fontsize of tick labels
    plt.rc('ytick', labelsize=18)

# Examine scatterplots of latitude and longitude colored by converted
legible_graph(15,6)
ax = sns.scatterplot(x="long", y="lat", hue="converted", data=data, alpha=0.3)
legend = ax.legend()
legend.texts[0].set_text("Converted")
plt.setp(ax.get_legend().get_texts(), fontsize='14') # Fontsize for legend text
plt.setp(ax.get_legend().get_title(), fontsize='18') # Fontsize for legend
 ↪title
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

[5]:
```
Text(0, 0.5, 'Latitude')
```



The figure above shows the conversion rate in the various user locations. No observable differences in conversion rates due to location were observed, but there were points outside of the US (from Saint Petersburg).

[6]:
```
# Examine points from outside US map
print(data[(data.long>0) & (data.lat<40)].head()) # Points from St. Petersburg,␣
↪USA
print(data[data.city == 'Saint Petersburg'].shape) # Number of datapoints from␣
↪St. Petersburg
data = data[data.city != 'Saint Petersburg'] # Remove strange Russian cities
```

```
      user_id            timestamp          source  device operative_system  \
3207   889901  2015-04-01 18:60:56    ads_facebook     web          windows
4357   735454  2015-05-16 06:60:01      ads-google  mobile          android
5021   786119  2015-03-09 13:42:42      seo-google     web            linux
5446   322909  2015-03-30 12:28:49      ads-google  mobile              iOS
7115   351134  2015-03-12 10:57:38  direct_traffic  mobile              iOS

      test  price  converted              city country    lat   long
3207     0     39          0  Saint Petersburg     USA  27.76  30.31
4357     1     59          0  Saint Petersburg     USA  27.76  30.31
5021     0     39          0  Saint Petersburg     USA  27.76  30.31
5446     1     59          0  Saint Petersburg     USA  27.76  30.31
7115     0     39          0  Saint Petersburg     USA  27.76  30.31
(705, 12)
```

The timestamps were converted to datetime objects. During the process, a strange error was discovered, where over 10,000 entries had 60 as the minutes or seconds for the timestamps.

[7]:
```
# Change the datatype of timestamp to datetime
data['timestamp_ignore_errors'] = pd.to_datetime(data.timestamp.astype(str),
                                                 format='%Y-%m-%d %H:%M:%S',␣
↪errors='ignore')
```

```
data['timestamp'] = pd.to_datetime(data.timestamp.astype(str),
                                    format='%Y-%m-%d %H:%M:%S',␣
 ↪errors='coerce')

# Output timestamps that had errors
print(data[[x=='NaT' for x in list(map(str, data['timestamp']))]].head()) # A␣
 ↪lot of 60's for minutes or seconds
print(data[[x=='NaT' for x in list(map(str, data['timestamp']))]].shape)
```

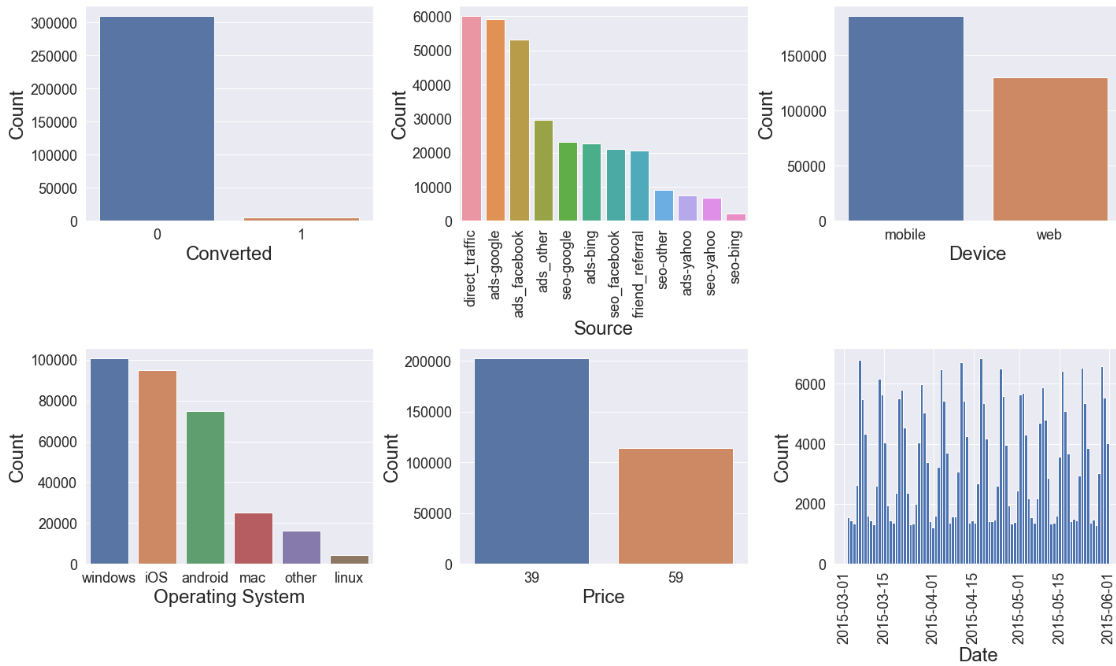
```
     user_id timestamp           source  device operative_system  test  price  \
54    370914       NaT   direct_traffic  mobile          android     0     39
104   549807       NaT  friend_referral  mobile              iOS     0     39
121   107010       NaT   direct_traffic     web          windows     0     39
278   287830       NaT   direct_traffic     web          windows     1     59
282   676183       NaT       ads-google     web          windows     1     59

     converted             city country    lat    long  \
54           0  North Charleston     USA  32.91  -80.04
104          0       San Antonio     USA  29.46  -98.51
121          0            Dallas     USA  32.79  -96.77
278          0           Chicago     USA  41.84  -87.68
282          0         Las Vegas     USA  36.21 -115.22

    timestamp_ignore_errors
54      2015-04-24 12:60:46
104     2015-04-24 11:60:20
121     2015-03-14 12:60:02
278     2015-04-04 02:23:60
282     2015-05-11 12:60:53
(10225, 13)
```

The distribution of the various variables were considered to see if any trends could be observed.

[8]:
```
# Examine distribution of all the variables
legible_graph(20,12)

# Define plotting function
def countplot(i, j, k, series, xlabel, ylabel):
    plt.subplot(i, j, k)
    sns.countplot(series)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

# Dependent variable (converted)
countplot(2, 3, 1, data.converted, 'Converted', 'Count')

# Categorical variables (source, device, operative_system, price)
```

```
plt.subplot(2, 3, 2)
sns.countplot(data.source, order=data['source'].value_counts().index) # Plot in␣
↪order from most to least popular
plt.xticks(rotation=90)
plt.xlabel('Source')
plt.ylabel('Count')

countplot(2, 3, 3, data.device, 'Device', 'Count')

plt.subplot(2, 3, 4)
sns.countplot(data.operative_system, order=data['operative_system'].
↪value_counts().index)
plt.xlabel('Operating System')
plt.ylabel('Count')

countplot(2, 3, 5, data.price, 'Price', 'Count')

# Remaining variable (timestamp), after removing 'NaT' entries
plt.subplot(2, 3, 6)
data_cleaned = data[[x!='NaT' for x in list(map(str, data['timestamp']))]].
↪reset_index(drop=True)
plt.hist(data_cleaned.timestamp, bins = 93)
plt.xticks(rotation=90)
plt.xlabel('Date')
plt.ylabel('Count')
plt.tight_layout()
```

```
//anaconda3/lib/python3.7/site-packages/pandas/plotting/_converter.py:129:
FutureWarning: Using an implicitly registered datetime converter for a
matplotlib plotting method. The converter was registered by pandas on import.
Future versions of pandas will require you to explicitly register matplotlib
converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```



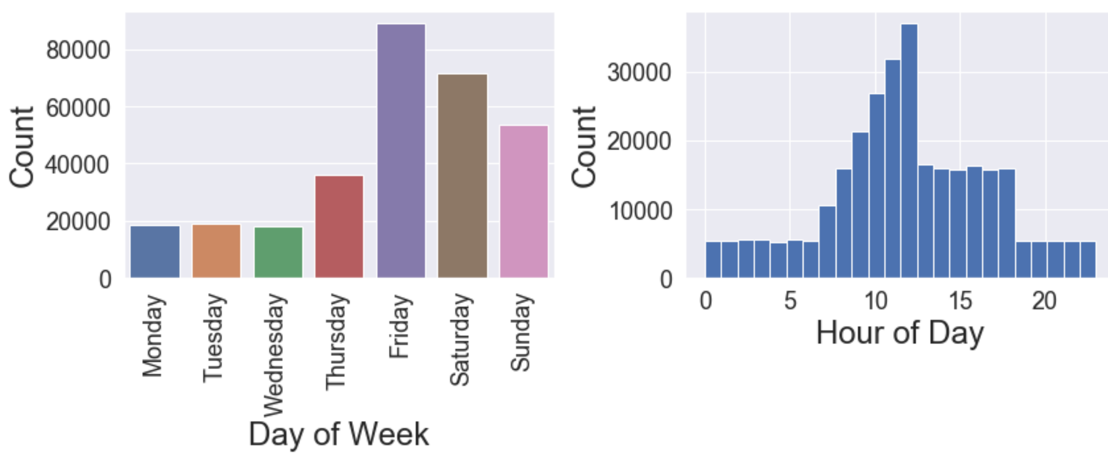
The figure above demonstrates the distribution of the various variables.

Some observations that stood out were: - There was a very low conversion rate - The main sources of traffic were direct traffic as well as Google, Facebook, and other ads - Slightly more users visited the site from a mobile device - Of the web users, Windows operating system was by far the more popular, while iOS was more popular for mobile users - The price distribution was about as expected (2:1 ratio for \$39 vs. \$59) - There seemed to be a weekly cycle in popularity patterns based on date, with fairly constant traffic over time.

Based on the periodic nature of the time data, the dates were converted into day of the week. Hour of day was also extracted.

[9]:
```
# Determine day of week based on time stamp
# Functions for map to use
weekDays =␣
 ↪["Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"]

def to_weekday(day):
    return weekDays[day.weekday()]

def to_hour(day):
    return day.hour

# Perform mapping of timestamp to day of week and hour of day
data_cleaned['day_of_week'] = list(map(to_weekday, data_cleaned.timestamp))
data_cleaned['hour_of_day'] = list(map(to_hour, data_cleaned.timestamp))

# For day of week data, specify order for the days
data_cleaned['day_of_week'] = data_cleaned['day_of_week'].astype(
```

```
    pd.api.types.CategoricalDtype(categories=weekDays, ordered=True))

# Replot timestamp data based on day of week and hour of day
legible_graph(12, 5)

# Day of Week
plt.subplot(1, 2, 1)
sns.countplot(data_cleaned.day_of_week)
plt.xticks(rotation=90)
plt.xlabel('Day of Week')
plt.ylabel('Count')

# Hour of Day
plt.subplot(1, 2, 2)
plt.hist(data_cleaned.hour_of_day, bins=24)
plt.xlabel('Hour of Day')
plt.ylabel('Count')
plt.tight_layout()
```

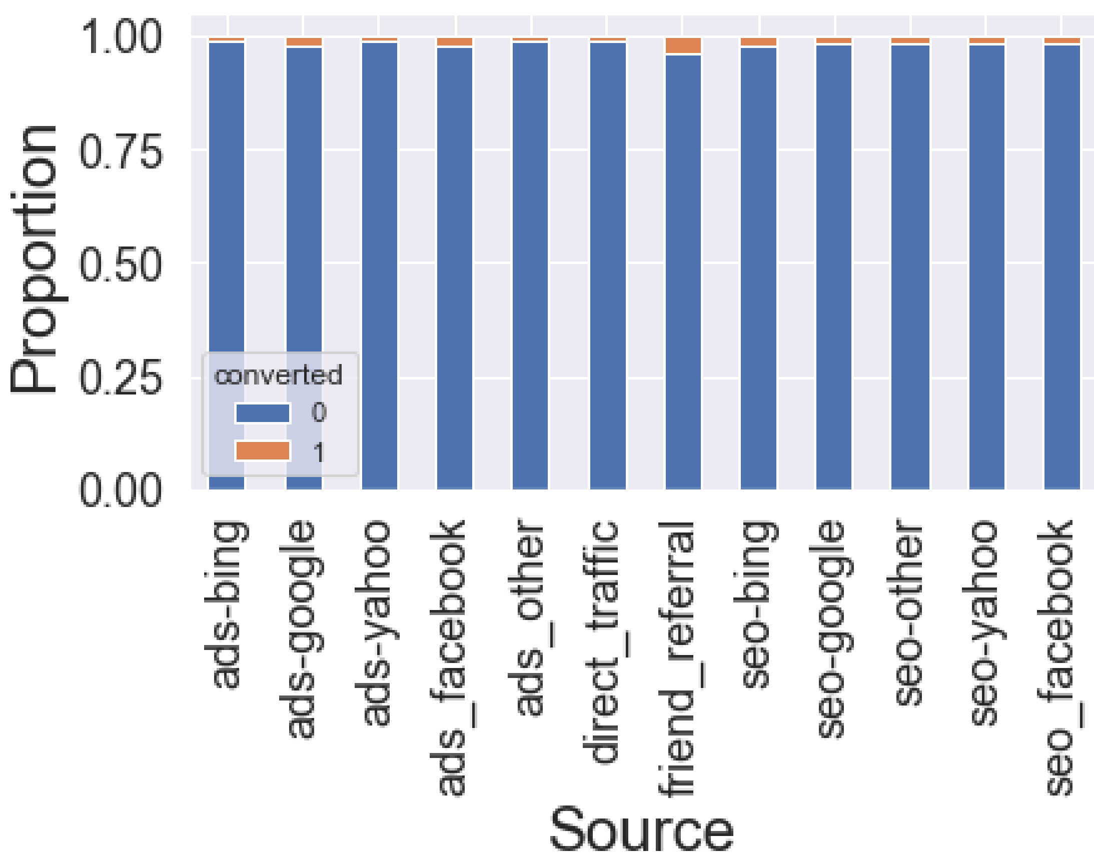


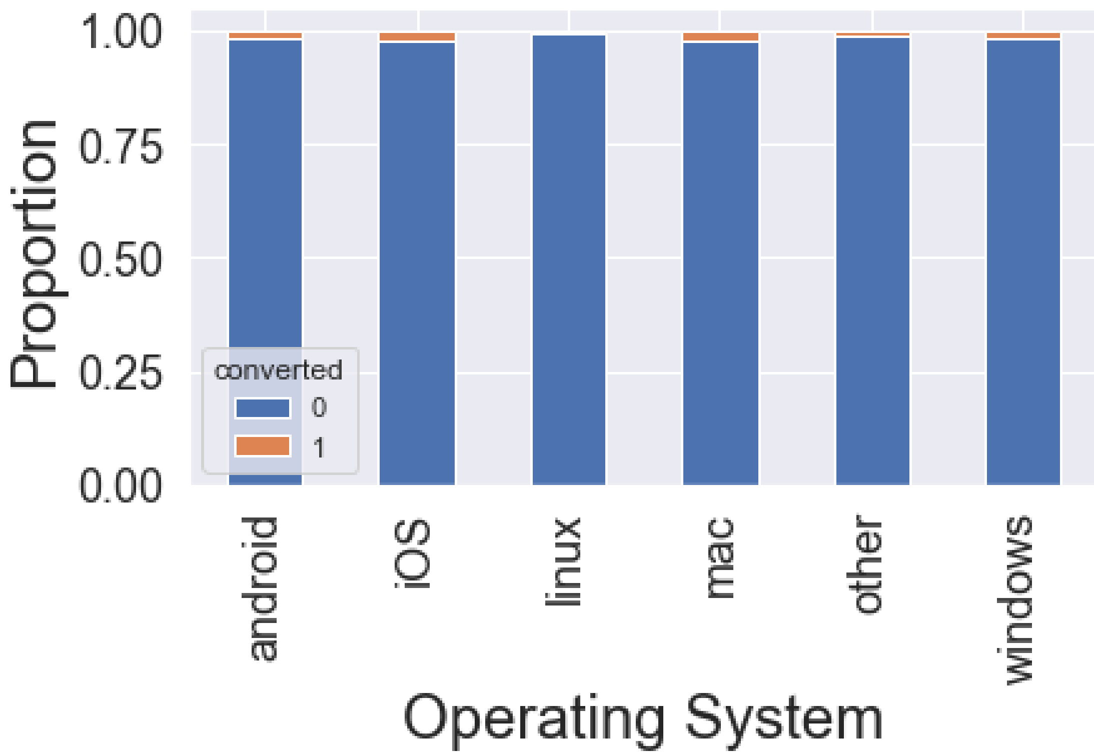
The figure above illustrates day of week and hour of day patterns in user traffic. Interesting patterns emerged where Friday, Saturday, and Sunday were the peaks days, and there seems to be a steady increase in shopping from 7 am to noon, then it drops off to a steady average level in the afternoon until 5 pm, then at night it drops to a steady low level.

Next, differences in conversion rate as a result of different variables was examined.

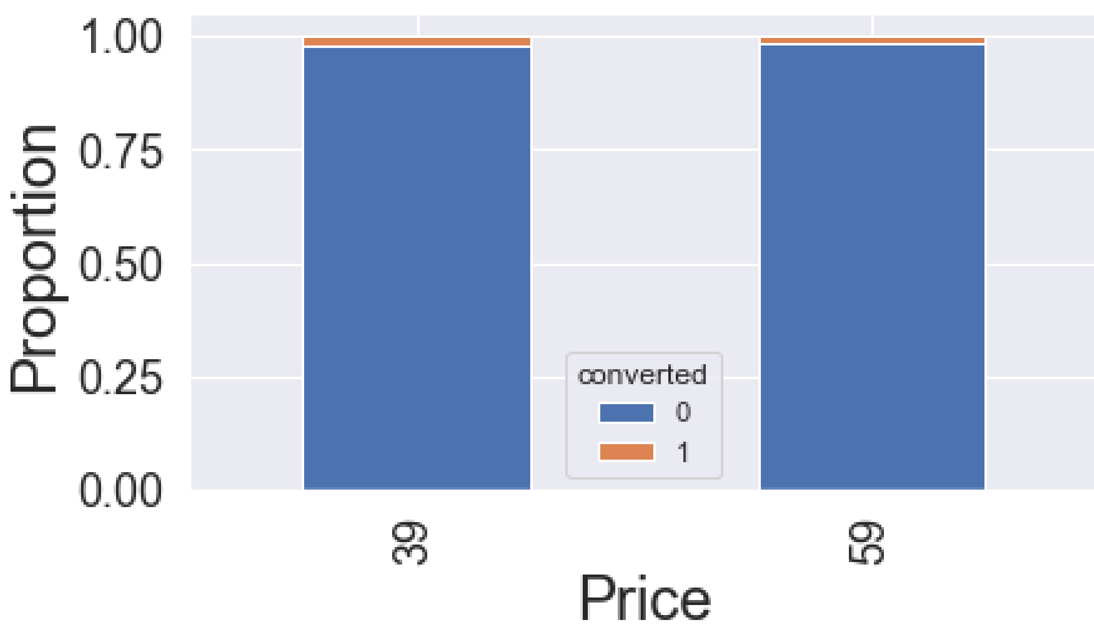
[10]:
```
# Examine stacked bar charts for proportion converted based on categorical␣
↪variables above
legible_graph(6.5, 3.5)

# Define plotting function
def crosstab_plot(series1, series2, xlabel, ylabel):
    table=pd.crosstab(series1, series2)
    table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
```

```
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

# Stacked bar charts for source, device, operating system, price, day of week,␣
↪and hour of day
crosstab_plot(data.source, data.converted, 'Source', 'Proportion')
crosstab_plot(data.operative_system, data.converted, 'Operating System',␣
↪'Proportion')
crosstab_plot(data.price, data.converted, 'Price', 'Proportion')
crosstab_plot(data_cleaned.day_of_week, data_cleaned.converted, 'Day of Week',␣
↪'Proportion')
```

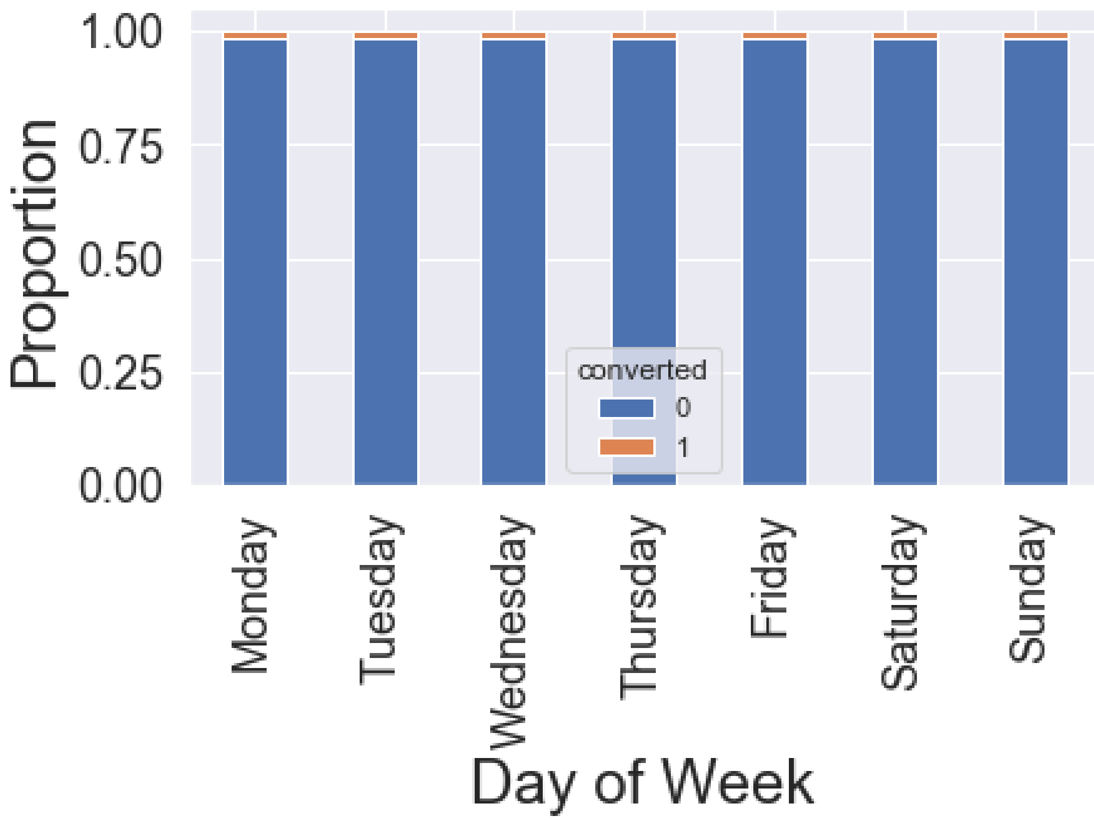




1.00
0.75
0.50
0.25
0.00
Proportion
converted
0
1
android
iOS
linux
mac
other
windows
Operating System



1.00
0.75
0.50
0.25
0.00
Proportion
converted
0
1
39
59
Price



The stacked bar charts above show that there is a higher rate of conversion for visitors coming from friend referrals or Google ads, abnormally low rates of conversion for linux users, and not surprisingly higher conversion for a price of \$39 vs. \$59. There does not seem to be a difference based on day of week.

4 Determining expected revenue differences due to price change

From explorations above, we saw that a lower proportion of consumers who saw a \$59 price converted.

Quantification of this effect revealed that 2% of consumers converted with a \$39 price, while only 1.6% converted with a \$59 price, a 1.25-fold reduction. Since \$59 is ~1.5-fold higher than \$39, this reduction in conversion would still result in a ~1.2-fold increase in revenue.

[11]:
```
# Separate test and non-test classes
control = data[data.test==0] # $39
test = data[data.test==1] # $59

# Determine percentage converted, grouped by price
control_converted = sum(control.converted == 1)/(len(control))*100
test_converted = sum(test.converted == 1)/(len(test))*100
print('Percent converted (control) = ', control_converted)
print('Percent converted (test) = ', test_converted)
```

```
# Determine difference in revenue
print('Fold change in revenue = ', test_converted*59/(control_converted*39))
```

```
Percent converted (control) =  1.9912018961942513
Percent converted (test) =  1.5557765243178079
Fold change in revenue =  1.1820050211136275
```

5 Building a logistic regression model

A logistic regression model was built in order to determine how various factors affected conversion rate.

Categorical variables were changed into dummy variables, the data was split into train and test sets, then a logistic regression classifier was used to model the data using Grid Search to find the optimal cost parameter.

Location data was not used since exploratory analysis indicated location did not affect conversion rate. Also, this allows us to keep data from users that are missing.

[13]:
```
# Create dummy variables (can't have both device and operataive_system since␣
↪they'll be collinear)
cat_vars=['source', 'operative_system', 'day_of_week', 'hour_of_day']

# New dataframe with one-hot encoding of dummy variables
data1=data_cleaned
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(data_cleaned[var], prefix=var)
    data1=data1.join(cat_list)

# Remove extra variables, original variables that were made into dummy␣
↪variables,
# and extra dummy variables (need to remove one of each to prevent␣
↪multicollinearity)
data = data1.drop(['user_id', 'timestamp', 'price', 'city', 'country',
                   'lat', 'long', 'timestamp_ignore_errors', 'source',␣
↪'operative_system',
                   'day_of_week', 'hour_of_day', 'device', 'source_ads_other',␣
↪'operative_system_other',
                   'day_of_week_Monday', 'hour_of_day_0'], axis=1)

# Check that everything is in order
data.head()
```

[13]:
```
   test  converted  source_ads-bing  source_ads-google  source_ads-yahoo  \
0     0          0                0                  0                 0
1     0          0                0                  0                 0
2     0          0                1                  0                 0
```

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

   source_ads_facebook  source_direct_traffic  source_friend_referral  \
0                    1                      0                       0
1                    0                      0                       0
2                    0                      0                       0
3                    0                      1                       0
4                    1                      0                       0

   source_seo-bing  source_seo-google  ...  hour_of_day_14  hour_of_day_15  \
0                0                  0  ...               0               0
1                0                  1  ...               0               0
2                0                  0  ...               0               1
3                0                  0  ...               0               0
4                0                  0  ...               0               0

   hour_of_day_16  hour_of_day_17  hour_of_day_18  hour_of_day_19  \
0               0               0               0               0
1               0               0               0               0
2               0               0               0               0
3               0               0               0               0
4               0               0               0               0

   hour_of_day_20  hour_of_day_21  hour_of_day_22  hour_of_day_23
0               0               0               0               0
1               0               1               0               0
2               0               0               0               0
3               0               0               0               0
4               0               0               0               0

[5 rows x 47 columns]
```

There was a huge imbalance between converted and not converted that was observed before. As determined below, there is a conversion rate of around 1.8%. To correct for this, upsampling of the minority class (converted) was performed.

[14]:
```
# Separate majority and minority classes
majority = data[data.converted==0]
minority = data[data.converted==1]

# Determine extent of imbalanced data
nmaj = len(majority)
nmin = len(minority)
print('Number of non-conversions:', nmaj)
print('Number of conversions:', nmin)
print('Percent converted:', nmin/(nmin+nmaj)*100)
```

```
# Perform upsampling for 'converted'
data_upsampled = resample(minority, replace=True, n_samples=nmaj,␣
 ↪random_state=23)
data_upsampled = pd.concat([data_upsampled, majority]).reset_index(drop=True)

# Double-check upsampling was performed correctly
data_upsampled.converted.value_counts()
```

```
Number of non-conversions: 299900
Number of conversions: 5605
Percent converted: 1.8346671903896827
```

[14]:
```
1    299900
0    299900
Name: converted, dtype: int64
```

Logisitic regression was selected because it is a classification method with results that are easily interpretable. First, the variance inflation factors were calculated. All VIF values were below 10, so multicollinearity should not be an issue.

[15]:
```
# Split data into X and y
y = data_upsampled['converted']
X = data_upsampled.drop(['converted'], axis=1)

# Split data into train and test sets for each
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state=23, stratify=y)

# Add a constant to perform logistic regression using sm package
X_constant = sm.add_constant(X_test)

# Check for no multicollinearity
vif = [variance_inflation_factor(X_constant.values, i) for i in␣
 ↪range(X_constant.shape[1])]
print(pd.DataFrame({'vif': vif[1:]}, index=X.columns).T)
```

```
//anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)

         test  source_ads-bing  source_ads-google  source_ads-yahoo  \
vif  1.002368         1.624344           2.730478          1.227397

     source_ads_facebook  source_direct_traffic  source_friend_referral  \
vif             2.589766               2.442472                1.984368

     source_seo-bing  source_seo-google  source_seo-other  ...  \
```

```
vif         1.086567          1.721006         1.293691  ...

     hour_of_day_14  hour_of_day_15  hour_of_day_16  hour_of_day_17  \
vif        3.879184        3.711961         3.92912         3.76054

     hour_of_day_18  hour_of_day_19  hour_of_day_20  hour_of_day_21  \
vif        3.852334        2.018249        1.923262        1.926235

     hour_of_day_22  hour_of_day_23
vif        1.970213        1.971035

[1 rows x 46 columns]
```

The coefficients and p-values from performing logistic regression using the statsmodels package were examined to determine their effect and significance for conversion.

[16]:
```python
logit_model=sm.Logit(y_test, X_constant)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.670309
         Iterations 5
                             Results: Logit
=========================================================================
Model:                 Logit              Pseudo R-squared:   0.033
Dependent Variable:    converted          AIC:                160914.4412
Date:                  2019-10-17 08:48   BIC:                161370.1022
No. Observations:      119960             Log-Likelihood:     -80410.
Df Model:              46                 LL-Null:            -83150.
Df Residuals:          119913             LLR p-value:        0.0000
Converged:             1.0000             Scale:              1.0000
No. Iterations:        5.0000
-------------------------------------------------------------------------
                          Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
-------------------------------------------------------------------------
const                    -0.6106   0.0613  -9.9591 0.0000 -0.7308 -0.4904
test                     -0.2554   0.0126 -20.3308 0.0000 -0.2800 -0.2308
source_ads-bing          -0.2209   0.0316  -6.9845 0.0000 -0.2829 -0.1589
source_ads-google         0.4139   0.0241  17.1801 0.0000  0.3667  0.4611
source_ads-yahoo          0.0014   0.0452   0.0317 0.9747 -0.0871  0.0899
source_ads_facebook       0.3993   0.0245  16.2983 0.0000  0.3513  0.4473
source_direct_traffic    -0.1637   0.0252  -6.5055 0.0000 -0.2130 -0.1144
source_friend_referral    1.0555   0.0283  37.2436 0.0000  0.9999  1.1110
source_seo-bing           0.4946   0.0692   7.1452 0.0000  0.3589  0.6303
source_seo-google         0.2196   0.0298   7.3610 0.0000  0.1611  0.2780
source_seo-other          0.1084   0.0406   2.6676 0.0076  0.0288  0.1880
source_seo-yahoo          0.0942   0.0460   2.0468 0.0407  0.0040  0.1843
source_seo_facebook       0.1061   0.0309   3.4346 0.0006  0.0455  0.1666
```

```
operative_system_android  0.1512   0.0312   4.8425 0.0000  0.0900  0.2124
operative_system_iOS      0.5801   0.0303  19.1759 0.0000  0.5208  0.6394
operative_system_linux   -0.4895   0.0711  -6.8876 0.0000 -0.6288 -0.3502
operative_system_mac      0.6659   0.0346  19.2393 0.0000  0.5981  0.7337
operative_system_windows  0.2828   0.0304   9.3087 0.0000  0.2233  0.3424
day_of_week_Tuesday      -0.1140   0.0343  -3.3261 0.0009 -0.1811 -0.0468
day_of_week_Wednesday    -0.0594   0.0344  -1.7294 0.0837 -0.1268  0.0079
day_of_week_Thursday     -0.0263   0.0296  -0.8885 0.3743 -0.0842  0.0317
day_of_week_Friday       -0.0082   0.0265  -0.3110 0.7558 -0.0602  0.0437
day_of_week_Saturday     -0.0185   0.0270  -0.6854 0.4931 -0.0716  0.0345
day_of_week_Sunday        0.0160   0.0279   0.5735 0.5663 -0.0387  0.0707
hour_of_day_1             0.0930   0.0628   1.4803 0.1388 -0.0301  0.2162
hour_of_day_2             0.1132   0.0624   1.8132 0.0698 -0.0092  0.2356
hour_of_day_3            -0.0950   0.0636  -1.4932 0.1354 -0.2198  0.0297
hour_of_day_4             0.3648   0.0613   5.9511 0.0000  0.2446  0.4849
hour_of_day_5            -0.1077   0.0641  -1.6803 0.0929 -0.2334  0.0179
hour_of_day_6             0.0696   0.0629   1.1076 0.2680 -0.0536  0.1929
hour_of_day_7             0.0692   0.0547   1.2660 0.2055 -0.0379  0.1764
hour_of_day_8             0.2850   0.0512   5.5635 0.0000  0.1846  0.3854
hour_of_day_9             0.0400   0.0500   0.7995 0.4240 -0.0580  0.1380
hour_of_day_10            0.0998   0.0490   2.0389 0.0415  0.0039  0.1958
hour_of_day_11            0.0105   0.0484   0.2163 0.8287 -0.0844  0.1053
hour_of_day_12            0.0669   0.0478   1.4006 0.1613 -0.0267  0.1606
hour_of_day_13            0.0582   0.0514   1.1341 0.2568 -0.0424  0.1589
hour_of_day_14            0.1600   0.0514   3.1149 0.0018  0.0593  0.2606
hour_of_day_15            0.0528   0.0518   1.0205 0.3075 -0.0486  0.1543
hour_of_day_16            0.1438   0.0513   2.8056 0.0050  0.0434  0.2443
hour_of_day_17            0.1454   0.0516   2.8147 0.0049  0.0441  0.2466
hour_of_day_18            0.0953   0.0514   1.8530 0.0639 -0.0055  0.1960
hour_of_day_19            0.2048   0.0624   3.2839 0.0010  0.0826  0.3271
hour_of_day_20            0.0603   0.0639   0.9445 0.3449 -0.0649  0.1855
hour_of_day_21            0.0078   0.0638   0.1223 0.9026 -0.1173  0.1329
hour_of_day_22            0.0257   0.0631   0.4078 0.6834 -0.0979  0.1494
hour_of_day_23            0.1822   0.0631   2.8865 0.0039  0.0585  0.3060
=========================================================================
```

Logistic regression was then performed again using sklearn. Even though other methods such as ensemble methods might be able to provide more accuracy, logistic regression was able to achieve a decent performance with an AUC of 0.62.

For conversion rate, we would like to err on the side of trying to obtain as many positives as possible so we know who to potentially target for more conversions. The original recall was low at 56%, so the probability threshold was changed to reduce false positives and allow a larger margin of error for false negatives. Doing this, the recall was adjusted to 81%, while precision was maintained at 55%. Also, accuracy only dropped from 58% to 57%.

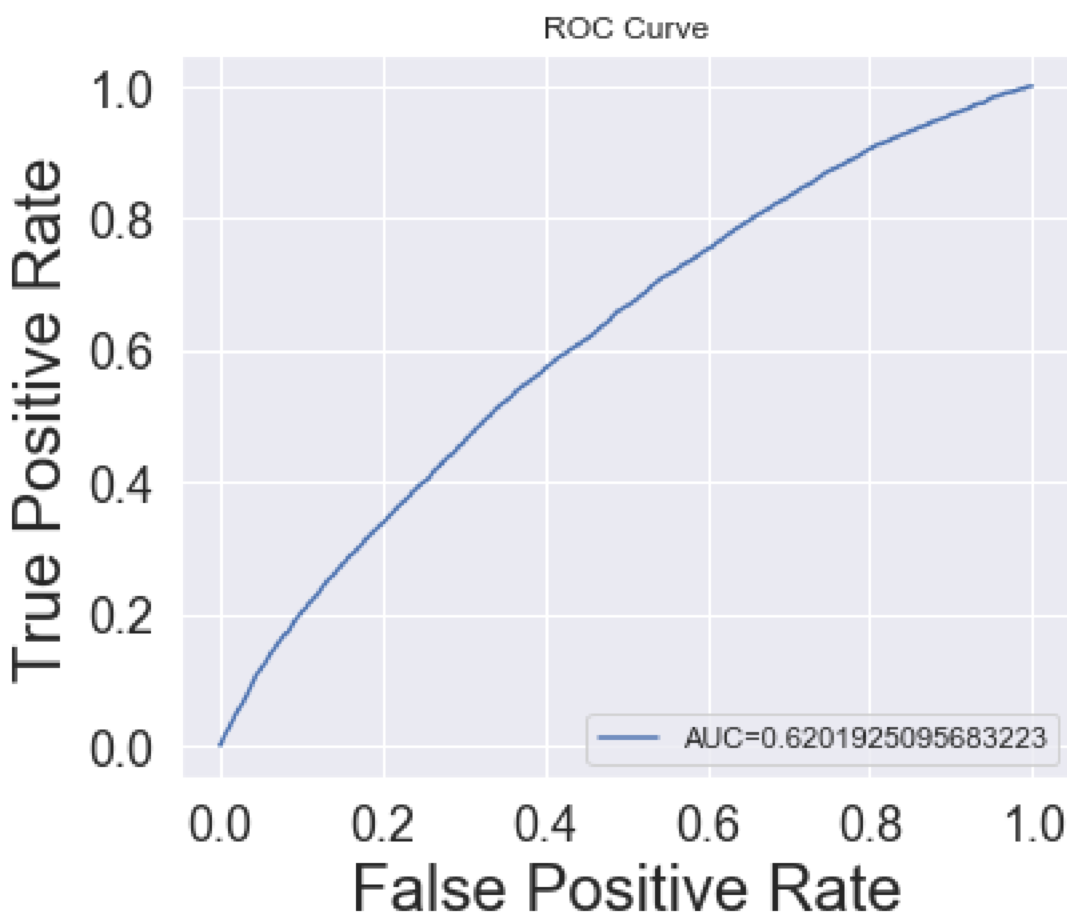
[17]:
```python
# Instantiate logistic regression classifier and parameters to tune
lr = LogisticRegression(solver='newton-cg', random_state=23)
```

```
lr_par= {'C': [0.2, 0.5, 1]}

# Use GridSearch on the training set to tune parameters using 5-fold␣
↪cross-validation
lr_cv = GridSearchCV(lr, lr_par, cv=5, iid=False)
lr_cv.fit(X_train, y_train)

# Output the results
# Print the tuned parameters
print(lr_cv.best_params_)

# Print the training accuracies
print("Training accuracy: ", lr_cv.cv_results_['mean_test_score'])
print("Std. dev.: ", lr_cv.cv_results_['std_test_score'])

# Print the test accuracy
print("Test accuracy: ", accuracy_score(y_test, lr_cv.predict(X_test)))

# Print classification report
print(classification_report(y_test, lr_cv.predict(X_test))) # Only 56% recall
```

```
{'C': 1}
Training accuracy:  [0.58362162 0.58359036 0.58362371]
Std. dev.:  [0.00190395 0.00183554 0.00180866]
Test accuracy:  0.5849783261087029
              precision    recall  f1-score   support

           0       0.58      0.61      0.60     59980
           1       0.59      0.56      0.57     59980

    accuracy                           0.58    119960
   macro avg       0.59      0.58      0.58    119960
weighted avg       0.59      0.58      0.58    119960
```

[18]:
```
# Create ROC curve
y_pred_proba = lr_cv.predict_proba(X_test)[::,1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)

legible_graph(6,5)
plt.plot(fpr, tpr, label="AUC="+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc=4)
plt.show()
```

```
# Change probability cut-off for determining conversion rate (increase recall␣
↪to 81%)
y_predict = pd.DataFrame(lr_cv.predict_proba(X_test)[:,1], # 43% threshold␣
↪instead of 50%
                         columns=['prediction'])['prediction'].apply(lambda x:␣
↪0 if x<0.43 else 1)
print(classification_report(y_test, y_predict)) # Recall = 0.81, precision = 0.
↪55
print(accuracy_score(y_test, y_predict)) # Accuracy = 57%
```




```
              precision    recall  f1-score   support

           0       0.63      0.34      0.44     59980
           1       0.55      0.81      0.65     59980

    accuracy                           0.57    119960
```

```
   macro avg       0.59      0.57      0.55    119960
weighted avg       0.59      0.57      0.55    119960

0.57148216072024
```

6 Model evaluation

Since there are a lot of features after one-hot encoding, if there was more time, it would be recommended to perform regularization to reduce the multiple comparisons effect.

Based on p-values and coefficients of the variables, we are able to get a general sense of the effect of each variable (for example, a negative coefficient indicates lower likelihood of conversion). The main findings match what was observed from the exploratory data analysis.

They include:

- People who see a price of $59 are less likely to convert.
- More conversion was observed with friend referral, Google and Facebook ads, and Bing search.
- Mac and iOS operating systems also resulted in higher conversion, while Linux resulted in lower conversion
- For time of day, there seems to be higher conversions at 8 am, 2 pm, 4-5 pm, 7 pm, and 11 pm

7 Recommendations

For the product team to improve conversion rates, the following general actions are recommended:

- Encourage customers to recommend the product to friends
- Focus more on Google and Facebook advertisements
  - If the higher conversion rate is already due to focus in these areas, then focus in other areas to gain more conversion across the board
- Determine why Mac and iOS operating systems result in higher conversion
  - Does the software work better on these operating systems? If so, improve software for other operating systems(especially since Windows users are most prevalent)
  - Are there issues with the purchasing website on other operating systems? If so, improve website design for the other systems