HW2

February 22, 2019

1 Looking at the data

In [580]:
```
import numpy as np
import pandas as pd
import csv
```

In [581]:
```
filename = 'train.csv'
df = pd.read_csv(filename, header = 0, delimiter = ',')
```

In [582]:
```
df.shape
```

Out[582]:
```
(18000, 12)
```

In [583]:
```
df.head()
```

Out[583]:
```
  COLLEGE  INCOME  OVERAGE  LEFTOVER   HOUSE  HANDSET_PRICE  \
0    zero   28987      191        20  175953            217
1    zero   45201        0         0  841177            160
2     one  110663        0         0  902611            529
3    zero   40646      169        71  772903            146
4     one  132530        0        10  196535            559

   OVER_15MINS_CALLS_PER_MONTH  AVERAGE_CALL_DURATION REPORTED_SATISFACTION  \
0                           28                      5                 unsat
1                            1                     15                 unsat
2                            1                     13           very_unsat
3                           24                      2           very_unsat
4                            0                      6           very_unsat

  REPORTED_USAGE_LEVEL CONSIDERING_CHANGE_OF_PLAN  LEAVE
0          very_little                considering      1
1                  avg     actively_looking_into_it      0
2                 high                    perhaps      0
3               little                considering      1
4                  avg                    perhaps      0
```

In [584]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18000 entries, 0 to 17999
Data columns (total 12 columns):
COLLEGE                        18000 non-null object
INCOME                         18000 non-null int64
OVERAGE                        18000 non-null int64
LEFTOVER                       18000 non-null int64
HOUSE                          18000 non-null int64
HANDSET_PRICE                  18000 non-null int64
OVER_15MINS_CALLS_PER_MONTH    18000 non-null int64
AVERAGE_CALL_DURATION          18000 non-null int64
REPORTED_SATISFACTION          18000 non-null object
REPORTED_USAGE_LEVEL           18000 non-null object
CONSIDERING_CHANGE_OF_PLAN     18000 non-null object
LEAVE                          18000 non-null int64
dtypes: int64(8), object(4)
memory usage: 1.6+ MB
```

In [585]: df.dtypes

```
Out[585]: COLLEGE                        object
          INCOME                          int64
          OVERAGE                         int64
          LEFTOVER                        int64
          HOUSE                           int64
          HANDSET_PRICE                   int64
          OVER_15MINS_CALLS_PER_MONTH     int64
          AVERAGE_CALL_DURATION           int64
          REPORTED_SATISFACTION          object
          REPORTED_USAGE_LEVEL           object
          CONSIDERING_CHANGE_OF_PLAN     object
          LEAVE                           int64
          dtype: object
```

In [586]: df.isnull().any()

```
Out[586]: COLLEGE                        False
          INCOME                         False
          OVERAGE                        False
          LEFTOVER                       False
          HOUSE                          False
          HANDSET_PRICE                  False
          OVER_15MINS_CALLS_PER_MONTH    False
          AVERAGE_CALL_DURATION          False
          REPORTED_SATISFACTION          False
          REPORTED_USAGE_LEVEL           False
          CONSIDERING_CHANGE_OF_PLAN     False
```

```
LEAVE                           False
dtype: bool
```

In [587]: `df.shape`

Out[587]: `(18000, 12)`

In [588]: `df['LEAVE'].value_counts()`

Out[588]:
```
0    9106
1    8894
Name: LEAVE, dtype: int64
```

In [589]: `df.groupby('LEAVE').mean()`

Out[589]:
```
              INCOME     OVERAGE   LEFTOVER         HOUSE  HANDSET_PRICE  \
LEAVE
0       76316.473534   65.756424  22.261146  546953.57852     370.648693
1       84604.298403  106.852822  25.580391  438319.60434     410.641444

       OVER_15MINS_CALLS_PER_MONTH  AVERAGE_CALL_DURATION
LEAVE
0                         6.203931               6.049418
1                         9.853047               5.961660
```

In [590]: `df['COLLEGE'] = df['COLLEGE'].astype(str)`

In [591]: `df.groupby('LEAVE').mean()`

Out[591]:
```
              INCOME     OVERAGE   LEFTOVER         HOUSE  HANDSET_PRICE  \
LEAVE
0       76316.473534   65.756424  22.261146  546953.57852     370.648693
1       84604.298403  106.852822  25.580391  438319.60434     410.641444

       OVER_15MINS_CALLS_PER_MONTH  AVERAGE_CALL_DURATION
LEAVE
0                         6.203931               6.049418
1                         9.853047               5.961660
```

for x in df['COLLEGE']: if x is 'one': df['COLLEGE'].str.replace('one','1') else: df['COLLEGE'].str.replace('zero','0')

In [592]: `df.groupby('REPORTED_SATISFACTION').mean()`

Out[592]:
```
                             INCOME    OVERAGE   LEFTOVER          HOUSE  \
REPORTED_SATISFACTION
avg                    79727.667035  84.640487  23.596792  499801.858960
sat                    82011.881210  85.597192  24.424406  479487.074514
unsat                  80104.956010  87.032222  23.908378  483439.400392
very_sat               80260.010577  83.657779  24.135963  494286.926620
```

```
very_unsat              80626.246403  87.522978  23.758067  497675.340411

                       HANDSET_PRICE  OVER_15MINS_CALLS_PER_MONTH  \
REPORTED_SATISFACTION
avg                       390.062500                     7.637168
sat                       404.425486                     7.820734
unsat                     387.047913                     8.112076
very_sat                  391.094094                     7.859850
very_unsat                389.926247                     8.165386

                       AVERAGE_CALL_DURATION     LEAVE
REPORTED_SATISFACTION
avg                                 6.055310  0.476770
sat                                 5.910367  0.461123
unsat                               6.016251  0.509106
very_sat                            5.948215  0.490965
very_unsat                          6.037575  0.497276
```

In [593]:
```
df.groupby('COLLEGE').mean()
```

Out[593]:
```
                INCOME    OVERAGE   LEFTOVER          HOUSE  HANDSET_PRICE  \
COLLEGE
one       80917.668319  85.720864  23.744657  492915.042300     392.611919
zero      79896.642569  86.410334  24.060525  493643.924344     388.168684

         OVER_15MINS_CALLS_PER_MONTH  AVERAGE_CALL_DURATION     LEAVE
COLLEGE
one                         7.946464                6.00727  0.501322
zero                        8.068594                6.00482  0.486774
```

In [594]:
```
df.groupby('REPORTED_USAGE_LEVEL').mean()
```

Out[594]:
```
                             INCOME    OVERAGE   LEFTOVER          HOUSE  \
REPORTED_USAGE_LEVEL
avg                    82197.858903  85.522956  23.920493  498221.000000
high                   80415.259259  84.462134  23.069652  490782.482034
little                 80262.583451  86.673680  24.025699  491718.856538
very_high              80422.684726  86.971062  23.825936  495791.617276
very_little            80246.483702  84.646685  24.164088  493156.289779

                      HANDSET_PRICE  OVER_15MINS_CALLS_PER_MONTH  \
REPORTED_USAGE_LEVEL
avg                      398.847704                     7.861142
high                     382.181316                     8.321172
little                   393.962158                     8.003671
very_high                389.745213                     7.993908
very_little              386.333149                     7.909116

                      AVERAGE_CALL_DURATION     LEAVE
```

```
REPORTED_USAGE_LEVEL
avg                               5.879059  0.483763
high                              6.159204  0.485904
little                            6.013132  0.490822
very_high                         6.046127  0.501088
very_little                       5.896133  0.498343
```

In [595]:
```
df.groupby('CONSIDERING_CHANGE_OF_PLAN').mean()
```

Out[595]:
```
                                 INCOME    OVERAGE   LEFTOVER          HOUSE  \
CONSIDERING_CHANGE_OF_PLAN
actively_looking_into_it   80421.375615  88.368456  24.113199  489830.130425
considering                80222.750243  86.211910  24.167386  494897.659802
never_thought              79211.485003  84.709677  22.421053  498432.727221
no                         81067.017024  83.648545  23.696046  492625.321527
perhaps                    81532.347966  85.851178  24.438972  490076.694861

                           HANDSET_PRICE  OVER_15MINS_CALLS_PER_MONTH  \
CONSIDERING_CHANGE_OF_PLAN
actively_looking_into_it      391.430649                     8.095749
considering                   388.797829                     8.097676
never_thought                 387.312960                     7.811545
no                            392.149918                     7.833333
perhaps                       396.996788                     7.931478

                           AVERAGE_CALL_DURATION     LEAVE
CONSIDERING_CHANGE_OF_PLAN
actively_looking_into_it                5.942953  0.495973
considering                             5.990260  0.490886
never_thought                           6.111488  0.484437
no                                      6.012081  0.496980
perhaps                                 6.206638  0.517131
```

Observations on features that seem to have an effect: Categorical Data
1. college: seems to barely make a difference, .02 more likely to leave if person went to college 2. REPORTED_SATISFACTION: unsat has slightly over 50% chance of leaving and very_unsat has a bit lower than half chance of leaving. These are the 2 highest chances of leaving based on sat ranking. ALSO, sat ranking may be tied with the OVERAGE. These 2 categories had the highest overage, so maybe there are high overcharge fees or they just got upset about being overcharged. 3. REPORTED_USAGE_LEVEL: very high usage has slightly over 50% churn (highest in group) followed by very little. So the rates are probably expensive if you're a heavy user or too pricey to be for a casual user 4. CONSIDERING_CHANGE_OF_PLAN: Those who had the highest overage are most looking into changing plans. Those with highest overage also happen to have lowest estimated house value. HOWEVER!!! Those that chose to leave are "perhaps" (51.7) and the next are "no"(49.69) and then "actively looking into it"(.4959). Maybe if we were to average all these values on a scale where perhaps is 3 and no is 2, the values in between the range of 2-3, this could tell us something.

Number Data 1. INCOME: those who left had higher income than those who didnt 2. OVERAGE: those who left had ALMOST DOUBLE overage than those who didn't 3. LEFTOVER: those who left had more leftover minutes per month 4. HOUSE: those who left had cheaper house value 5. HANDSET $$: those who left had more expensive handsets 6. OVER 15 MIN: those who left had more over 15 min calls (maybe related to overage) 7. AVG CALL DURATION: those who left had slightly lover avg call duration but both are pretty close to 6 min

why are income and house indirectly related here?

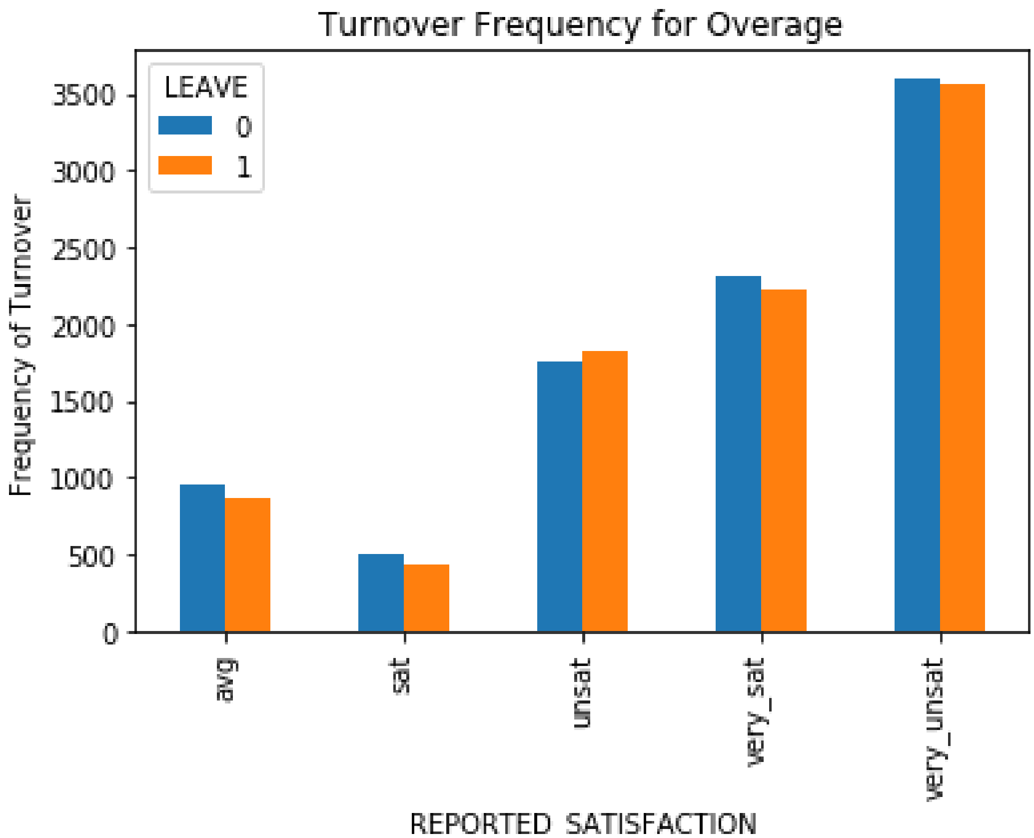
correlation between high overage and cheaper house value correlation between income and handset price correlation between satisfaction and high overage = higher overage, less satisfaction

Features to include: ***OVERAGE –> higher overage, more chance of leaving LEFTOVER –> those who had more leftover minutes, less usage = higher chance of leaving House value? –> those who had cheaper houses AND high overage are more likely to leave

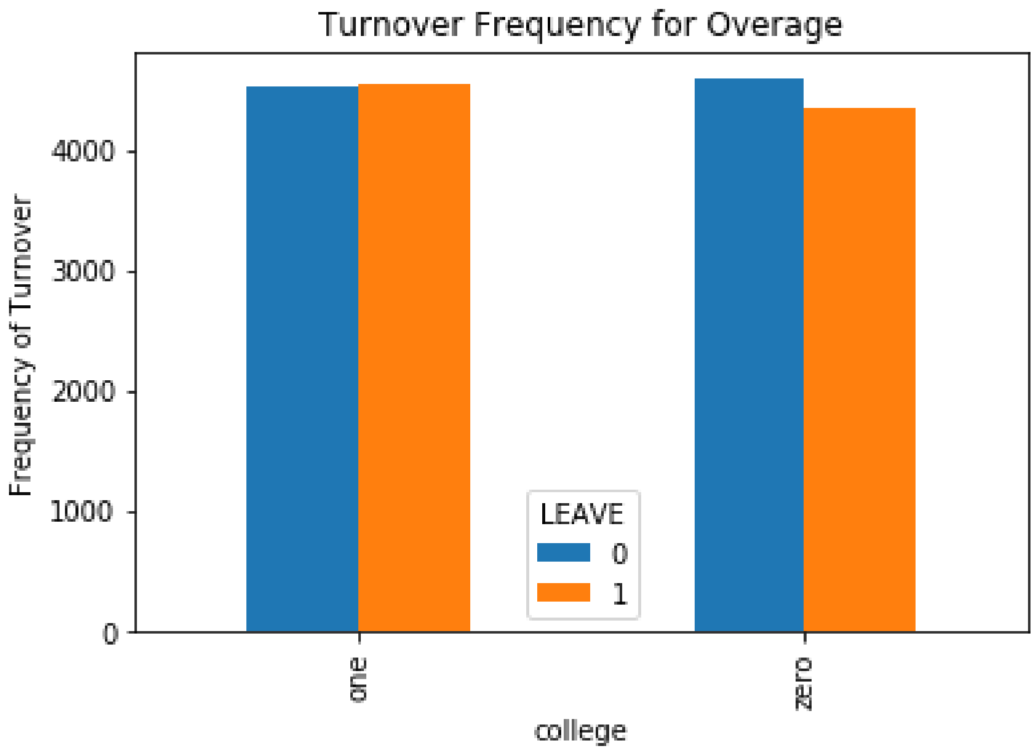
In [596]:
```
df.groupby('LEAVE').mean()
```

Out[596]:
```
             INCOME     OVERAGE   LEFTOVER         HOUSE  HANDSET_PRICE  \
LEAVE
0      76316.473534   65.756424  22.261146  546953.57852     370.648693
1      84604.298403  106.852822  25.580391  438319.60434     410.641444

       OVER_15MINS_CALLS_PER_MONTH  AVERAGE_CALL_DURATION
LEAVE
0                         6.203931               6.049418
1                         9.853047               5.961660
```

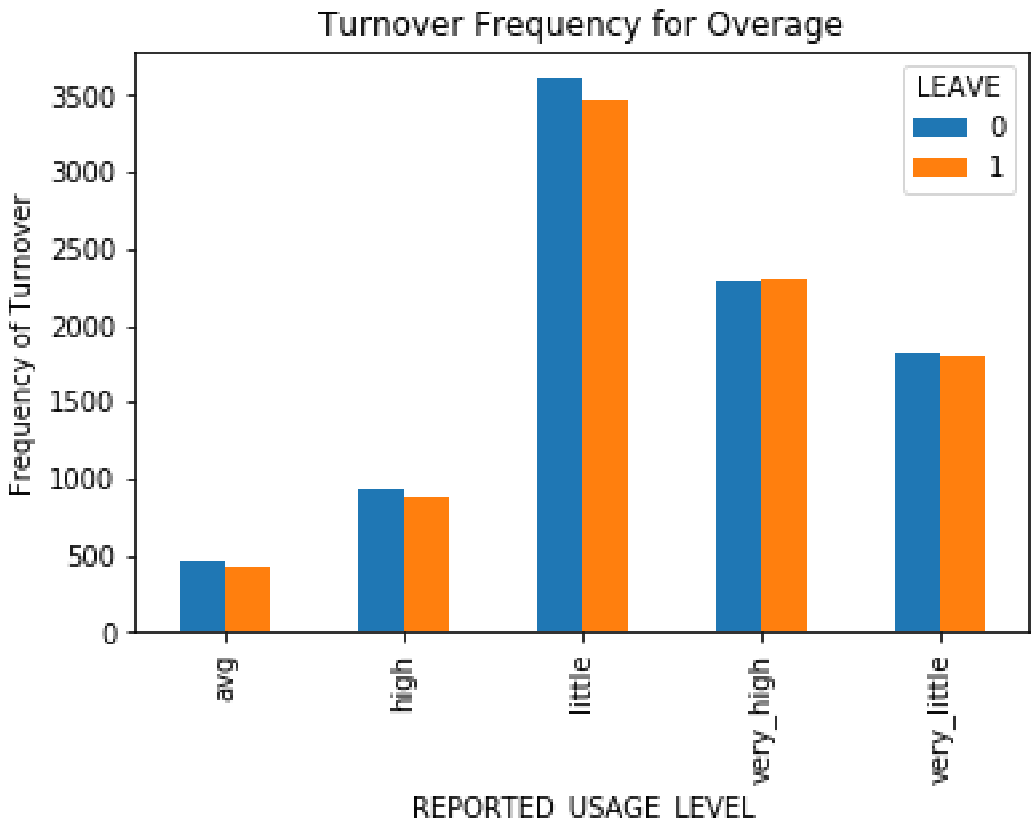
In [597]:
```
%matplotlib inline
import matplotlib.pyplot as plt
pd.crosstab(df['REPORTED_SATISFACTION'],df['LEAVE']).plot(kind='bar')
plt.title('Turnover Frequency for Overage')
plt.xlabel('REPORTED_SATISFACTION')
plt.ylabel('Frequency of Turnover')
plt.savefig('department_bar_chart')
```



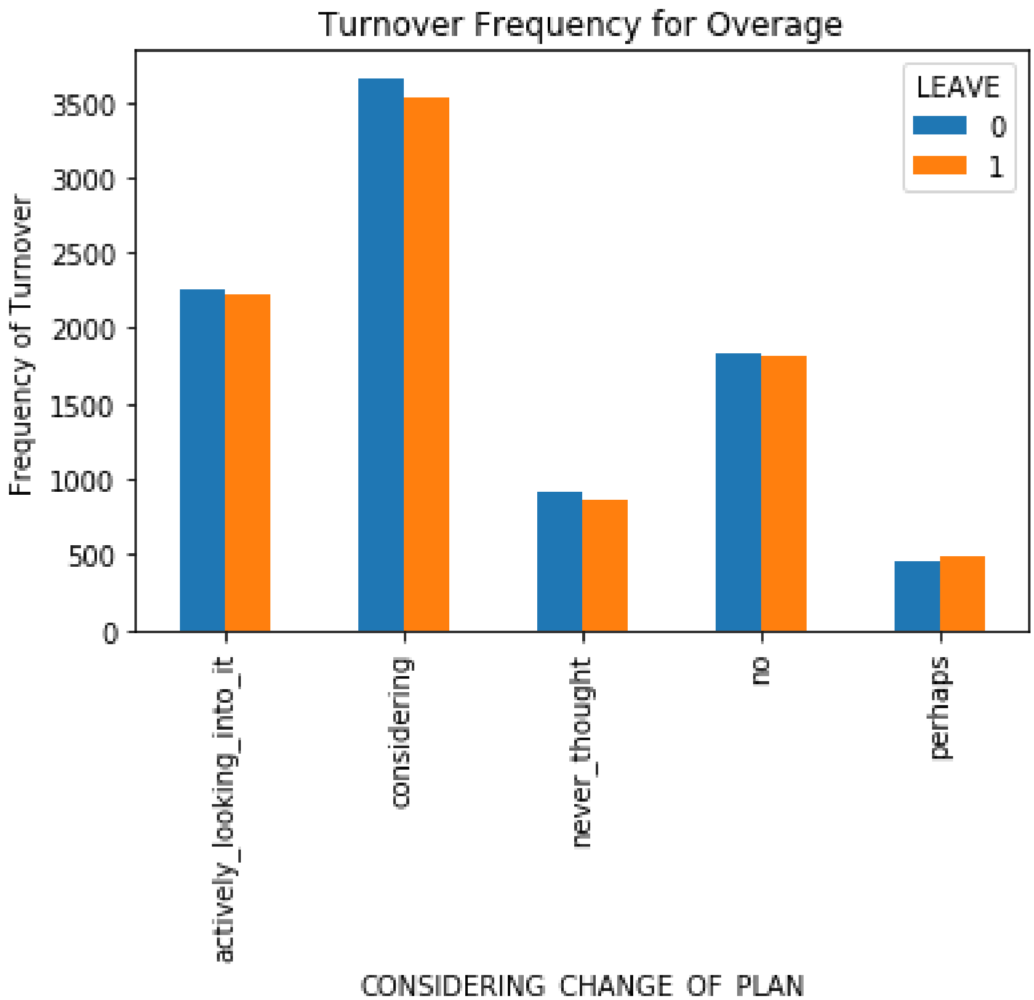
In [598]:
```
%matplotlib inline
import matplotlib.pyplot as plt
pd.crosstab(df['COLLEGE'],df['LEAVE']).plot(kind='bar')
plt.title('Turnover Frequency for Overage')
plt.xlabel('college')
plt.ylabel('Frequency of Turnover')
plt.savefig('department_bar_chart')
```




```
In [599]: %matplotlib inline
          import matplotlib.pyplot as plt
          pd.crosstab(df['REPORTED_USAGE_LEVEL'],df['LEAVE']).plot(kind='bar')
          plt.title('Turnover Frequency for Overage')
          plt.xlabel('REPORTED_USAGE_LEVEL')
          plt.ylabel('Frequency of Turnover')
          plt.savefig('department_bar_chart')
```




```
In [600]: %matplotlib inline
          import matplotlib.pyplot as plt
          pd.crosstab(df['CONSIDERING_CHANGE_OF_PLAN'],df['LEAVE']).plot(kind='bar')
          plt.title('Turnover Frequency for Overage')
          plt.xlabel('CONSIDERING_CHANGE_OF_PLAN')
          plt.ylabel('Frequency of Turnover')
          plt.savefig('department_bar_chart')
```

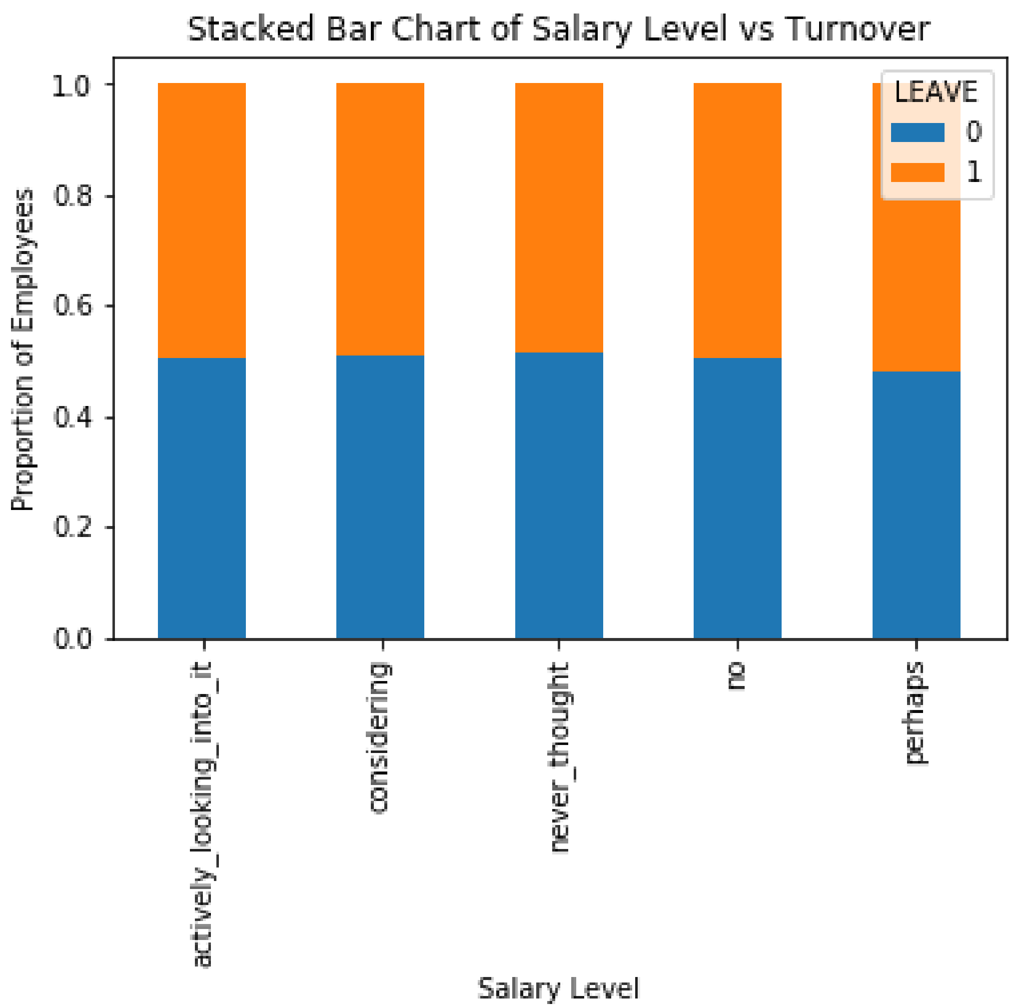


Categorical Data

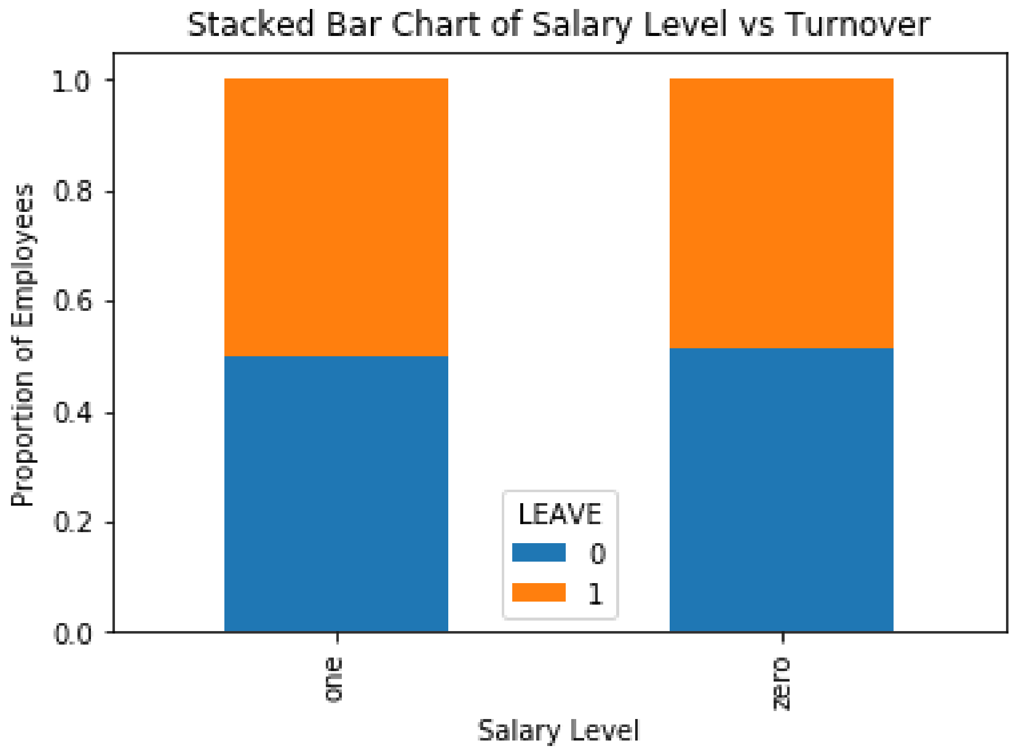
```
REPORTED_SATISFACTION: highest reportings of very unsatisfied followed by very satisfied
COLLEGE: more variability in people staying vs leaving in those who didnt go to college
REPORTED_USAGE_LEVEL: most variety in people who report little, also most people by far report
CONSIDERING_CHANGE_OF_PLAN: most people report considering, which also has most variance, then
```

Conclusions: Don't really trust the categorical data.

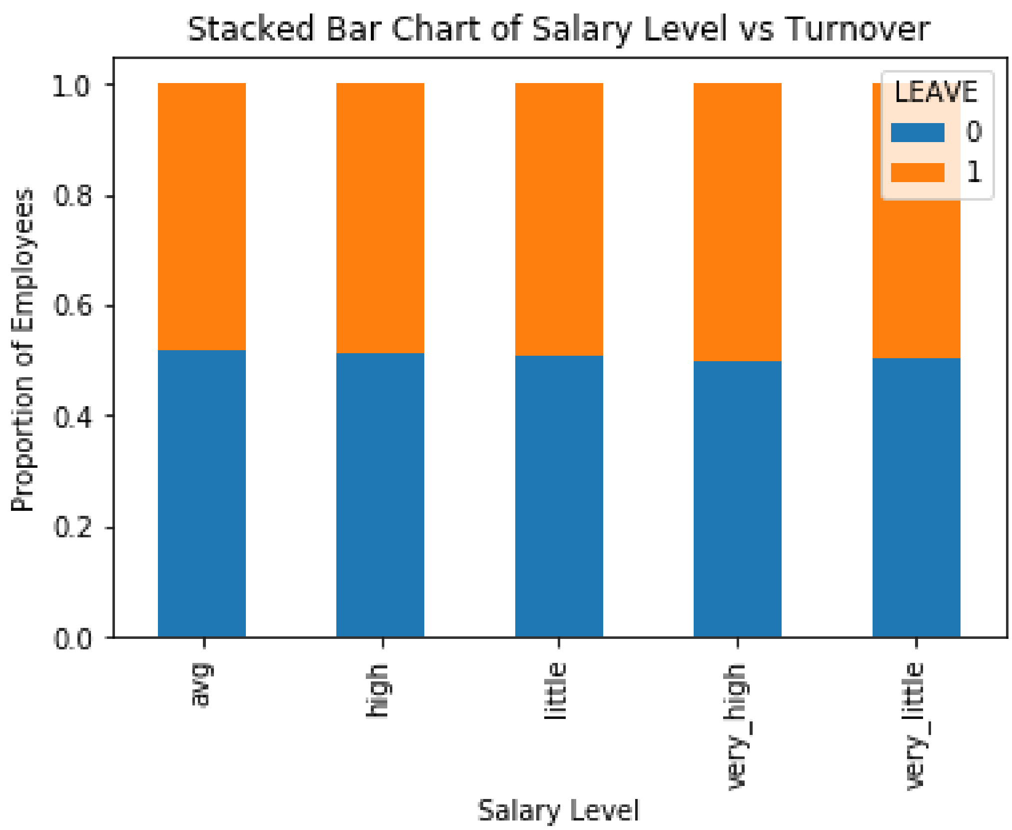
```
In [601]: table=pd.crosstab(df['CONSIDERING_CHANGE_OF_PLAN'],df['LEAVE'])
          table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
          plt.title('Stacked Bar Chart of Salary Level vs Turnover')
          plt.xlabel('Salary Level')
          plt.ylabel('Proportion of Employees')
          plt.savefig('salary_bar_chart')
```



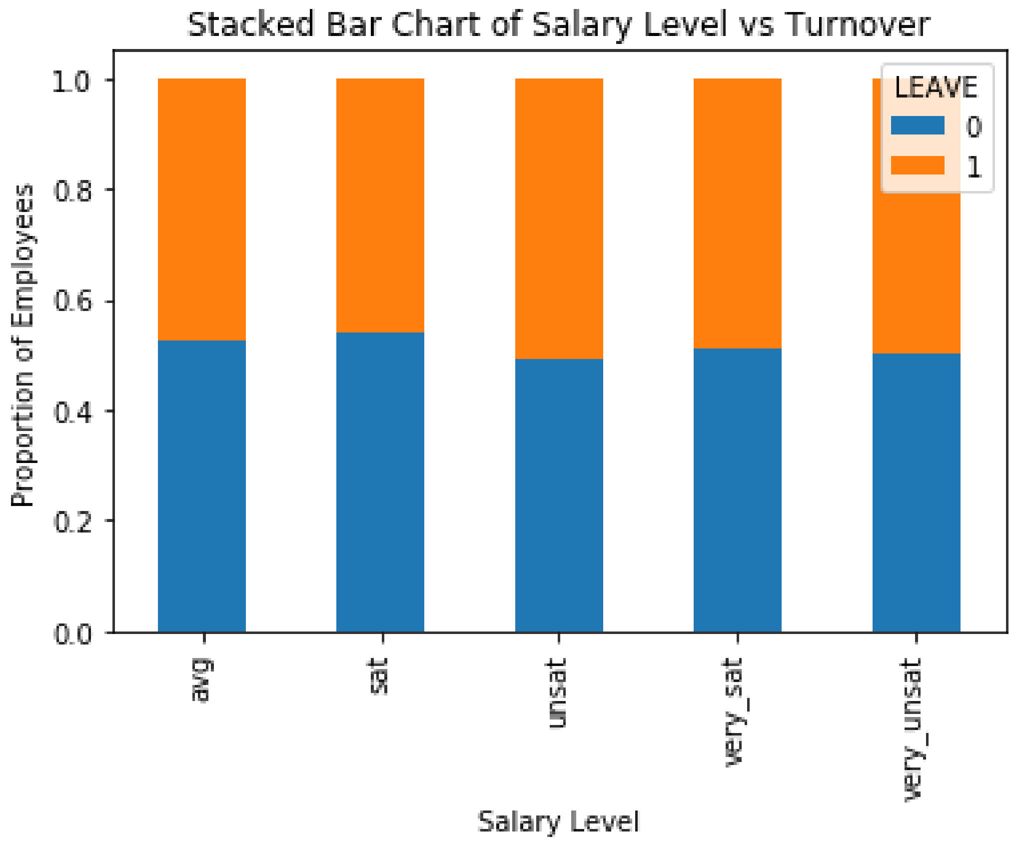

```
In [602]: table=pd.crosstab(df['COLLEGE'],df['LEAVE'])
          table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
          plt.title('Stacked Bar Chart of Salary Level vs Turnover')
          plt.xlabel('Salary Level')
          plt.ylabel('Proportion of Employees')
          plt.savefig('salary_bar_chart')
```




```
In [603]: table=pd.crosstab(df['REPORTED_USAGE_LEVEL'],df['LEAVE'])
          table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
          plt.title('Stacked Bar Chart of Salary Level vs Turnover')
          plt.xlabel('Salary Level')
          plt.ylabel('Proportion of Employees')
          plt.savefig('salary_bar_chart')
```



In [604]:
```
table=pd.crosstab(df['REPORTED_SATISFACTION'],df['LEAVE'])
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Salary Level vs Turnover')
plt.xlabel('Salary Level')
plt.ylabel('Proportion of Employees')
plt.savefig('salary_bar_chart')
```



2 Manipulating data for training model

In [605]:
```
cat_vars=['COLLEGE','REPORTED_SATISFACTION', 'REPORTED_USAGE_LEVEL', 'CONSIDERING_CHA
for var in cat_vars:
    cat_list='var'+'_'+var
    cat_list = pd.get_dummies(df[var], prefix=var)
    df1=df.join(cat_list)
    df=df1
```

In [606]:
```
df.drop(df.columns[[0,8,9,10]], axis=1, inplace=True) #drop categorical data
df.columns.values
```

Out[606]:
```
array(['INCOME', 'OVERAGE', 'LEFTOVER', 'HOUSE', 'HANDSET_PRICE',
       'OVER_15MINS_CALLS_PER_MONTH', 'AVERAGE_CALL_DURATION', 'LEAVE',
       'COLLEGE_one', 'COLLEGE_zero', 'REPORTED_SATISFACTION_avg',
       'REPORTED_SATISFACTION_sat', 'REPORTED_SATISFACTION_unsat',
       'REPORTED_SATISFACTION_very_sat',
       'REPORTED_SATISFACTION_very_unsat', 'REPORTED_USAGE_LEVEL_avg',
       'REPORTED_USAGE_LEVEL_high', 'REPORTED_USAGE_LEVEL_little',
       'REPORTED_USAGE_LEVEL_very_high',
```

```
                'REPORTED_USAGE_LEVEL_very_little',
                'CONSIDERING_CHANGE_OF_PLAN_actively_looking_into_it',
                'CONSIDERING_CHANGE_OF_PLAN_considering',
                'CONSIDERING_CHANGE_OF_PLAN_never_thought',
                'CONSIDERING_CHANGE_OF_PLAN_no',
                'CONSIDERING_CHANGE_OF_PLAN_perhaps'], dtype=object)
```

In [607]:
```
#25 columns
df.columns.shape
```

Out[607]:
```
(25,)
```

In [608]:
```
df_vars=df.columns.values.tolist()
y=['LEAVE']
X=[i for i in df_vars if i not in y]
```

In [609]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
rfe = RFE(model, 15) #pick number of columns you want
rfe = rfe.fit(df[X], df[y])
print(rfe.support_)
print(rfe.ranking_)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/utils/validation.py:761: DataConversionWarning:
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)


[False False False False False  True False  True  True  True  True False
  True  True  True  True  True  True False  True  True  True  True False]
```

```
[ 9  6  4 10  8  1  2  1  1  1  1  7  1  1  1  1  1  1  3  1  1  1  1  5]
```

In [610]:
```
#[5,7,8,9,10,12,13,14,15,16,17,19,20,21,22]
X
```

Out[610]:
```
['INCOME',
 'OVERAGE',
 'LEFTOVER',
 'HOUSE',
 'HANDSET_PRICE',
 'OVER_15MINS_CALLS_PER_MONTH',
 'AVERAGE_CALL_DURATION',
 'COLLEGE_one',
 'COLLEGE_zero',
 'REPORTED_SATISFACTION_avg',
 'REPORTED_SATISFACTION_sat',
 'REPORTED_SATISFACTION_unsat',
 'REPORTED_SATISFACTION_very_sat',
 'REPORTED_SATISFACTION_very_unsat',
 'REPORTED_USAGE_LEVEL_avg',
 'REPORTED_USAGE_LEVEL_high',
 'REPORTED_USAGE_LEVEL_little',
 'REPORTED_USAGE_LEVEL_very_high',
 'REPORTED_USAGE_LEVEL_very_little',
 'CONSIDERING_CHANGE_OF_PLAN_actively_looking_into_it',
 'CONSIDERING_CHANGE_OF_PLAN_considering',
 'CONSIDERING_CHANGE_OF_PLAN_never_thought',
 'CONSIDERING_CHANGE_OF_PLAN_no',
 'CONSIDERING_CHANGE_OF_PLAN_perhaps']
```

The Recursive Feature Elimination (RFE) works by recursively removing variables and building a model on those variables that remain. It uses the model accuracy to identify which variables (and combination of variables) contribute the most to predicting the target attribute.

'OVER_15MINS_CALLS_PER_MONTH', 'COLLEGE_one', 'COLLEGE_zero', 'REPORTED_SATISFACTION_avg', 'REPORTED_SATISFACTION_sat', 'REPORTED_SATISFACTION_very_sat', 'REPORTED_SATISFACTION_very_unsat', 'REPORTED_USAGE_LEVEL_avg', 'REPORTED_USAGE_LEVEL_high', 'REPORTED_USAGE_LEVEL_little', 'REPORTED_USAGE_LEVEL_very_high', 'CONSIDERING_CHANGE_OF_PLAN_actively_looking_into_it', 'CONSIDERING_CHANGE_OF_PLAN_considering', 'CONSIDERING_CHANGE_OF_PLAN_never_thought', 'CONSIDERING_CHANGE_OF_PLAN_no'

In [611]:
```
test = pd.read_csv(filename, header = 0, delimiter = ',')
test.drop(test.columns[[0,8,9,10]], axis=1, inplace=True) #drop categorical data
test.columns.values
```

Out[611]:
```
array(['INCOME', 'OVERAGE', 'LEFTOVER', 'HOUSE', 'HANDSET_PRICE',
       'OVER_15MINS_CALLS_PER_MONTH', 'AVERAGE_CALL_DURATION', 'LEAVE'],
      dtype=object)
```

```
In [612]: test_vars=test.columns.values.tolist()
          yt=['LEAVE']
          Xt=[i for i in test_vars if i not in yt]

          testmodel = LogisticRegression()
          rfe = RFE(testmodel, 4) #pick number of columns you want, this tests which cols migh
          rfe = rfe.fit(test[Xt], test[yt])
          print(rfe.support_)
          print(rfe.ranking_)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/utils/validation.py:761: DataConversionWarning:
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

```
[False  True  True False False  True  True]
[3 1 1 4 2 1 1]
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

```
In [613]: Xt
```

```
Out[613]: ['INCOME',
           'OVERAGE',
           'LEFTOVER',
           'HOUSE',
           'HANDSET_PRICE',
           'OVER_15MINS_CALLS_PER_MONTH',
           'AVERAGE_CALL_DURATION']
```

i agree with these ones! 'OVERAGE', 'LEFTOVER', but it really thinks these are important: 'OVER_15MINS_CALLS_PER_MONTH', 'AVERAGE_CALL_DURATION'

```
In [614]: colstest=['OVERAGE', 'LEFTOVER', 'OVER_15MINS_CALLS_PER_MONTH', 'HOUSE']#, 'AVERAGE_
          Xtest=test[colstest]
          ytest=test['LEAVE']
```

3 LOGISTIC REGRESSION

```
In [615]: from sklearn.model_selection import train_test_split
          from sklearn.linear_model import LogisticRegression
```

```
from sklearn import metrics

Xt_train, Xt_test, yt_train, yt_test = train_test_split(Xtest, ytest, test_size=0.3,
logregtest = LogisticRegression()
logregtest.fit(Xt_train, yt_train)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

Out[615]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [616]:
```
from sklearn.metrics import accuracy_score
from scipy.stats import linregress
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import datasets, linear_model

### with test where we only use columns provided we get Logistic regression accuracy
### 'HOUSE' brings Logistic regression accuracy: .613 --> 0.623 Mean squared error:

print('Logistic regression accuracy: {:.3f}'.format(accuracy_score(yt_test, logregte
yt_pred = logregtest.predict(Xt_test)
lin_mset = mean_squared_error(yt_pred, yt_test)
lin_rmset = np.sqrt(lin_mset)

print('Coefficients: \n', logregtest.coef_)
print("Mean squared error: %.2f"% lin_rmset) #how off the prediction is
print('Variance/R^2 score: %.4f' % r2_score(yt_test, yt_pred)) #closer to 1 = less e
```

```
Logistic regression accuracy: 0.623
Coefficients:
 [[ 6.23308994e-03  4.53441684e-03  9.29884304e-04 -1.46848360e-06]]
Mean squared error: 0.61
Variance/R^2 score: -0.5100
```

4 Random Forest

In [617]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(Xt_train, yt_train)
## with test where theres only columns provided we get Random Forest Accuracy: 0.62
print('Random Forest Accuracy: {:.3f}'.format(accuracy_score(yt_test, rf.predict(Xt_
```

```
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The defau
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
Random Forest Accuracy: 0.616
```

5 Support Vector Machine

```
In [618]: from sklearn.svm import SVC

          svc = SVC()
          svc.fit(Xt_train, yt_train)
          ## with test, Support vector machine accuracy: 0.507
          print('Support vector machine accuracy: {:.3f}'.format(accuracy_score(yt_test, svc.p
```

```
/usr/local/lib/python3.7/site-packages/sklearn/svm/base.py:196: FutureWarning: The default valu
  "avoid this warning.", FutureWarning)
```

```
Support vector machine accuracy: 0.507
```

Cross validation attempts to avoid overfitting while still producing a prediction for each observation dataset. We are using 10-fold Cross-Validation to train our Random Forest model.

```
In [619]: ### Cross validation

          from sklearn import model_selection
          from sklearn.model_selection import cross_val_score
          kfold = model_selection.KFold(n_splits=10, random_state=7)
          modelCV = LogisticRegression()#RandomForestClassifier()
          scoring = 'accuracy'
          results = model_selection.cross_val_score(modelCV, Xt_train, yt_train, cv=kfold, sco
          print("10-fold cross validation average accuracy: %.3f" % (results.mean()))
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

```
10-fold cross validation average accuracy: 0.636
```

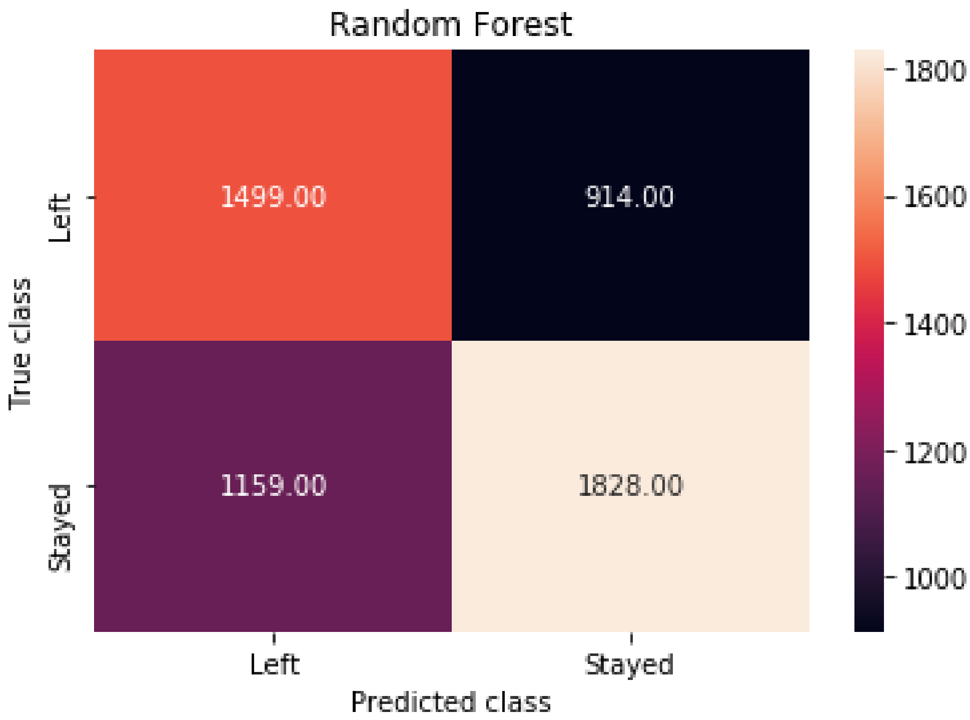
```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

6 Confusion Matrices

In [620]:
```
##RANDOM FOREST

from sklearn.metrics import classification_report
print(classification_report(yt_test, rf.predict(Xt_test)))
```

```
              precision    recall  f1-score   support

           0       0.61      0.67      0.64      2742
           1       0.62      0.56      0.59      2658

   micro avg       0.62      0.62      0.62      5400
   macro avg       0.62      0.62      0.61      5400
weighted avg       0.62      0.62      0.62      5400
```

In [621]:
```
y_pred = rf.predict(Xt_test)
from sklearn.metrics import confusion_matrix
import seaborn as sns
forest_cm = metrics.confusion_matrix(y_pred, yt_test, [1,0])
sns.heatmap(forest_cm, annot=True, fmt='.2f',xticklabels = ["Left", "Stayed"] , ytic
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Random Forest')
plt.savefig('random_forest')
```



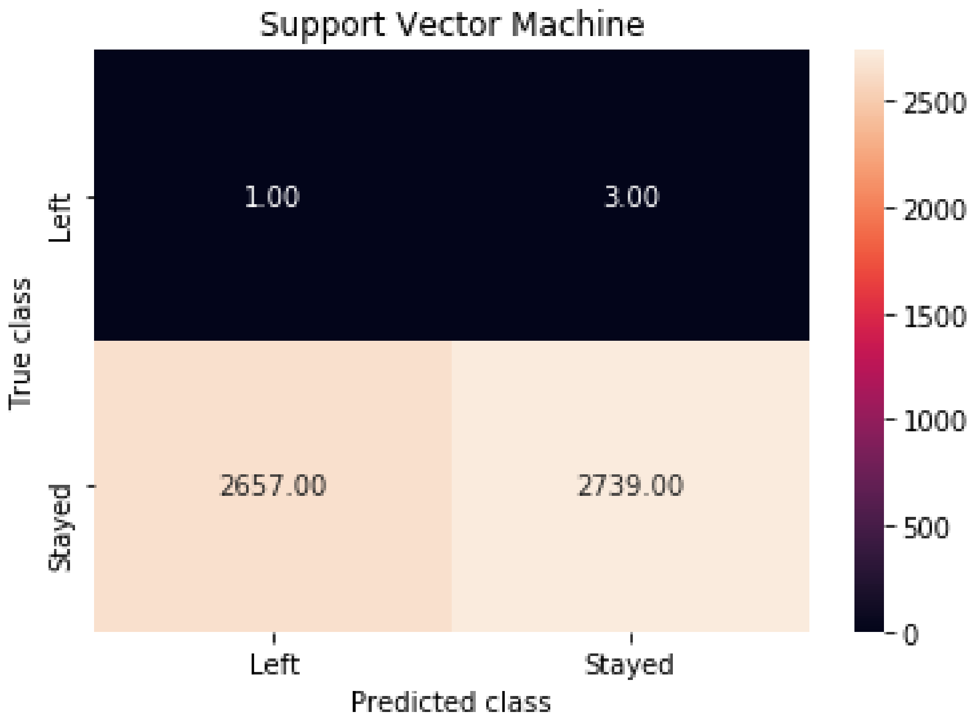
In [622]:
```
## LOGISTIC REGRESSION
print(classification_report(yt_test, logregtest.predict(Xt_test)))
```

```
              precision    recall  f1-score   support

           0       0.61      0.70      0.65      2742
           1       0.64      0.55      0.59      2658

   micro avg       0.62      0.62      0.62      5400
   macro avg       0.62      0.62      0.62      5400
weighted avg       0.62      0.62      0.62      5400
```

In [623]:
```
logreg_y_pred = logregtest.predict(Xt_test)
logreg_cm = metrics.confusion_matrix(logreg_y_pred, yt_test, [1,0])
sns.heatmap(logreg_cm, annot=True, fmt='.2f',xticklabels = ["Left", "Stayed"] , ytic
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.title('Logistic Regression')
plt.savefig('logistic_regression')
```

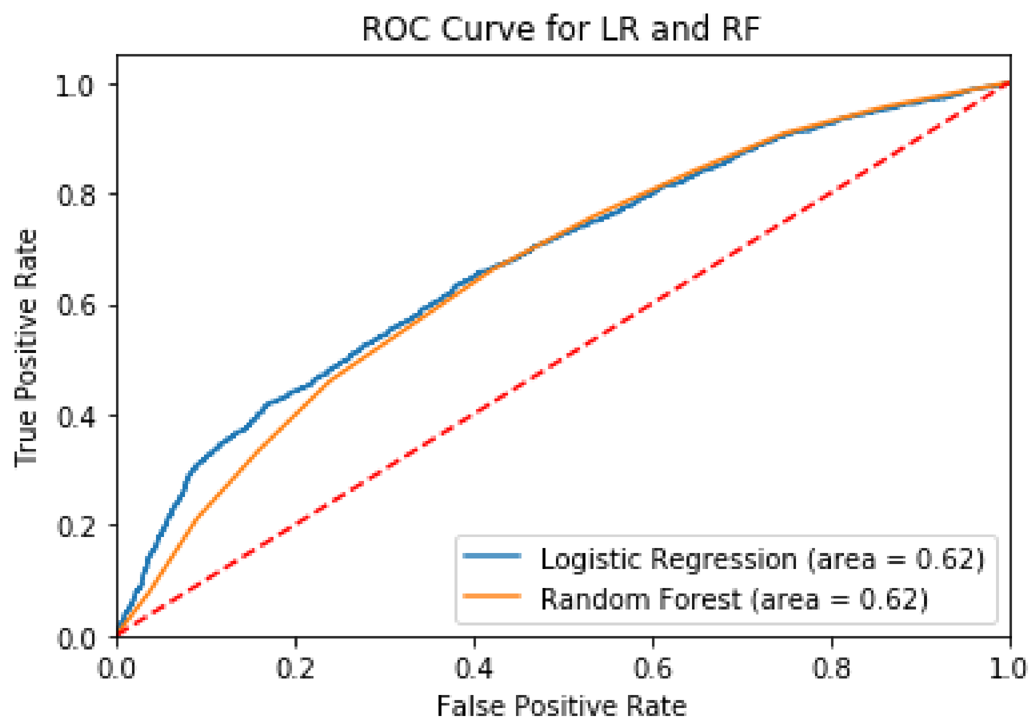

```
In [624]: ## Support Vector Machine
          print(classification_report(yt_test, svc.predict(Xt_test)))

              precision    recall  f1-score   support

           0       0.51      1.00      0.67      2742
           1       0.25      0.00      0.00      2658

   micro avg       0.51      0.51      0.51      5400
   macro avg       0.38      0.50      0.34      5400
weighted avg       0.38      0.51      0.34      5400
```

```
In [625]: svc_y_pred = svc.predict(Xt_test)
          svc_cm = metrics.confusion_matrix(svc_y_pred, yt_test, [1,0])
          sns.heatmap(svc_cm, annot=True, fmt='.2f',xticklabels = ["Left", "Stayed"] , yticklal
          plt.ylabel('True class')
          plt.xlabel('Predicted class')
          plt.title('Support Vector Machine')
          plt.savefig('support_vector_machine')
```



7 ROC Curve

In [628]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(yt_test, logregtest.predict(Xt_test))
fpr, tpr, thresholds = roc_curve(yt_test, logregtest.predict_proba(Xt_test)[:,1])
rf_roc_auc = roc_auc_score(yt_test, rf.predict(Xt_test))
rf_fpr, rf_tpr, rf_thresholds = roc_curve(yt_test, rf.predict_proba(Xt_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(rf_fpr, rf_tpr, label='Random Forest (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for LR and RF')
plt.legend(loc="lower right")
plt.savefig('ROC')
plt.show()
```



In [627]:
```
### Random Forest Model Feature Importance

feature_labels = np.array(['OVERAGE', 'LEFTOVER', 'OVER_15MINS_CALLS_PER_MONTH', 'HOU
importance = rf.feature_importances_
feature_indexes_by_importance = importance.argsort()
for index in feature_indexes_by_importance:
    print('{}-{:.2f}%'.format(feature_labels[index], (importance[index] *100.0)))
```

```
OVER_15MINS_CALLS_PER_MONTH-11.79%
LEFTOVER-19.35%
OVERAGE-20.87%
HOUSE-47.99%
```

8 Checking 14 columns

To mess around with:

cols=['OVER_15MINS_CALLS_PER_MONTH', 'COLLEGE_one', 'COLLEGE_zero', 'REPORTED_SATISFACTION_avg', 'REPORTED_SATISFACTION_sat', 'REPORTED_SATISFACTION_very_sat', 'REPORTED_SATISFACTION_very_unsat', 'REPORTED_USAGE_LEVEL_avg', 'REPORTED_USAGE_LEVEL_high', 'REPORTED_USAGE_LEVEL_little', 'REPORTED_USAGE_LEVEL_very_high', 'CONSIDERING_CHANGE_OF_PLAN_actively_looking_into_it', 'CONSIDERING_CHANGE_OF_PLAN_considering', 'CONSIDERING_CHANGE_OF_PLAN_never_thought', 'CONSIDERING_CHANGE_OF_PLAN_no']

It appears none of these extras really do much difference...

9 LR

In [570]:
```
#currently 14 columns
cols=['OVERAGE', 'LEFTOVER',
'OVER_15MINS_CALLS_PER_MONTH',
'COLLEGE_zero',
'REPORTED_SATISFACTION_avg',
'REPORTED_SATISFACTION_sat',
'REPORTED_SATISFACTION_very_sat',
'REPORTED_SATISFACTION_very_unsat',
'REPORTED_USAGE_LEVEL_little',
'REPORTED_USAGE_LEVEL_very_high',
'CONSIDERING_CHANGE_OF_PLAN_actively_looking_into_it',
'CONSIDERING_CHANGE_OF_PLAN_considering',
'CONSIDERING_CHANGE_OF_PLAN_never_thought',
'CONSIDERING_CHANGE_OF_PLAN_no']
X=df[cols]
y=df['LEAVE']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

#the test set is like new data, where outputs are withheld
#These are predictions for the values that are withheld based on the xTest set, how
y_pred = logreg.predict(X_test)
lin_mse = mean_squared_error(y_pred, y_test)
lin_rmse = np.sqrt(lin_mse)

print('Logistic regression accuracy: {:.3f}'.format(accuracy_score(y_test, logreg.pre
print('Coefficients: \n', logreg.coef_)
print("Mean squared error: %.2f"% lin_rmse) #how off the prediction is
print('Variance/R^2 score: %.4f' % r2_score(y_test, y_pred)) #closer to 1 = less err
```

```
Logistic regression accuracy: 0.613
Coefficients:
 [[ 0.00505076  0.00494868  0.01128977 -0.07206051 -0.05264074 -0.17197155
   0.00233897 -0.00425665 -0.02238741  0.03802889 -0.09431067 -0.13381085
  -0.17941043 -0.10639375]]
Mean squared error: 0.62
Variance/R^2 score: -0.5500
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

10 RF

```
In [571]: rf = RandomForestClassifier()
          rf.fit(X_train, y_train)
          ## with test where theres only columns provided we get Random Forest Accuracy: 0.62
          print('Random Forest Accuracy: {:.3f}'.format(accuracy_score(y_test, rf.predict(X_te
```

```
Random Forest Accuracy: 0.560
```

```
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The defa
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

11 SVM

```
In [572]: svc = SVC()
          svc.fit(X_train, y_train)
          ## with test, Support vector machine accuracy: 0.507
          print('Support vector machine accuracy: {:.3f}'.format(accuracy_score(y_test, svc.pre
```

```
/usr/local/lib/python3.7/site-packages/sklearn/svm/base.py:196: FutureWarning: The default val
  "avoid this warning.", FutureWarning)
```

```
Support vector machine accuracy: 0.597
```

- LR for 4 columns is most successful model out of 4/14 columns

12 Predicting testing data

```
In [573]: filename2 = 'test.csv'
          testdf = pd.read_csv(filename2, header = 0, delimiter = ',')
          testdf.columns.values
```

```
Out[573]: array(['COLLEGE', 'INCOME', 'OVERAGE', 'LEFTOVER', 'HOUSE',
                 'HANDSET_PRICE', 'OVER_15MINS_CALLS_PER_MONTH',
                 'AVERAGE_CALL_DURATION', 'REPORTED_SATISFACTION',
                 'REPORTED_USAGE_LEVEL', 'CONSIDERING_CHANGE_OF_PLAN'], dtype=object)
```

```
In [574]: testdf.shape
```

```
Out[574]: (2000, 11)
```

```
In [575]: #print(testdf)
```

```
In [576]: #testcol=['INCOME', 'OVERAGE', 'LEFTOVER', 'HOUSE', 'HANDSET_PRICE',
          #        'OVER_15MINS_CALLS_PER_MONTH', 'AVERAGE_CALL_DURATION']
          testcol=['OVERAGE', 'LEFTOVER', 'OVER_15MINS_CALLS_PER_MONTH', 'HOUSE']
          predX=testdf[testcol]

          newdf = pd.DataFrame(columns=['ID', 'LEAVE'])

          for x in newdf:
              newdf['ID'] = newdf.index
              newdf['LEAVE'] = logregtest.predict(predX)    #logregtest expects 3 samples, logr
```

```
In [577]: #print(newdf)
```

```
In [578]: newdf.head()
```

```
Out[578]:    ID  LEAVE
          0   0      0
          1   1      1
          2   2      0
          3   3      0
          4   4      1
```

```
In [579]: #newdf.to_csv('LRoutputWITHHOUSE.csv')
```

Notes:

- I want to know the accuracy when all columns are included
- To see accuracy of LR with testing where the extra categoricals are included, must mess with the shape of the output DF