Project Conclusion

At this part of the project, we will finalize what we did until now and what we achieved in this blog. At the beginning of the project, we spent our time on understanding the data set well and what can we achieve through these information sets. After that, the blog posts about exploring the data, visualizing some histograms and hypothesis testing came. Then, we spent some time on regression and logistic regression gave really good results about the likelihood of "matching" with the partner during the speed dating night. Choosing the appropriate target for the regression; a binary dependent variable made the regression model more successful than the other regression models.

The second part of our project development was primarily about Machine Learning techniques that helped us about classifications, regressions and ascertaining the patterns found in the data. In all techniques, our aim was to reveal the effect of several attributes; attractiveness, sincerity, intelligence, fun, ambitiousness, shared interests and the score that corresponds how much the attendee liked their partner on the decision given by the attendee at the end of the night. In general, we used **supervised learning algorithms** as our ML techniques. Supervised learning requires input data that has both predictor(independent) variables and a target(dependent) variable whose value is to be estimated.

As the first algorithm, we decided on implementing **decision tree** because it is simple to interpret, visualize and understand when compared the other techniques. They have several advantages like using for both classification and regression and also handling both numerical and categorical data. We used two different models for decision tree implementation. Or naive_model and param_model gave **0.70 and 0.80 accuracy scores** respectively which are the highest scores when compared the other algorithms that will be explained further in this post. One problem about our decision tree models was that we got over-complex tree results that we think that do not generalize the data well and complicated to reach to a decision from the tree. Another finding from the model was that our decision trees were changing quickly with small size changes in our data set. For the small numbers of rows, we got smaller accuracy scores but also smaller decision trees.

To overcome the overfitting problem we explained above for the decision trees, we also used **random forest**. As decision trees, random forest is also for both classification and regression. Difference between random forest and decision tree is that the process of finding the root node and splitting the feature nodes will run randomly. As the result of this model, we got **0.57 accuracy score**.

The last ML technique we used was **neural networks**. Specifically, we used **multilayer perceptron** as the neural network type. When we compare the neural networks to the decision trees and random forest, we found that they are very effective at modelling highly complex non-linear relationships because they can have many layers with non-linearities. Multilayer perceptron modeled more arbitrary functions and therefore was more accurate. As expected, this conclusion is affected by our the data set. Although we shrink our original data set, it was still large to obtain a simple decision tree but we got the best accuracy scores on that data size. We also got **0.68 accuracy score** for our neural network model. One problem about the neural networks was they are not easy to understand what these networks do inside to analyze the patterns happen in the data.

At the end, we got the highest accuracy score from decision tree implementation but we as a team think that, because our data is not a simple and small data and also it contains non-linearities, neural network's multilayer perceptron model gave more accurate results that explain the relation between our variables. One should remember that, the best fitted model is the one that most accurately fits your data.

May 19, 2018 Sevde Bozdogan Leave a comment

ML2 – Artificial Neural Networks & Multilayer Perceptron

In this part of the project, we will use a specific type of Artificial Neural Networks which is Multilayer Perceptron to ascertain the patterns and trends happening within our data.

We will use this technique the ascertain the affect of features perceived from matches during speed date to the decision of the attendee at the end of the night. “dec” variable can get 2 different values; “0” indicates the negative decision and “1” indicates the positive decision.

In the Multilayer Perceptron model, the input layer of Perceptron will receive the signal and its output layer will make predictions about our input (“dec” variable).

Import the essential libraries and read our data set file:

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [2]: from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import Perceptron
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
```

```
In [3]: import xlrd
        xlsfile = pd.ExcelFile('SPEED.xls')

        DF = xlsfile.parse('Sheet1')
```

Include only the needed variables for the analysis:

```
In [4]: input_vars = ['gender', 'samerace', 'attr', 'sinc', 'intel', 'fun', 'amb',
                      'shar', 'like', 'dec', ]
```

```
In [5]: df = DF.loc[:, input_vars]
```

```
In [6]: df.isnull().sum()
        df.dropna(inplace=True)
```

Features and target to ascertain the pattern between the attributes of the match and the decision at the end of the night:

```
In [7]: features = df.drop(['dec'], axis=1)
        target = df[['dec']]
```

Use train_test_split:

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3)
```

Train the scaler:

```
In [11]: #Train the scaler, which standardizes all the features to have mean=0 and unit variance
         sc = StandardScaler()
         sc.fit(X_train)

Out[11]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

Apply the scaler to the train and test data:

```
In [12]: #Apply the scaler to the X training data
         X_train_std = sc.transform(X_train)

         #Apply the SAME scaler to the X test data
         X_test_std = sc.transform(X_test)
```

Perceptron object:

```
In [13]: #Create a perceptron object with the parameters: 40 iterations (epochs) over the data, and a learning rate of 0.1
         ppn = Perceptron(max_iter = 40, eta0 = 0.1, random_state = 0)

         #Train the perceptron
         ppn.fit(X_train_std, y_train)

         C:\Users\SUUSER\Anaconda3\lib\site-packages\sklearn\utils\validation.py:578: DataConversionWarning: A column-vector
         as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
           y = column_or_1d(y, warn=True)

Out[13]: Perceptron(alpha=0.0001, class_weight=None, eta0=0.1, fit_intercept=True,
               max_iter=40, n_iter=None, n_jobs=1, penalty=None, random_state=0,
               shuffle=True, tol=None, verbose=0, warm_start=False)
```

Apply the trained perceptron on the X data(features) to make predicts on Y data(target dec):

```
In [14]: #Apply the trained perceptron on the X data to make predicts for the y test data
         y_pred = ppn.predict(X_test_std)

In [15]: print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))

         Accuracy: 0.68
```

We got a **68% accuracy score** from the perceptron.

Import the essential libraries for the Multilayer Perceptron model:

```
In [16]: from sklearn.neural_network import MLPClassifier
         from sklearn.preprocessing import MinMaxScaler
         import matplotlib.pyplot as plt

         %matplotlib inline
```

Fit the model to the data using MLPClassifier:

```
In [18]: features = MinMaxScaler().fit_transform(features)

In [19]: mlp = MLPClassifier(verbose=0, random_state=0, max_iter=40, nesterovs_momentum=False,
                             solver='sgd', learning_rate='invscaling', momentum=0.9, learning_rate_init=0.2)

In [20]: mlp.fit(features,target)

C:\Users\SUUSER\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:912: Da
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samp
sing ravel().
  y = column_or_1d(y, warn=True)
C:\Users\SUUSER\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:564: C
ochastic Optimizer: Maximum iterations (40) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Out[20]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(100,), learning_rate='invscaling',
                learning_rate_init=0.2, max_iter=40, momentum=0.9,
                nesterovs_momentum=False, power_t=0.5, random_state=0, shuffle=True,
                solver='sgd', tol=0.0001, validation_fraction=0.1, verbose=0,
                warm_start=False)
```

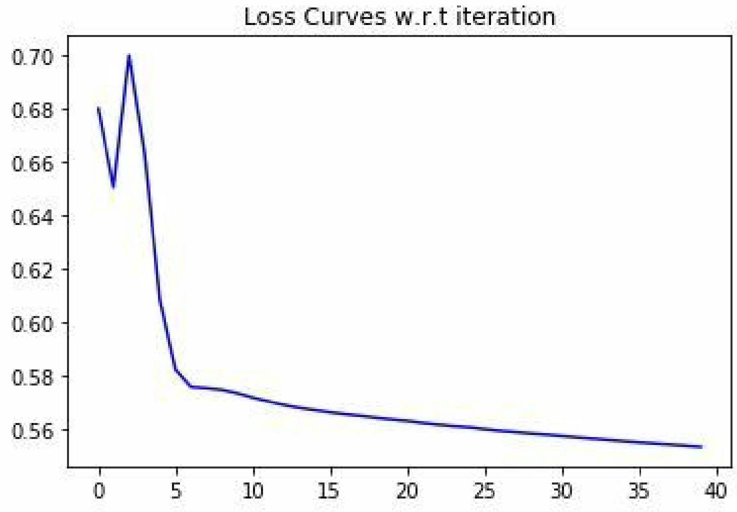
Get the training set score and training set loss:

```
In [22]: print("Training set score: %f" % mlp.score(features, target))
         print("Training set loss: %f" % mlp.loss_)

Training set score: 0.747755
Training set loss: 0.553258
```

Plot the loss function with respect to iteration:

In [23]:
```
#plot the loss function
plt.title("Loss Curves w.r.t iteration")
plt.plot(mlp.loss_curve_, c='blue', linestyle='-')
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0xc7db630>]
```



Our GitHub project repository link: cs210-project

May 17, 2018 Sevde Bozdogan Leave a comment

ML1 – Decision Trees &

Random Forest

In this part of the project, we will create and visualize a decision tree from a newly created small set of our data as a ML technique for our project. Our goal here is to create a model that predicts the value of our target variable which is "dec" that indicates the decision of attendees from speed dating about seeing their matches again or not. We will use the scores that the attendee distributes for the attributes of their matches.

Our small set of data can be found here: SPEED

```
In [168]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt

          %matplotlib inline

In [169]: import xlrd
          xlsfile =pd.ExcelFile('SPEED.xls')

          DF =xlsfile.parse('Sheet1')

In [170]: DF.describe()

Out[170]:
```

	iid	id	gender	idg	condtn	wave	round	position
count	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000
mean	53.744856	7.844993	0.432099	15.045267	1.725652	2.617284	15.596708	8.611111
std	28.279819	4.922946	0.495538	9.583610	0.446338	1.062380	3.582431	5.177196
min	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	10.000000	1.000000
25%	31.000000	4.000000	0.000000	7.000000	1.000000	2.000000	10.000000	4.000000
50%	51.000000	7.000000	0.000000	14.000000	2.000000	2.000000	18.000000	8.000000
75%	80.000000	11.000000	1.000000	22.000000	2.000000	4.000000	18.000000	13.000000
max	100.000000	20.000000	1.000000	35.000000	2.000000	4.000000	19.000000	19.000000

8 rows × 192 columns

We only included the first 1458 rows to our small sample data set.

Only take the columns that corresponds to the gender of the attendee (gender), having same race or not (samerace), attractiveness score of partner (attr), sincerity score of partner (sinc), intelligence score of partner(intel), fun score of partner(fun), ambitiousness score of partner (amb), shared interests' score of partner (shar), whether the attendee liked the partner or not (like) and lastly, the decision after the night (dec).

```
In [171]: input_vars = ['gender', 'samerace', 'attr', 'sinc', 'intel', 'fun', 'amb',
                        'shar', 'like', 'dec', ]
          #gender value 0 for women, 1 for men
          #samerace has value 1 for having same race, 0 for different races
          #attr, sinc, intel, fun, amb, shar has values of partners' attributes after speed date, 1=awful 10=great
          #like corresponds to how much the participant liked their matches
          #dec_o corresponds to decision after date with value 0 for no, 1 for yes

In [172]: df = DF.loc[:, input_vars]

In [173]: df.isnull().sum()
          df.dropna(inplace=True)

In [174]: df.describe()

Out[174]:
```

	gender	samerace	attr	sinc	intel	fun	amb	shar	like	de
count	1225.000000	1225.000000	1225.000000	1225.000000	1225.000000	1225.000000	1225.000000	1225.000000	1225.000000	1225.0000
mean	0.418776	0.413061	5.962449	7.074694	7.345714	6.290204	6.788163	5.425306	6.054694	0.3926
std	0.493560	0.492585	1.956277	1.702415	1.556097	2.045456	1.815258	2.123133	1.931705	0.4885
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000	5.000000	6.000000	6.000000	5.000000	6.000000	4.000000	5.000000	0.0000
50%	0.000000	0.000000	6.000000	7.000000	7.000000	6.000000	7.000000	5.000000	6.000000	0.0000
75%	1.000000	1.000000	7.000000	8.000000	8.000000	8.000000	8.000000	7.000000	7.000000	1.0000

Decision tree to predict the behaviour of attendees to say "yes" to their partners during speed dating, use "dec" column as target:

```
In [175]: from sklearn import tree

In [176]: from sklearn.tree import DecisionTreeClassifier, export_graphviz

          from sklearn.metrics import accuracy_score, confusion_matrix
          from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import LabelEncoder

In [177]: features = df.drop(['dec'], axis=1)
          target = df[['dec']]
```

We have two different models for decision trees; naive_model and

param_model:

```
In [178]: naive_model = DecisionTreeClassifier(random_state=42)
          param_model = DecisionTreeClassifier(max_depth=5, min_samples_split=10, min_samples_leaf=10, random_state=42)
```

```
In [179]: def train_and_predict(model, features, target):
              X_train, X_test, y_train, y_test = train_test_split(features,target, test_size=0.33, random_state=42)
              model.fit(X_train, y_train)
              y_pred = model.predict(X_test)
              print("accuracy score: %.2f" % accuracy_score(y_test, y_pred))
              (tn, fp, fn, tp) = confusion_matrix(y_test, y_pred).ravel()
              print("confusion matrix")
              print("tn, fp, fn, tp")
              print(tn, fp, fn, tp)
```

Accuracy scores & confusion matrices:

```
In [180]: train_and_predict(naive_model, features, target)

          accuracy score: 0.70
          confusion matrix
          tn, fp, fn, tp
          193 50 71 91
```

```
In [181]: train_and_predict(param_model, features, target)

          accuracy score: 0.80
          confusion matrix
          tn, fp, fn, tp
          208 35 48 114
```

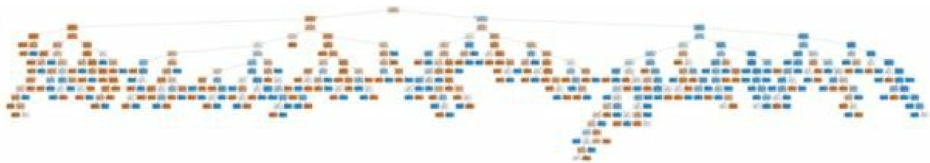
naive_model:

```
In [21]: dot_data = StringIO()
         export_graphviz(naive_model, out_file=dot_data, filled=True, rounded=True, special_characters=True)
```

```
In [22]: import pydotplus
```

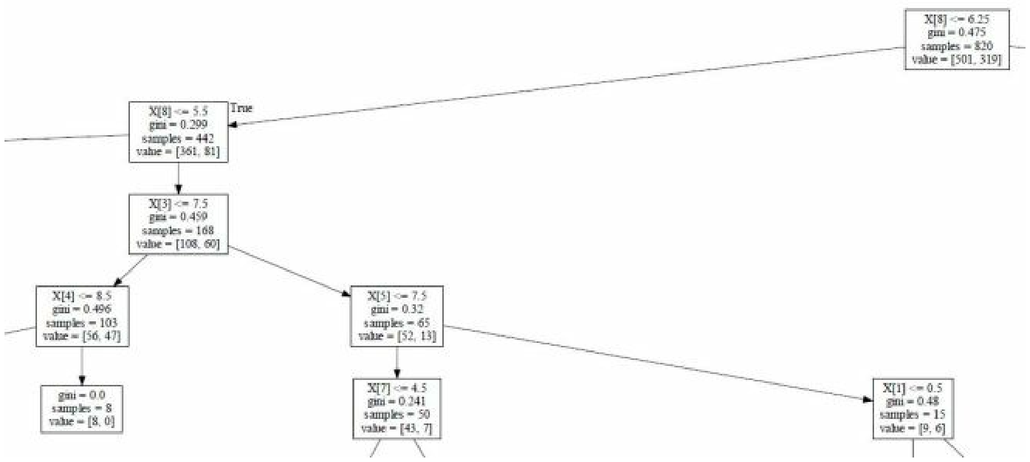
```
In [23]: graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         Image(graph.create_png())
```

Out[102]:



```
In [66]: with open("naive_model.dot", "w") as f:
             f = tree.export_graphviz(naive_model, out_file=f)
```

pdf of our naive_model decision tree: naive_model.pdf



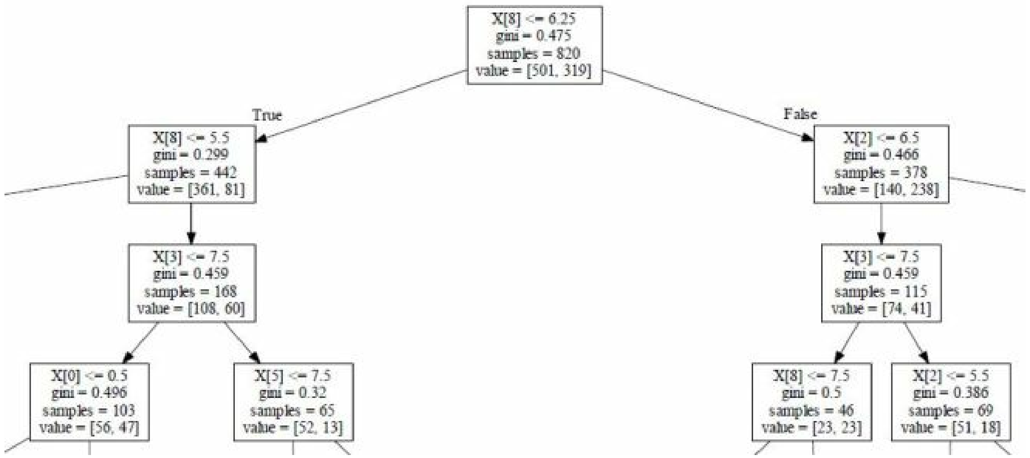
param_model:

```
In [24]: dot_data = StringIO()
         export_graphviz(param_model, out_file=dot_data, filled=True, rounded=True, special_characters=True)

In [25]: import pydotplus

In [26]: graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         Image(graph.create_png())
```

Out[121]:

```
In [122]: with open("param_model.dot", "w") as f:
              f = tree.export_graphviz(param_model, out_file=f)
```

pdf of our param_model decision tree: param_model.pdf



Some control on trained score:

In [144]:
```
xtr=features[:100]
xte=features[100:]
ytr=target[:100]
yte=target[100:]
```

In [145]:
```
model2=DecisionTreeClassifier()
model2.fit(xtr,ytr)
Prediction=model2.predict(xte)
```

In [146]:
```
Prediction
```

Out[146]:
```
array([0, 0, 0, ..., 1, 0, 0], dtype=int64)
```

Confusion matrix and classification report:

In [148]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [150]:
```
print("Trained Score")
print(confusion_matrix(yte,Prediction))
print(classification_report(yte,Prediction))
```

```
Trained Score
[[534 163]
 [266 162]]
             precision    recall  f1-score   support

          0       0.67      0.77      0.71       697
          1       0.50      0.38      0.43       428

avg / total       0.60      0.62      0.61      1125
```

Now, try it with **Random forest:**

In [152]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [153]:
```
tree=RandomForestClassifier()
model3=tree.fit(xtr,ytr)
```

```
C:\Users\SUUSER\Anaconda3\lib\site-packages\ipykernel
d when a 1d array was expected. Please change the sha
```

In [154]:
```
Prediction2=model3.predict(xte)
```

In [155]:
```
Prediction2
```

Out[155]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Results:

```
In [33]: print("Real Values", str(yte.values))
         print("Estimated Values", str(Prediction2))

Real Values [[0]
 [0]
 [0]
 ...,
 [1]
 [0]
 [1]]
Estimated Values [0 0 0 ..., 0 0 0]

In [34]: print("Error:" , str(np.mean(yte.values!=Prediction2)))

Error: 0.438468740741

In [35]: print(accuracy_score(yte, Prediction2))

0.593777777778
```

According the random forest, we got an accuracy score **of 59%** between the real values and estimation.

Our GitHub project repository link: cs210-project

May 13, 2018 Sevde Bozdogan Leave a comment

Logistic Regression

In this part of the project, we used logistic regression to understand our data better, and to know the likelihood of some events during speed date. In the experiments that create our data set results, attendees were asked to answer to the questions on a scorecard:

Scorecard:
[Filled out by subjects after each "date" during the event.]

SCORECARD
YOUR ID NUMBER:
Circle "Yes" or "No" below the ID number of each person you meet to indicate whether or not you would like to see him or her again. Rate their attributes on a scale of 1-10; (1=awful, 10=great). If you haven't formed an opinion based on your conversation, fill in N/A, but please fill in all boxes. This will be TOTALLY confidential and will NOT be shared with anyone. Then, answer the remaining questions for each person you meet.

ID #:	1	2	3	4	5	6	7	8	9	10

dec

Decision	1=yes 0=no	Y n	yes no	yes no	yes no	yes no	yes no	yes no	yes no	yes no

Attributes (1=awful, 10=great)										
Attractive	attr									
Sincere	sinc									
Intelligent	intel									
Fun	fun									
Ambitious	amb									
Shared Interests/Hobbies	shar									

(From the data key doc, the link to the document is: https://www.kaggle.com/annavictoria/speed-dating-experiment/data)

In the logistic regression model, we want to predict the likelihood of a "match" -want to see him/her again- based on their ratings for their matches during the first speed date.

Logistic regression gave better results than linear regression because we want to predict if there will be a match or not that means we want to ascertain if there will be the value of “1” or “0” (“yes” or “no”). Dependent variable -in this case, it is **“match”- is binary in logistic regression models.**

In the data set, **a “match” occurs when the both parties -attendee and his/her partner- say “yes” to each other**. “1” indicates that there is a match between decisions and “0” indicates that at least one of the parties said no. The keys in the form “attr_o” corresponds to the all six attribute **rating by partner** during the night. As you can see above, “attr”, “sinc”, “intel”, “fun”, “amb”, “shar” keys corresponds to the **ratings by attendee** from 1 to 10 to score the attributes of the partners in the scorecard. We will use these keys in our regression model:

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sbn
        import statsmodels.api as sma

C:\Users\SUUSER\Anaconda3\lib\site-packages\statsmodels\compat\pandas.
odule is deprecated and will be removed in a future version. Please us
  from pandas.core import datetools

In [2]: import xlrd
        xlsfile = pd.ExcelFile('SPEED.xls')

        DF = xlsfile.parse('Sheet1')
```

Subset the data to the women. Each parameter is a 2-way interaction:

```
In [22]: wk=DF.loc[DF['gender']==0, ['iid', 'gender', 'match', 'attr', 'attr_o', 'sinc', 'sinc_o', 'intel', 'intel_o',
                                    'fun', 'fun_o', 'amb', 'amb_o', 'shar', 'shar_o']]
```

```
In [7]: wk = wk.dropna(axis=0)
```

Take the averages of 2-way attributes and create them as new variables:

```
In [52]: wk['attr_2avg']=abs((wk['attr']+wk['attr_o'])/2)
         wk['sinc_2avg']=abs((wk['sinc']+wk['sinc_o'])/2)
         wk['intel_2avg']=abs((wk['intel']+wk['intel_o'])/2)
         wk['fun_2avg']=abs((wk['fun']+wk['fun_o'])/2)
         wk['amb_2avg']=abs((wk['amb']+wk['amb_o'])/2)
         wk['shar_2avg']=abs((wk['shar']+wk['shar_o'])/2)
```

```
In [53]: wk = wk.drop(labels=['attr', 'attr_o', 'sinc', 'sinc_o', 'intel', 'intel_o', 'fun', 'fun_o','amb', 'amb_o', 'shar',
                      axis=1)
         wk['intercept']=1.0
```

Dataframe:

```
In [54]: print('Speed Dating dataframe shape')
         print(wk.shape)
         print('Speed Dating dataframe column names')
         print(wk.columns)
         print('Speed Dating dataframe summary')
         print(wk.describe())
```

```
Speed Dating dataframe shape
(3013, 10)
Speed Dating dataframe column names
Index(['iid', 'gender', 'match', 'attr_2avg', 'sinc_2avg', 'intel_2avg',
       'fun_2avg', 'amb_2avg', 'shar_2avg', 'intercept'],
      dtype='object')
Speed Dating dataframe summary
               iid  gender        match    attr_2avg    sinc_2avg  \
count  3013.000000  3013.0  3013.000000  3013.000000  3013.000000
mean    277.457351     0.0     0.177896     6.216628     7.183704
std     159.707690     0.0     0.382488     1.350864     1.281388
min       1.000000     0.0     0.000000     1.500000     1.000000
25%     148.000000     0.0     0.000000     5.500000     6.500000
50%     267.000000     0.0     0.000000     6.500000     7.500000
75%     417.000000     0.0     0.000000     7.000000     8.000000
max     530.000000     0.0     1.000000    10.000000    10.000000

        intel_2avg     fun_2avg     amb_2avg    shar_2avg  intercept
count  3013.000000  3013.000000  3013.000000  3013.000000     3013.0
mean      7.391304     6.432625     6.782609     5.505393        1.0
std       1.132624     1.491782     1.283691     1.685084        0.0
min       2.500000     0.500000     1.500000     0.500000        1.0
25%       6.500000     5.500000     6.000000     4.500000        1.0
50%       7.500000     6.500000     7.000000     5.500000        1.0
75%       8.000000     7.500000     7.500000     6.500000        1.0
max      10.000000    10.500000    10.000000    10.000000        1.0
```

According to data frame summary, we have 3013 observations. We have a mean of “match” as 0.17. This means that **a match only occurs between 17.7% of participant pairs.** “att_2avg”, “sinc_2avg”, “intel_2avg”, “fun_2avg”, “amb_2avg” and “shar_2avg” keys correspond to the **average of each participants and partners’ ratings of one another**.

From the data key document, we know that, in the rating from 1 to 10, “1” corresponds to the awful and “10” corresponds to the great.

Logistic Regression & Summary Statistics:

In [55]:
```
indepv = wk.columns[3:] #only the attributes
logreg = sma.Logit(wk['match'],wk[indepv])
logfit = logreg.fit()

print(logfit.summary2())

sbn.set_style('whitegrid')
```

```
Optimization terminated successfully.
         Current function value: 0.357885
         Iterations 7
                          Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.236
Dependent Variable: match            AIC:              2170.6165
Date:               2018-05-18 12:58 BIC:              2212.6914
No. Observations:   3013             Log-Likelihood:   -1078.3
Df Model:           6                LL-Null:          -1410.6
Df Residuals:       3006             LLR p-value:      2.5748e-140
Converged:          1.0000           Scale:            1.0000
No. Iterations:     7.0000
------------------------------------------------------------------
               Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
------------------------------------------------------------------
attr_2avg      0.5776    0.0591    9.7778  0.0000   0.4618   0.6933
sinc_2avg     -0.0624    0.0679   -0.9191  0.3581  -0.1955   0.0707
intel_2avg     0.1385    0.0834    1.6607  0.0968  -0.0249   0.3019
fun_2avg       0.3753    0.0622    6.0312  0.0000   0.2533   0.4973
amb_2avg      -0.2500    0.0651   -3.8386  0.0001  -0.3777  -0.1224
shar_2avg      0.3606    0.0477    7.5610  0.0000   0.2671   0.4541
intercept     -9.0024    0.5033  -17.8881  0.0000  -9.9888  -8.0161
=================================================================
```

From the logistic regression result, we can easily see that **"attractiveness" has the greatest impact for the decision of matching**, because it has the coefficient 0.57 which is the largest. Attributes "sincerity" and "intelligence" do not have a huge impact on matching decision as we can understand from their little coefficients.

In [56]:
```
#groupby 'match' column
wk.groupby('match').mean()
```

Out[56]:

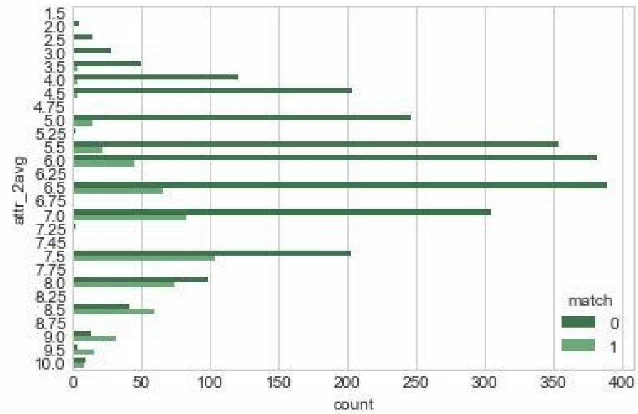
match	iid	gender	attr_2avg	sinc_2avg	intel_2avg	fun_2avg	amb_2avg	shar_2avg	intercept
0	279.132822	0.0	5.975959	7.047033	7.265038	6.178341	6.658862	5.222447	1.0
1	269.714552	0.0	7.328825	7.815299	7.974813	7.607743	7.354478	6.812966	1.0

Looking at this brief glance of data, as expected, it seems that a **match occurs when the average ratings of attributes are higher**, especially for attractiveness, shared interests and fun attributes.

Some histograms:

For “attractiveness” that has the greatest impact:

```
In [58]: #Factorplot for attractiveness score average with match hue
         sbn.countplot(y='attr_2avg', hue='match', data=wk, palette='Greens_d')

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0xcde60b8>
```

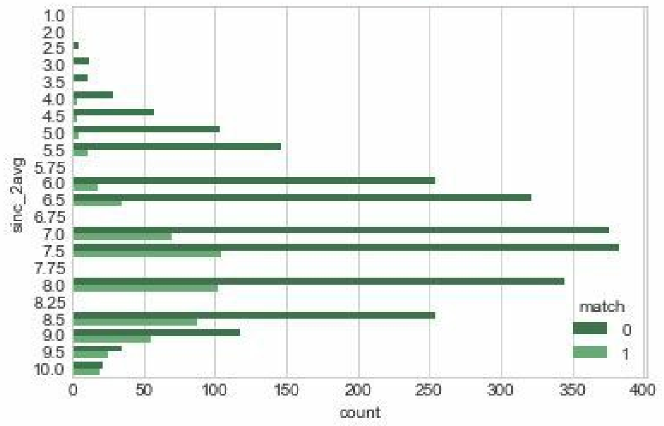


Looks like the probability of matching increase between the attractiveness scores from 1 to 7.5

For “sincerity” that has the smallest impact:

```
In [59]: #Factorplot for sincerity score average with match hue
         sbn.countplot(y='sinc_2avg', hue='match', data=wk, palette='Greens_d')
```

```
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x128cbeb8>
```



Set up the X and Y variables for logistic regression model:

```
In [62]: X = wk.drop(['match', 'iid', 'gender'], axis=1)
         Y = wk.match
```

```
In [63]: X.head()
```

Out[63]:

	attr_2avg	sinc_2avg	intel_2avg	fun_2avg	amb_2avg	shar_2avg	intercept
0	6.0	8.5	7.5	7.5	7.0	5.5	1.0
1	7.0	8.0	8.5	7.5	6.0	5.5	1.0
2	7.5	9.0	9.5	9.0	7.5	8.5	1.0
3	7.0	7.0	8.5	7.5	7.5	8.0	1.0
4	6.5	6.5	8.0	6.5	7.5	6.5	1.0

```
In [64]: Y.head()
```

```
Out[64]: 0    0
         1    0
         2    1
         3    1
         4    1
         Name: match, dtype: int64
```

Flatten the array of Y:

```
In [65]: #to use the Y with scikit learn, set it as 1-D array, flatten the array
         Y = np.ravel(Y)

         #check results
         Y

Out[65]: array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

Now, create the **Logistic Regression Model:**

```
In [66]: log_model = LogisticRegression()

         #Fit our data
         log_model.fit(X,Y)

         #Check our accuracy
         log_model.score(X,Y)

Out[66]: 0.84732824427480913
```

We got a **0.847 (or 84.7%) accuracy rating**.

Compare this to original Y data:

```
In [68]: #Check the percentage of a match
         Y.mean()

Out[68]: 0.17789578493196151
```

This means that if our model just simply guessed "no match" we would have had 1-0.17 = 0.83 accuracy (or 83%). So while we are doing better than the null error rate, we are not doing that much better:

```
In [69]: #Split the data
         X_train, X_test, Y_train, Y_test = train_test_split(X,Y)

         #Make a new log model
         log_model2 = LogisticRegression()

         #Now fit the new model
         log_model2.fit(X_train, Y_train)

Out[69]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [70]: #Predict the classes of the testing data set
         class_predict = log_model2.predict(X_test)

         #Compare the predicted classes to the actual test classes
         print (metrics.accuracy_score(Y_test,class_predict))

         0.855437665782
```

At the end, we got a **accuracy score of 85.5%** between the predicted and actual test classes.

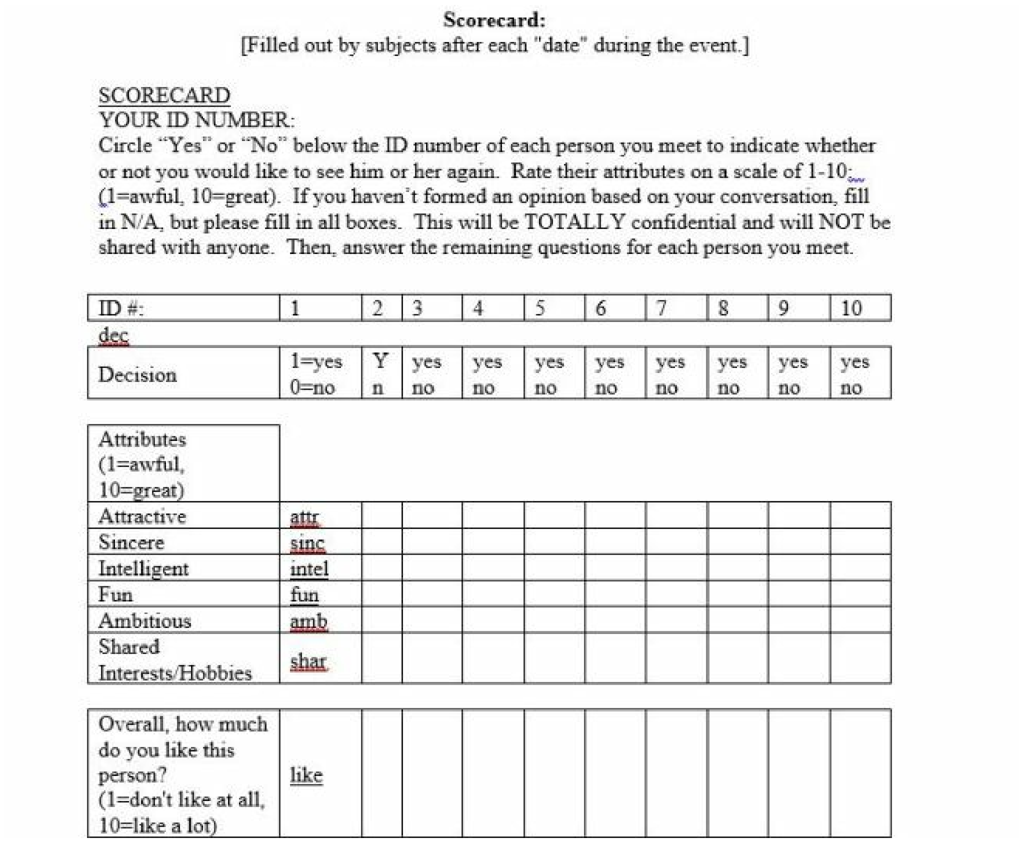
Our GitHub project repository link: cs210-project

April 29, 2018 Sevde Bozdogan Leave a comment

A Simple Linear Regression

In this part of the project, we will carry out a simple linear regression to create a model to determine what factors men and women in the experiment can be used to predict if a “yes” decision is made after the speed date.

This is the scorecard that is given to the speed date attendee to fill out. “dec” corresponds to the decision of the attendee to meet again with his/her partner. Attributes are filled with points by attendee from 1 to 10 that correspond to the perceived attributes of partner at that night. “like” point indicates how much the attendee liked their partner.



Scorecard:

[Filled out by subjects after each "date" during the event.]

SCORECARD

YOUR ID NUMBER:

Circle “Yes” or “No” below the ID number of each person you meet to indicate whether or not you would like to see him or her again. Rate their attributes on a scale of 1-10: (1=awful, 10=great). If you haven’t formed an opinion based on your conversation, fill in N/A, but please fill in all boxes. This will be TOTALLY confidential and will NOT be shared with anyone. Then, answer the remaining questions for each person you meet.

ID #:	1	2	3	4	5	6	7	8	9	10

dec

Decision	1=yes 0=no	Y n	yes no	yes no	yes no	yes no	yes no	yes no	yes no	yes no

Attributes (1=awful, 10=great)										
Attractive	attr									
Sincere	sinc									
Intelligent	intel									
Fun	fun									
Ambitious	amb									
Shared Interests/Hobbies	shar									

Overall, how much do you like this person? (1=don't like at all, 10=like a lot)	like									

We have two different models for men and women.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sbn
        import statsmodels.api as sma
```

```
C:\Users\SUUSER\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The pandas.cor
odule is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

```
In [2]: import xlrd
        xlsfile = pd.ExcelFile('SPEED.xls')

        DF = xlsfile.parse('Sheet1')
```

Use only the needed keys:

```
In [23]: input_vars= ['attr', 'sinc', 'intel', 'fun', 'amb', 'shar', 'like', 'prob']
```

Female model:

```
In [39]: #female model
         f = DF.loc[DF.gender == 0, :]
         f_data = f.copy()
         f_data = f.dropna(subset=input_vars)
         f_model = sma.OLS(f_data.dec, sma.add_constant(f_data.loc[:, input_vars]))
         f_results = f_model.fit()
         f_results.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	dec	**R-squared:**	0.285
Model:	OLS	**Adj. R-squared:**	0.283
Method:	Least Squares	**F-statistic:**	169.5
Date:	Sun, 13 May 2018	**Prob (F-statistic):**	2.87e-241
Time:	01:53:02	**Log-Likelihood:**	-1785.9
No. Observations:	3409	**AIC:**	3590.
Df Residuals:	3400	**BIC:**	3645.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.3958	0.036	-11.121	0.000	-0.466	-0.326
attr	0.0496	0.005	10.220	0.000	0.040	0.059
sinc	-0.0248	0.005	-4.583	0.000	-0.035	-0.014
intel	0.0101	0.007	1.472	0.141	-0.003	0.024
fun	0.0204	0.005	3.783	0.000	0.010	0.031
amb	-0.0208	0.005	-4.020	0.000	-0.031	-0.011
shar	0.0236	0.005	5.041	0.000	0.014	0.033
like	0.0615	0.006	9.469	0.000	0.049	0.074
prob	0.0182	0.004	4.849	0.000	0.011	0.026

Omnibus:	1316.065	Durbin-Watson:	1.456
Prob(Omnibus):	0.000	Jarque-Bera (JB):	197.731
Skew:	0.225	Prob(JB):	1.16e-43
Kurtosis:	1.909	Cond. No.	93.7

As expected, "like" has the biggest probability 0.615 of saying "yes" to the partner and the most powerful attribute that affects the decision of saying yes is "attractiveness" with the coefficient 0.0496 for women.

Male model:

In [40]:

```
#male model
f = DF.loc[DF.gender == 1, :]
f_data = f.copy()
f_data = f.dropna(subset=input_vars)
f_model = sma.OLS(f_data.dec, sma.add_constant(f_data.loc[:, input_vars]))
f_results = f_model.fit()
f_results.summary()
```

Out[40]:

OLS Regression Results

Dep. Variable:	dec	R-squared:	0.368
Model:	OLS	Adj. R-squared:	0.367
Method:	Least Squares	F-statistic:	258.3
Date:	Sun, 13 May 2018	Prob (F-statistic):	0.00
Time:	01:54:55	Log-Likelihood:	-1762.4
No. Observations:	3554	AIC:	3543.
Df Residuals:	3545	BIC:	3598.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.4309	0.037	-11.712	0.000	-0.503	-0.359
attr	0.0812	0.005	16.155	0.000	0.071	0.091
sinc	-0.0380	0.006	-6.559	0.000	-0.049	-0.027
intel	-0.0100	0.007	-1.463	0.144	-0.023	0.003
fun	0.0180	0.005	3.284	0.001	0.007	0.029
amb	-0.0215	0.005	-4.117	0.000	-0.032	-0.011
shar	0.0111	0.005	2.442	0.015	0.002	0.020
like	0.0916	0.007	13.992	0.000	0.079	0.104
prob	0.0253	0.004	6.559	0.000	0.018	0.033

Omnibus:	387.341	Durbin-Watson:	1.503
Prob(Omnibus):	0.000	Jarque-Bera (JB):	120.228
Skew:	-0.153	Prob(JB):	7.81e-27
Kurtosis:	2.153	Cond. No.	103.

As expected, "like" has the biggest probability 0.0916 of saying "yes" to the partner and the most powerful attribute that affects the decision of saying yes is "attractiveness" with the coefficient 0.0812 for women.

Our GitHub project repository link: cs210-project

March 30, 2018 Sevde Bozdogan Leave a comment

Hypothesis Testing

In this step, to understand and analyze our data better, we will perform one hypothesis testing about the women's and men's perception on some of the six attributes(attractive, sincere, intelligent, fun, ambitious, shared interests) during speed date. We determined our alternative hypothesis that we want to investigate about and the null hypothesis as;

Ho: Women and men give the same importance to the "attractiveness" of their matches during a speed date.

Ha: Women and men do not give the same importance to the "attractiveness" of their matches during a speed date.

The link to the speed dating experiment's data key document: https://www.kaggle.com/annavictoria/speed-dating-experiment/data (Speed Dating Data Key.doc) This document explains which column name corresponds to what in our data SPEED.xls.

Note: we changed the extension of our data SPEED.csv to SPEED.xls because of the reading problems of data in JupyterLab.

```
In [76]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns

         %matplotlib inline
```

```
In [77]: import xlrd
         xlsfile=pd.ExcelFile('SPEED.xls')

         #workbook = xlrd.open_workbook('myxls.xls')
         DF =xlsfile.parse('Sheet1')
```

```
In [78]: DF
```

Out[78]:

	iid	id	gender	idg	condtn	wave	round	position	positin1	order	...	attr3_3	sinc3_3	intel3_3	fun3_3	amb3_3	attr5_3	si
0	1	1.0	0	1	1	1	10	7	NaN	4	...	5.0	7.0	7.0	7.0	7.0	NaN	
1	1	1.0	0	1	1	1	10	7	NaN	3	...	5.0	7.0	7.0	7.0	7.0	NaN	
2	1	1.0	0	1	1	1	10	7	NaN	10	...	5.0	7.0	7.0	7.0	7.0	NaN	
3	1	1.0	0	1	1	1	10	7	NaN	5	...	5.0	7.0	7.0	7.0	7.0	NaN	
4	1	1.0	0	1	1	1	10	7	NaN	7	...	5.0	7.0	7.0	7.0	7.0	NaN	
5	1	1.0	0	1	1	1	10	7	NaN	6	...	5.0	7.0	7.0	7.0	7.0	NaN	
6	1	1.0	0	1	1	1	10	7	NaN	1	...	5.0	7.0	7.0	7.0	7.0	NaN	
7	1	1.0	0	1	1	1	10	7	NaN	2	...	5.0	7.0	7.0	7.0	7.0	NaN	
8	1	1.0	0	1	1	1	10	7	NaN	8	...	5.0	7.0	7.0	7.0	7.0	NaN	

During the experiment, the question "You have 100 points to distribute among the following attributes — give more points to those attributes that are more important in a potential date, and fewer points to those attributes that are less important in a potential date. Total points must equal 100." is asked to the attendees. We used the answers to this questions -distributed points to the six attributes- to test our hypothesis.

attr1_1 +

sinc1_1 +

intel1_1 +

fun1_1 +

amb1_1 +

shar1_1 +

100

In the data key document, “attr1_1” corresponds to the point of “attractiveness” attribute under this question. In the gender column, “0” corresponds to the “female” and “1” corresponds to the “male”.

We divided the attractiveness scores that women and men give that correspond to the importance of the attribute:

```
In [6]: w_attr=[] # attr1_1 points for women
        b_attr=[] # attr1_1 points for men
```

```
In [7]: for i in range (0,8378):
            if(DF['gender'][i]==0):
                w_attr.append(DF['attr1_1'][i])
            else:
                b_attr.append(DF['attr1_1'][i])
```

Then, we balanced the data number for men and women to compare accurately:

```
In [9]: len(w_attr)

Out[9]: 4184

In [10]: len(b_attr)

Out[10]: 4194
```

```
In [13]: b_attr = b_attr[0:4184]
```

```
In [14]: len(b_attr)
Out[14]: 4184
```

Cross tab for attractiveness scores of women/men:

```
In [82]: def change_gender(gender):
             if gender > 0:
                 return 'male'
             else:
                 return 'female'
```

```
In [83]: DF['gender'] = DF['gender'].apply(change_gender)
```

```
In [84]: table = pd.crosstab(index=[DF['attr1_1']], columns=DF['gender'])
         table
```

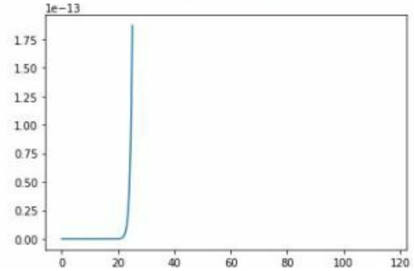
Out[84]:

gender / attr1_1	female	male
0.00	21	0
2.00	9	0
5.00	60	0
6.67	5	14
7.00	21	0
7.50	0	20
8.00	19	0
8.33	16	0
8.51	16	0
9.00	18	0

From the cross tab, we can easily see that **women and men differ in scoring the importance of "attractiveness" attribute in their match**, but to complete our hypothesis testing, we need to apply chisquare and t-tests.

```
In [260]: import scipy.stats as stats

In [261]: chi_stats = stats.chi2_contingency(freq)
          chi_stats

Out[261]: (3312.5118939701242, 0.0, 93, array([[  10.43800458,   10.56199542],
                  [   4.47343053,    4.52656947],
                  [  29.82287023,   30.17712977],
                  [   9.4439089 ,    9.5560911 ],
                  [  10.43800458,   10.56199542],
                  [   9.94095674,   10.05904326],
                  [   9.4439089 ,    9.5560911 ],
                  [   7.95276539,    8.04723461],
                  [   7.95276539,    8.04723461],
                  [   8.94686107,    9.05313893],
                  [   4.97047837,    5.02952163],
                  [  14.91143511,   15.08856489],
                  [   7.95276539,    8.04723461],
                  [ 401.11760453,  405.88239547],
                  [   7.95276539,    8.04723461],
                  [   7.95276539,    8.04723461],
                  [   9.94095674,   10.05904326],
                  [  32.80515725,   33.19484275],
                  [   4.97047837,    5.02952163],
```

```
In [262]: alpha = 0.05

In [263]: critical_value = crit = stats.chi2.ppf(q = 1 - alpha, #find the critical value for 95% level confidence
                                              df = chi_stats[2])
          critical_value

Out[263]: 116.51104728087354

In [264]: x = np.linspace(0, 25, 2000)
          plt.plot(x, stats.chi2.pdf(x, chi_stats[2]))
          plt.axvline(x=critical_value, ymin = 0.05, ymax = 0.05, c = 'r')
          plt.annotate('Critical Value = {0:.2f}'.format(critical_value), xy=(critical_value, 0.02), xytext=(critical_value, 0.04
                       arrowprops=dict(facecolor='black', shrink=0.5), verticalalignment='top')
          plt.fill_between(x, stats.chi2.pdf(x, chi_stats[2]), where= x > critical_value, facecolor='red', interpolate= True)

Out[264]: <matplotlib.collections.PolyCollection at 0xd1ddeb8>
```



Fill the NaN values with means:

```
In [264]: from pandas import Series
          women_attr = pd.Series(w_attr)
          boy_attr=pd.Series(b_attr)

In [ ]:

In [265]: women_attr=women_attr.fillna(women_attr.mean())
          boy_attr=boy_attr.fillna(boy_attr.mean())
```

Perform the chi-square test:

```
In [267]: (t, p) = stats.chisquare(women_attr, boy_attr, ddof=1)
          print ('Test t=%f p-value = %f' % (t, p))

          alpha = 0.05  # significance level

Test t=35842.684631 p-value = 0.000000
```

Hypothesis testing:

Our hypothesis was **two tailed** in a form:

H1: μ M ≠ μ W   (alternative)

H0: μ M = μ W   (null)

(μ M corresponds to the points men gave for attractiveness and μ W corresponds to the points women gave for attractiveness)

```
In [25]: #two-tailed test
if p <= alpha/2:
    # we reject null hypothesis
    print ('Null hypothesis  is unlikely to except')
else:
    # we reject alternative hypothesis
    print ('Null hypothesis cannot be rejected')

Null hypothesis  is unlikely to except
```

As you can see from the above results, Chi square test does not give the appropriate result to test our hypothesis **(p-value** = **0)** -although it rejected the null hypothesis-, so **we changed our solution method to t-test:**

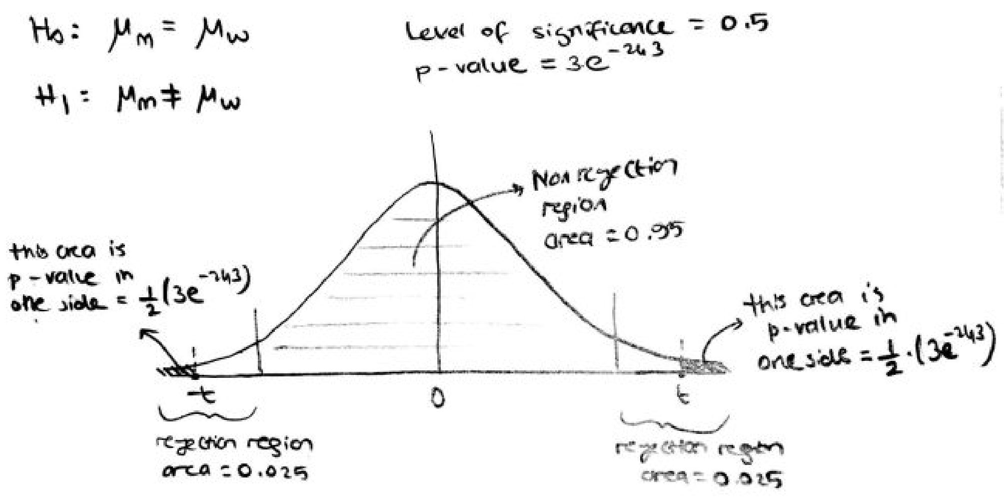
```
In [58]: from scipy import stats

In [59]: t_stat, p_val = stats.ttest_ind(women_attr,boy_attr)
t_stat, p_val

Out[59]: (-34.43947106034647, 3.4380654612314287e-243)
```

```
In [26]: #two-tailed test
if p_val <= alpha/2:
    # we reject null hypothesis
    print ('Null hyptohesis is unlikely to except')
else:
    #we reject alternative hypothesis
    print ('Null hypothesis cannot be rejected')

Null hyptohesis is unlikely to except
```

After t-test, we found our p-value approximately equals to **3.4*e^-243 which is really close to 0**, so it is smaller than the alpha values 0.025 for left and right. At the end, we must **reject the null**

hypothesis.

Explanation in the **rejection region figure**:



That means, **men give much more importance to the attractiveness of their match during a speed date when compared with women, so we reject the null hypothesis.**

Our GitHub project repository link: cs210-project

March 30, 2018 Sevde Bozdogan Leave a comment

Exploring and Describing Our Data

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

In [2]: %matplotlib inline

In [3]: import xlrd
        xlsfile = pd.ExcelFile('SPEED.xls')

        #workbook = xlrd.open_workbook('myxls.xls')
        DF = xlsfile.parse('Sheet1')

In [4]: DF
```

Out[4]:

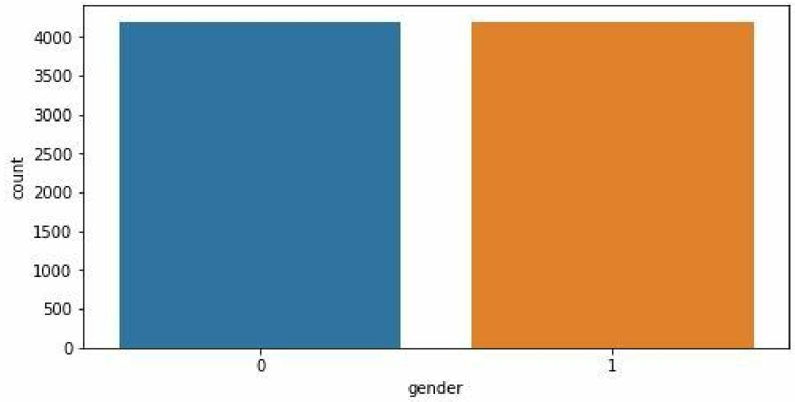
	iid	id	gender	idg	condtn	wave	round	position	positin1	order	...	attr3_3	sinc3_3	intel3_3	fun3_3	amb3_3	attr5_3	si
0	1	1.0	0	1	1	1	10	7	NaN	4	...	5.0	7.0	7.0	7.0	7.0	NaN	
1	1	1.0	0	1	1	1	10	7	NaN	3	...	5.0	7.0	7.0	7.0	7.0	NaN	
2	1	1.0	0	1	1	1	10	7	NaN	10	...	5.0	7.0	7.0	7.0	7.0	NaN	
3	1	1.0	0	1	1	1	10	7	NaN	5	...	5.0	7.0	7.0	7.0	7.0	NaN	
4	1	1.0	0	1	1	1	10	7	NaN	7	...	5.0	7.0	7.0	7.0	7.0	NaN	
5	1	1.0	0	1	1	1	10	7	NaN	6	...	5.0	7.0	7.0	7.0	7.0	NaN	
6	1	1.0	0	1	1	1	10	7	NaN	1	...	5.0	7.0	7.0	7.0	7.0	NaN	
7	1	1.0	0	1	1	1	10	7	NaN	2		5.0	7.0	7.0	7.0	7.0	NaN	

Shape of our data:

```
In [27]: DF.shape
Out[27]: (8378, 195)
```

Check for the number of women attendees and men attendees:

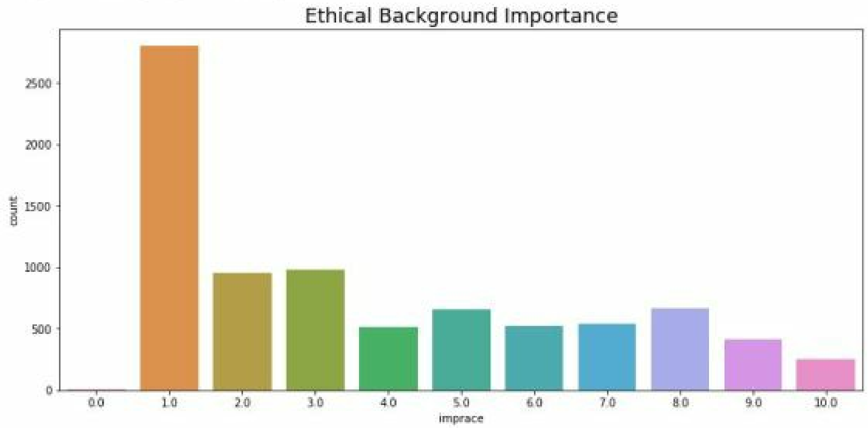
```
In [392]: plt.figure(figsize=(8,4))
          sns.countplot(x='gender', data=DF)

Out[392]: <matplotlib.axes._subplots.AxesSubplot at 0xfa40e10>
```



Look into the “importance of sharing the same ethical background” –> use “imprace”. From the data key document, it corresponds to the answer to the following question:

How important is it to you (on a scale of 1-10) that a person you date be of the same racial/ethnic background?

```
In [393]: plt.figure(figsize=(13,6))
          plt.title('Ethical Background Importance', fontsize=18)
          sns.countplot(DF['imprace'])

Out[393]: <matplotlib.axes._subplots.AxesSubplot at 0xfa40ef0>
```



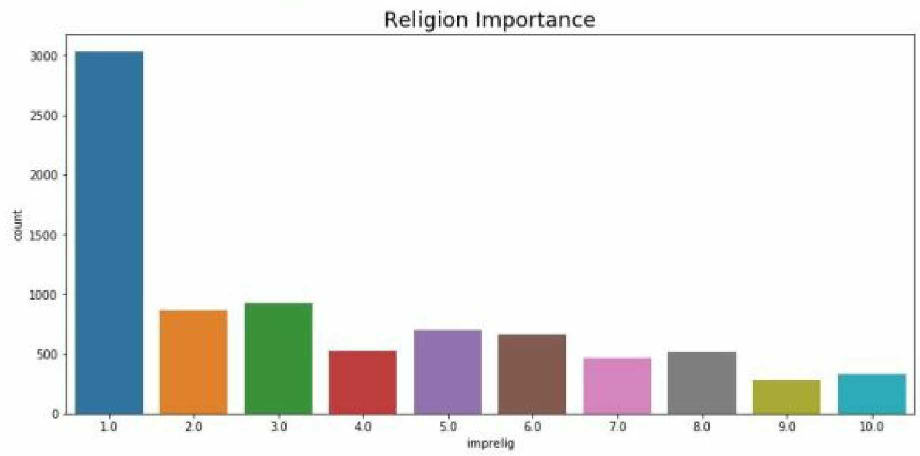
A majority of the attendees sees the “same ethical background” not much important in their decisions during a speed date.

Look into the “importance of sharing the same religion” –> use “imprelig”. From the data key document, it corresponds to the answer to the following question:

How important is it to you (on a scale of 1-10) that a person you date be of the same religious background?

In [394]:
```
plt.figure(figsize=(13,6))
plt.title('Religion Importance', fontsize=18)
sns.countplot(DF['imprelig'])
```

Out[394]: <matplotlib.axes._subplots.AxesSubplot at 0xd0ae710>



A majority of the attendees sees the “same religion” not much important in their decisions during a speed date.

Mean values of data keys for women:

In [28]:
```
DF[DF.gender==0].mean()
```

Out[28]:
```
iid          275.430210
id             9.024140
gender         0.000000
idg           16.958652
condtn         1.829828
wave          11.358509
round         16.782027
position       9.048757
positin1       9.262707
order          8.891013
partner        8.907744
pid          292.297323
match          0.164914
int_corr       0.196300
samerace       0.396272
age_o         26.621901
race_o         2.732949
pf_o_att      26.893883
pf_o_sin      16.497231
pf_o_int      19.545869
pf_o_fun      17.769880
pf_o_amb       8.554378
pf_o_sha      10.996568
dec_o          0.474665
attr_o         6.461401
sinc_o         7.251053
intel_o        7.291202
fun_o          6.520164
amb_o          6.604591
```

Mean values of data keys for men:

In [29]:
```
DF[DF.gender==1].mean()
```

Out[29]:
```
iid          291.902003
id             8.896494
gender         1.000000
idg           17.694802
condtn         1.827849
wave          11.343348
round         16.961850
position       9.036719
positin1       9.328843
order          8.964235
partner        9.019313
pid          275.430210
match          0.164521
int_corr       0.195721
samerace       0.395327
age_o         26.105851
race_o         2.780488
pf_o_att      18.055224
pf_o_sin      18.305008
pf_o_int      21.002502
pf_o_fun      17.147292
pf_o_amb      12.827222
pf_o_sha      12.704194
dec_o          0.364568
attr_o         5.919422
sinc_o         7.099778
intel_o        7.447362
fun_o          6.280555
amb_o          6.952773
```

The mode for the ages of attendees:

In [30]:
```
DF['age'].mode()
```

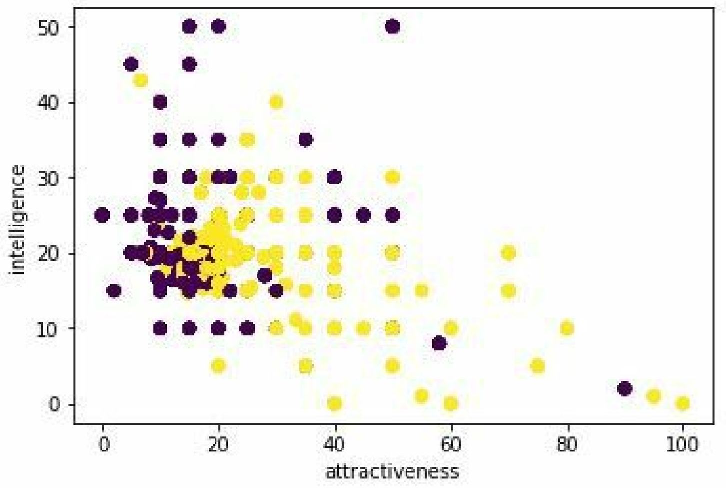
Out[30]:
```
0    27.0
dtype: float64
```

We have a young population for the experiment.

Scatter plot trials:

“attractiveness” and “intelligence” scores scatter plot for different genders:

```
In [17]: plt.scatter(DF.attr1_1, DF.intel1_1, c = DF.gender)
         plt.xlabel('attractiveness')
         plt.ylabel('intelligence');
```



yellow dots:men & purple dots:women

From this scatter plot, we can easily obtain that men tend to give more importance to attractiveness when compared with intelligence of their matches during speed date.

Our GitHub project repository link: cs210-project

March 20, 2018 Sevde Bozdogan Leave a comment

Follow

Speed Date Predictions

Create a free website or blog at WordPress.com.