Machine Learning Fundamentals:

Exploring the OKCupid dataset

Rebecca Green

5/20/19

(Apr 2 cohort)

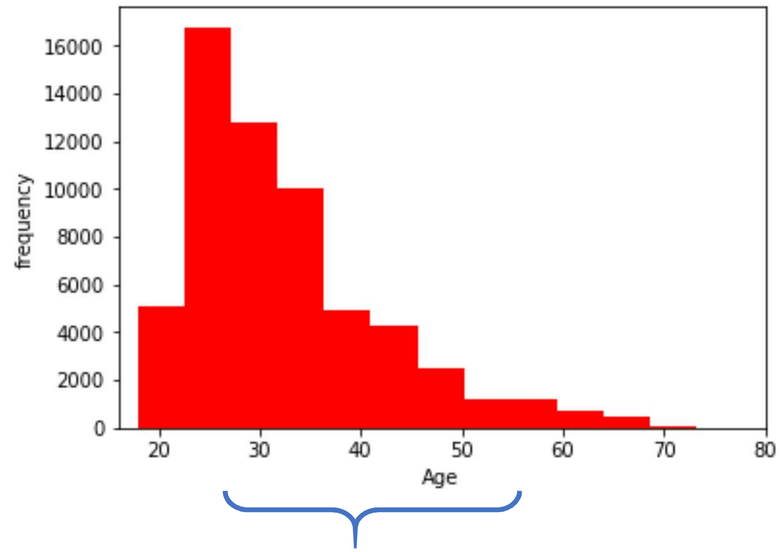
The OKcupid dataset contains dating profile and demographic information on ~60,000 individuals

- Profiles are described by the following columns:
  - **Age**
  - Body type
  - Diet
  - **Drinks**
  - Drugs
  - **Education**
  - Ethnicity
  - **Height**
  - **Religion**
  - **Income**
  - **Sexual Orientation**
  - Pets
  - **Sex**
  - Zodiac Sign
  - Smokes
  - Speaks (Languages)
  - Status
  - **+ 9 short essay questions**

**** Bolded categories were analyzed in this project ****

Exploring the data: *The dataset is largely composed of Millennials and GenX individuals*

```python
#general age distribution in dataset
plt.hist(df.age, bins=20, color="red")
plt.xlabel("Age")
plt.ylabel("frequency")
plt.xlim(16,80)
plt.show()
```

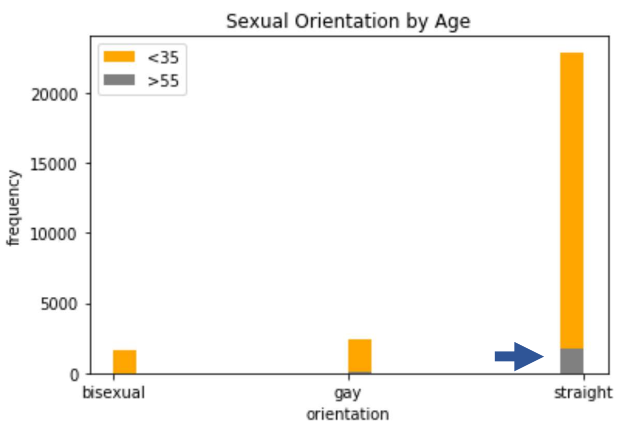


Exploring the data:

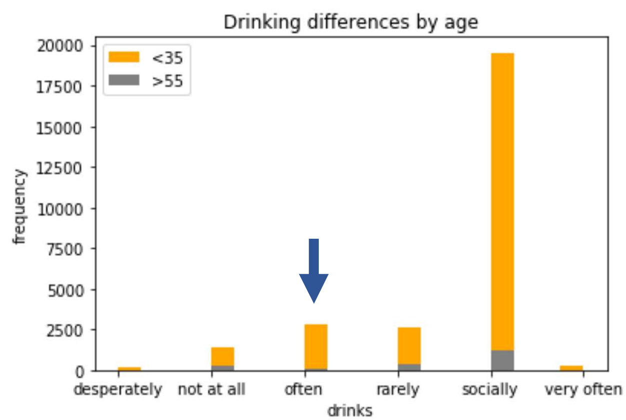
Millennials (less than 35) vs. Boomers(55+) demographic

**** there are significantly <u>more Millennials</u> in the dataset, than Boomers.**
Further analysis of generational trends would require normalization to the total for each category.

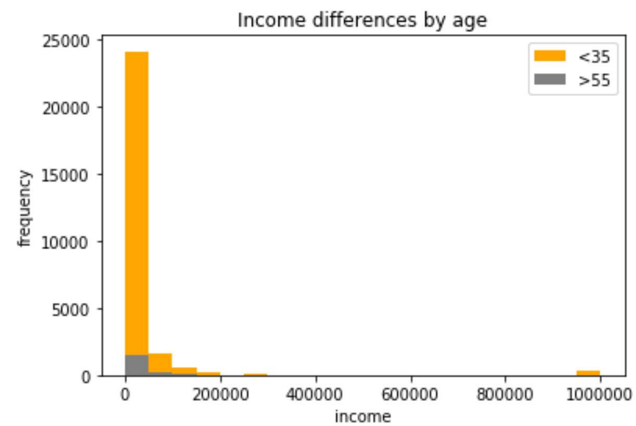
Baby boomers in database are almost all straight



Millennials are much more likely to drink "often" than Boomers



Millennials and Boomers are both likely to earn <$50,000/year



Populations identified using pandas .loc and nans dropped

```
# identify indicies for young and old cohort
young_inds = df.loc[df['age'] < 36]
young_inds = young_inds.dropna(subset=['sex','height','age','income','orientation','drinks','religion'])
old_inds = df.loc[df['age'] > 54]
old_inds = old_inds.dropna(subset=['sex','height','age','income','orientation','drinks','religion'])
# print(len(young_inds))
# print(len(old_inds))
```

plots generated with Matplotlib plt.hist()

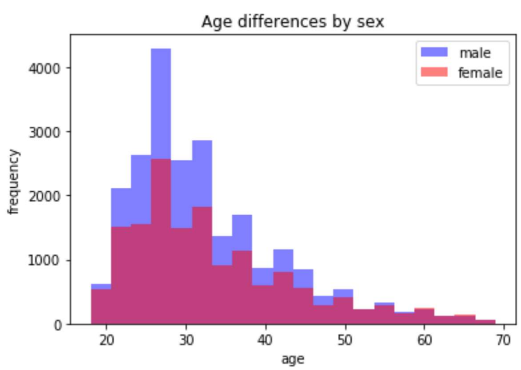
```
plt.hist(young_inds.income,bins=20, color="orange")
plt.hist(old_inds.income,bins=20, color="grey")
plt.xlabel("income")
plt.ylabel("frequency")
plt.legend(('<35', '>55'),loc='upper right')
plt.title("Income differences by age")
plt.show()
```

Exploring the OKcupid data set:
Male vs. Female demographic

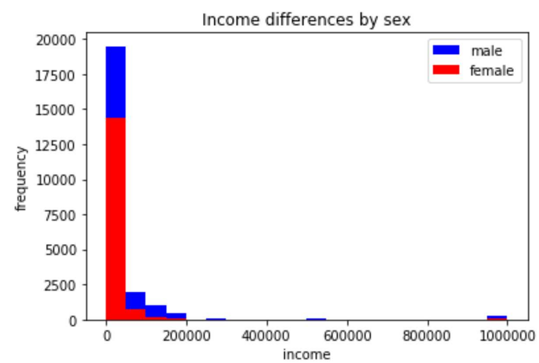
There are <u>more men</u> in the dating pool than women (35829- male vs 24117- female)
A direct comparison would require normalization by total number in each population

What **similar trends** can be observed between the sexes in the dataset?

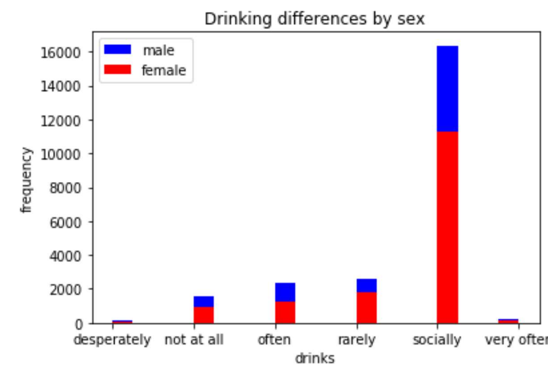
Age distribution is similar between the sexes



Income distribution is similar between the sexes



Drinking habits are similar between the sexes



Populations identified using pandas .loc and nans dropped

```
# identify indicies for male and female cohort
print(df.sex.value_counts())
male_inds = df.loc[df['sex'] == 'm']
male_inds = male_inds.dropna(subset=['sex','height','age','income','orientation','drinks','religion'])
female_inds = df.loc[df['sex'] == 'f']
female_inds = female_inds.dropna(subset=['sex','height','age','income','orientation','drinks','religion'])
```

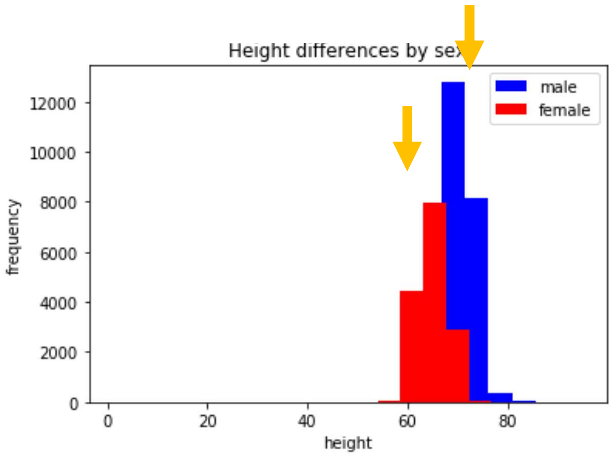
plots generated with Matplotlib plt.hist()

```
plt.hist(male_inds.height,bins=20, color="blue")
plt.hist(female_inds.height,bins=20, color="red")
plt.xlabel("height")
plt.ylabel("frequency")
plt.legend(('male', 'female'),loc='upper right')
plt.title("Height differences by sex")
plt.show()
```

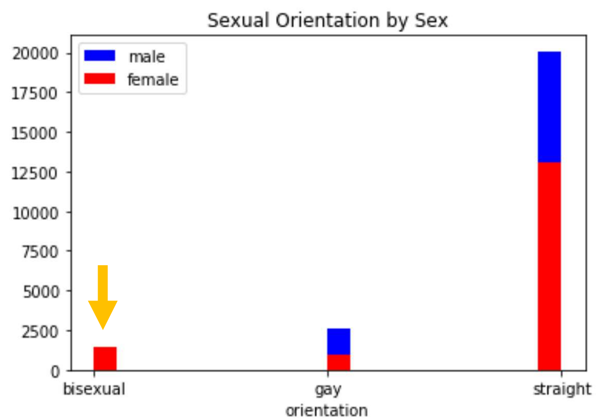
Exploring the OKcupid data set:
Male vs. Female demographic

What **differences** can be observed between the sexes in the dataset?

Males tend to be taller than females



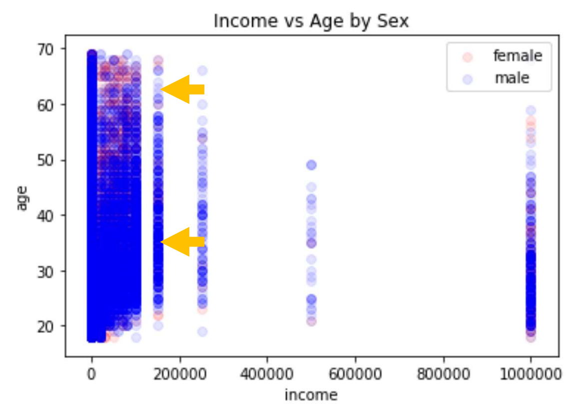
Women are more likely to be bisexual than men



Mid income ($20,000-100,000) workers show age differences by sex

Male workers are mostly 20-50

Sig. population of female workers are 50-70



Augmenting the dataset with additional columns

Multiple choice questions were mapped to numerical values to enable min-max normalization

- Columns converted:
  - Drinks?
  - Drugs?
  - Sex?
  - Orientation?
  - Status?
  - Income?

```python
# mapping
drink_mapping = {"not at all":0, "rarely":1, "socially":2, "often":3, "very often": 4, "desperately":5}
df["drinks_code"]= df.drinks.map(drink_mapping)
# print(df.drinks_code.value_counts())

# print(df.smokes.value_counts())
smokes_mapping = {"no":0, "sometimes":1, "when drinking":2, "yes":3, "trying to quit": 4}
df["smokes_code"]= df.smokes.map(smokes_mapping)
# print(df.smokes_code.value_counts())

# print(df.drugs.value_counts())
drugs_mapping = {"never":0, "sometimes":1, "often":2}
df["drugs_code"]= df.drugs.map(drugs_mapping)
# print(df.drugs_code.value_counts())
```

```python
# print(df.sex.value_counts())
sex_mapping = {"m":0, "f":1}
df["sex_code"]= df.sex.map(sex_mapping)
# print(df.sex_code.value_counts())

# print(df.orientation.value_counts())
orientation_mapping = {"straight":0, "gay":1, "bisexual":2}
df["orientation_code"]= df.orientation.map(orientation_mapping)
# print(df.orientation_code.value_counts())

# print(df.status.value_counts())
status_mapping = {"single":0, "seeing someone":1, "available":2, "married":3, "unknown":4}
df["status_code"]= df.status.map(status_mapping)
# print(df.status_code.value_counts())

# print(df.income.value_counts())
income_mapping = {-1:0, 20000:1, 30000:2, 40000:3, 50000:4, 60000:5, 70000:6, 80000:7, 100000:8, 150000:9, 250000:10, 500000:11,
df["income_code"]= df.income.map(income_mapping)
# print(df.income_code.value_counts())
```

Complex multiple choice questions were generalized and mapped to numerical values in new columns

- <u>Columns:</u> Religion and Education columns had too many nuanced choices.
- I mapped out more general categories to capture the overall trend
  - i.e. people who had 'started college' were lumped in with people who 'completed college' etc.

** It would probably be better to map each response on a continuum within each broad category (maybe build a dictionary for each religion or education subcategory), so that the nuances were not lost, however, this would take some time to program in.*

```
religion_mapping = {}
for x in all_religions:  # map religion generally, remove all the modifiers
#     print(x)
    if "atheism" in x:
        religion_mapping[x] = 0
    elif "agnosticism" in x:
        religion_mapping[x] = 1
    elif "christianity" in x:
        religion_mapping[x] = 2
    elif "catholicism" in x:
        religion_mapping[x] = 3
    elif "judaism" in x:
        religion_mapping[x] = 4
    elif "buddhism" in x:
        religion_mapping[x] = 5
    elif "islam" in x:
        religion_mapping[x] = 6
    elif "other" in x:
        religion_mapping[x] = 7
# print(religion_mapping)
df["religion_code"]= df.religion.map(religion_mapping)
# print(df.religion_code.value_counts())
```

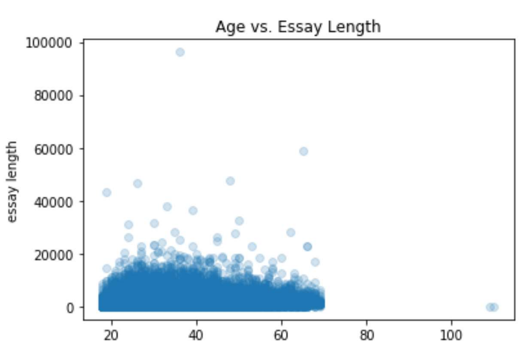
```
education_mapping = {}
for x in all_education:  # map education generally, remove all the modifiers
#         print(x)
    if "high school" in x:
        education_mapping[x] = 0
    elif "two-year" in x:
        education_mapping[x] = 1
    elif "college/university" in x:
        education_mapping[x] = 2
    elif "masters program" in x:
        education_mapping[x] = 3
    elif "med school" in x:
        education_mapping[x] = 4
    elif "law school" in x:
        education_mapping[x] = 5
    elif "ph.d program" in x:
        education_mapping[x] = 6
    elif "space camp" in x:
        education_mapping[x] = 7
# print(education_mapping)
df["education_code"]= df.education.map(education_mapping)
# print(df.education_code.value_counts())
```

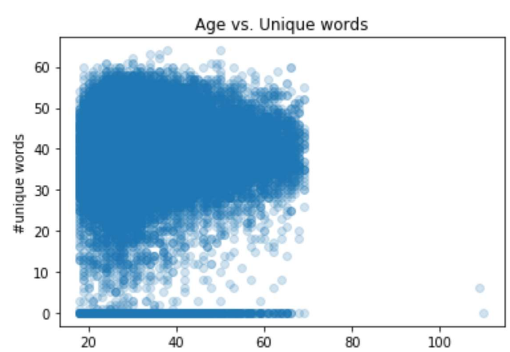
Text fields were combined, evaluated and new columns were mapped using lambda operator

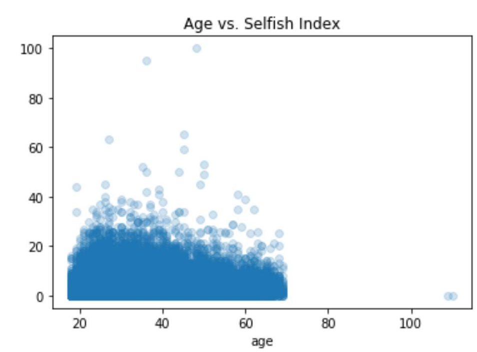
- Essay Length
- Number of unique words
- Selfish word count- number of times the words “me” or “I” were used

```
# mapping essay columns
essay_cols = ["essay0","essay1","essay2","essay3","essay4","essay5","essay6","essay7","essay8","essay9"]
all_essays = df[essay_cols].replace(np.nan,"",regex=True)  # remove all nans from the essay responses
# print(df[essay_cols].head())

all_essays = all_essays[essay_cols].apply(lambda x:"".join(x), axis=1)  # combine all essay answers
df["essay_len"] = all_essays.apply(lambda x: len(x))  # create a new column that reports the length of all essay answers
# print(df.essay_len[0])
df["unique_words"] = all_essays.apply(lambda x: len(set(x))) # create a new column that reports the number of unique words
df["selfish_index"] = all_essays.apply(lambda x: x.count("me ") + x.count("I ")) # create a new column that reports use of I and
# print(df.selfish_index[0])
```

Young people appear *slightly* more likely to use more unique words, write longer essay responses, And use ‘self’ish terms







Features to consider were defined and min-max normalization was performed on the dataset

```
#define features to consider
mapped_columns = ["orientation_code", "sex_code", "religion_code", "drugs_code", "smokes_code", "drinks_code", "age", "status_coc
essay_columns_to_use = ["essay_len","unique_words","selfish_index"]
features_to_use = essay_columns_to_use + mapped_columns  # define list of columns to consider
features = df[features_to_use]
print(features.head())

features = features.dropna()  # remove rows with nans
```

```
#normalize data
from sklearn import preprocessing

x=features.values
min_max_scale = preprocessing.MinMaxScaler()
x_scaled = min_max_scale.fit_transform(x)
features = pd.DataFrame(x_scaled, columns=features.columns)
```

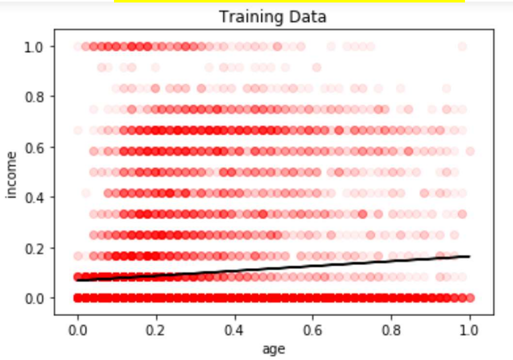
Regression Analysis

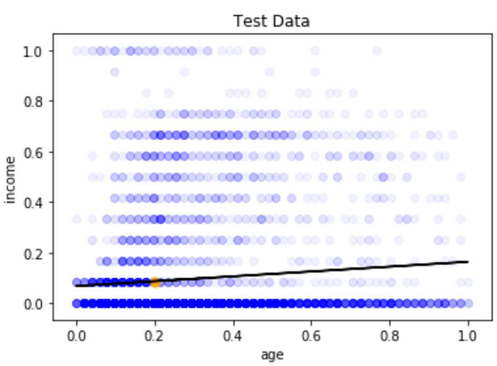
-single linear regression
-multiple linear regression

Simple linear regression: Can we predict income from age?

All Data

Model Score = 0.0069





** Axis are coded and normalized*

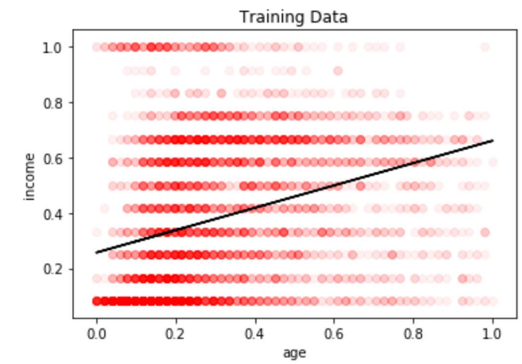
- Income vs age data was dominated by people who reported no income, which skewed the regression

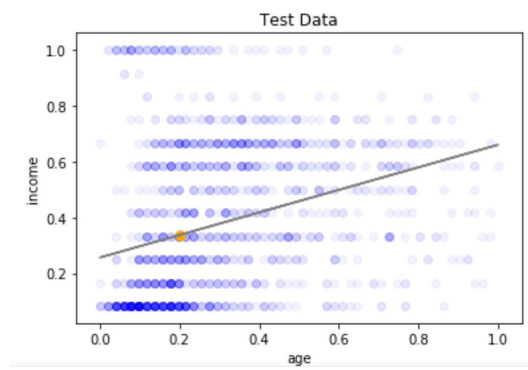
- I eliminated these rows and re-trained only on people who answered the income question- the predictive value was considerably improved!

```
# regression- can we predict income from age?

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# x=features[['age']]
# y=np.array(features.income_code).reshape(-1,1)

non_zero_inds = features.loc[features['income_code'] != 0]
x = non_zero_inds[['age']]
y=np.array(non_zero_inds.income_code).reshape(-1,1)

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8,
model = LinearRegression()
model.fit(x_train,y_train)
print(model.coef_)
print(model.intercept_)
pred = model.predict(x_test)

d= {'age': [0.2]}
test = pd.DataFrame(data=d)
pred2 = model.predict(test)
print(pred2)

plt.scatter(x_train['age'], y_train, color="red", alpha=0.05)
plt.plot(x_train['age'], pred, color="k", alpha=1)
plt.xlabel('age')
plt.ylabel("income")
plt.title("Training Data")
plt.show()
```

Remove individuals with no reported income

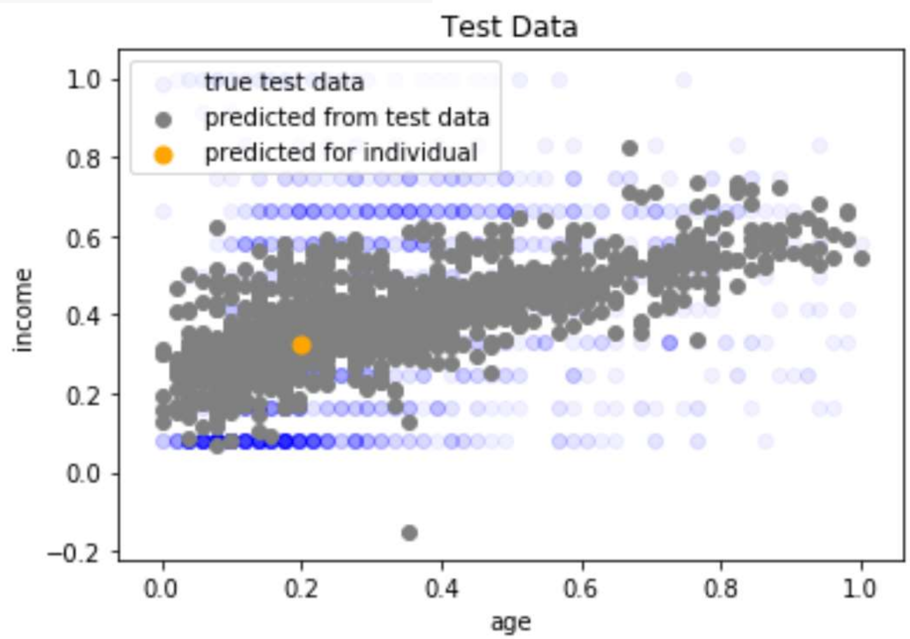
Model Score = 0.069
(10 fold improvement)





<u>Multiple linear regression</u>: Can we predict income from age, orientation, education, selfish_index, and height?

```
# multiple linear regression- can we predict income better when we have many more variables defined?

x=non_zero_inds[['age','orientation_code', 'education_code', 'selfish_index', 'height']]
y=np.array(non_zero_inds.income_code).reshape(-1,1)

x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=6)
model = LinearRegression()
model.fit(x_train,y_train)
print(model.coef_)
# print(model.intercept_)
pred = model.predict(x_test)
print("score")
print(model.score(x_test,y_test))
pred_train = model.predict(x_train)

# pred_color = ["blue" if x < 0.4 else "red" for x in pred]
plt.scatter(x_train['age'], y_train, color="red", alpha=0.05)
plt.scatter(x_train['age'], pred_train, color="k", alpha=1)
plt.xlabel('age')
plt.ylabel("income")
plt.title("Training Data")
plt.show()

plt.scatter(x_test['age'],y_test, color="blue", alpha=0.05)
plt.scatter(x_test['age'],pred, color="grey", alpha=1)
plt.xlabel('age')
plt.ylabel("income")
plt.title("Test Data")
# print(pred[0])
```



Remove individuals with no reported income And include multiple Variables for fitting:

Model Score = 0.128 (additional 2 fold improvement)

Classification Analysis

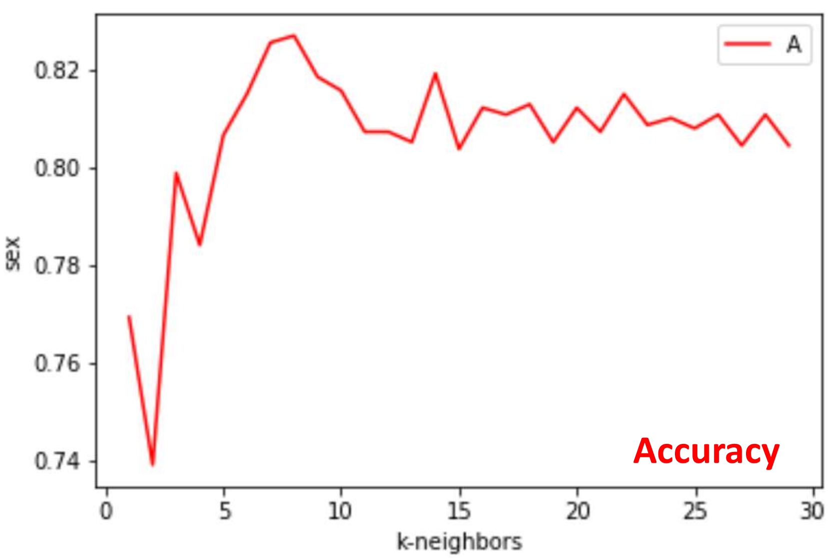
-K nearest neighbor (supervised)

-K-means (unsupervised)

K-nearest neighbor classification to predict sex from age, income, education, height, sexual orientation and 'selfish' language

- KNN uses the class of the k nearest points to predict the class of the test points

Using 8 neighbors, we can get 83% accuracy of predicting the correct sex in the test set

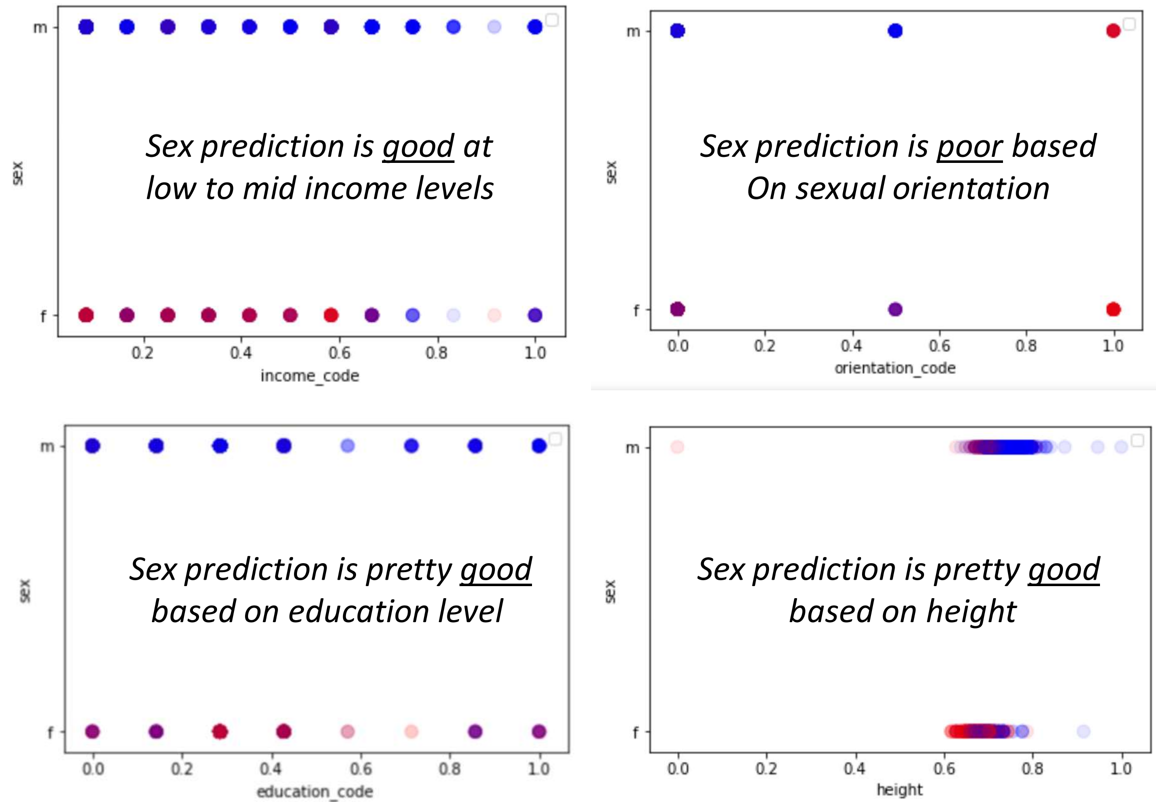



```
#define x,y to split for training
x = non_zero_inds[['age','income_code', 'education_code', 'selfish_index', 'height','orientation_code']]
y = non_zero_inds.sex
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8, test_size = 0.2, random_state=6)
print('done')

points = x_train[['age','income_code', 'education_code', 'selfish_index', 'height','orientation_code']]
labels = y_train

i_list = []
ac_list = []
f1_list = []
pr_list = []

#find best k value for classifier
for i in range(1,30):
    classifier = KNeighborsClassifier(n_neighbors=i)  # optimal i is 65
    classifier.fit(points,labels)

    #test classifier
    guess = classifier.predict(x_test)

    ac = accuracy_score(y_test,guess)
    re = recall_score(y_test,guess, average=None)
    pr = precision_score(y_test,guess,average=None)
    f1 = f1_score(y_test,guess,average=None)
    i_list.append(i)
    ac_list.append(ac)
    f1_list.append(f1)
    pr_list.append(pr)
    print("the F1 metric is {0}, accuracy is {1}, recall is {2}, and precision is {3}").format(f1,ac,re,pr)
```

K-nearest neighbor classifier performance by descriptor

Predicted Male
Predicted Female

Ground Truth

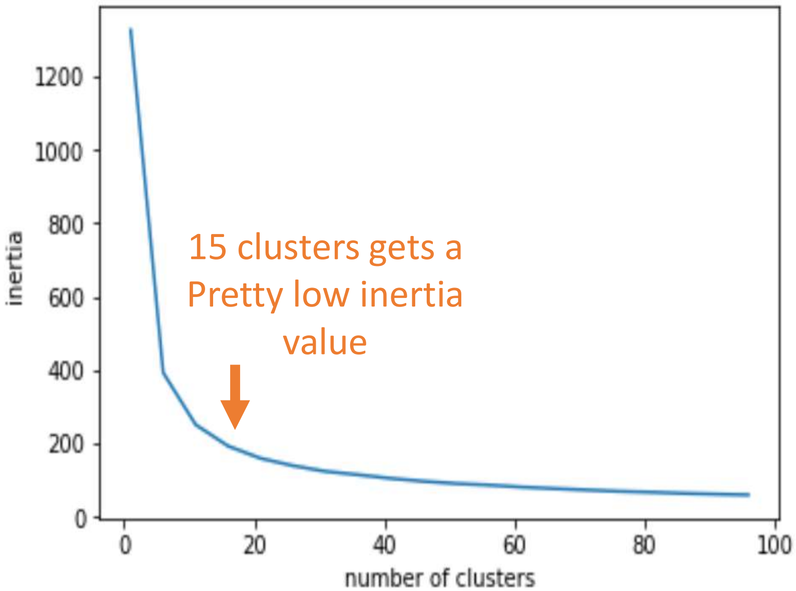



```
'''use the best k value going forward'''
classifier = KNeighborsClassifier(n_neighbors=8)  # optimal i is 65
classifier.fit(points,labels)
#test classifier
guess = classifier.predict(x_test)
print(classifier.score(x_test,y_test))
pred_color = ["blue" if x == 'm' else "red" for x in guess]

'''test case'''
d = {'age': [0.3], 'height': [0.6], 'selfish_index':[0.7], 'education_code':[0.6],'o
test = pd.DataFrame(data=d)
pred2 = classifier.predict(test)
print(pred2)

#plot what descriptors are well segregated
plt.scatter(x_test["income_code"],y_test, color=pred_color, s=70, alpha=0.1)
plt.xlabel('income_code')
plt.ylabel('sex')
plt.legend('predicted Male','predicted Female')
plt.show()

plt.scatter(x_test["education_code"],y_test, color=pred_color, s=70, alpha=0.1)
plt.xlabel('education_code')
plt.ylabel('sex')
plt.legend('predicted Male','predicted Female')
plt.show()

plt.scatter(x_test["height"],y_test, color=pred_color, s=70, alpha=0.1)
plt.xlabel('height')
plt.ylabel('sex')
plt.legend('predicted Male','predicted Female')
plt.show()

plt.scatter(x_test["orientation_code"],y_test, color=pred_color, s=70, alpha=0.1)
plt.xlabel('orientation_code')
plt.ylabel('sex')
plt.legend('predicted Male','predicted Female')
plt.show()
```

** It would be better to convert axis labels back to strings to make classifier more user friendly*

K-means- unsupervised clustering

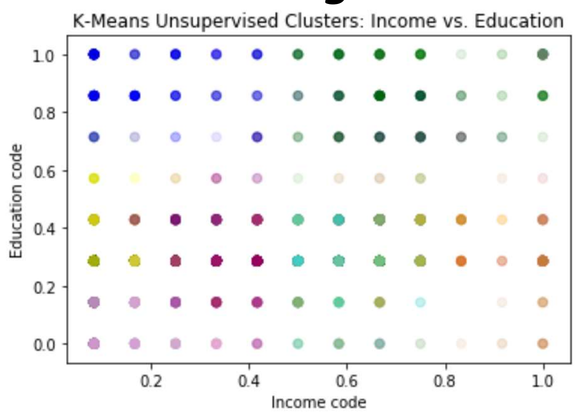
- *Inertia is the distance from each sample to the centroid of its cluster.*
- *Want to pick the lowest k number of clusters with the lowest inertia*
- *In this case it is ~ 10-20 clusters*

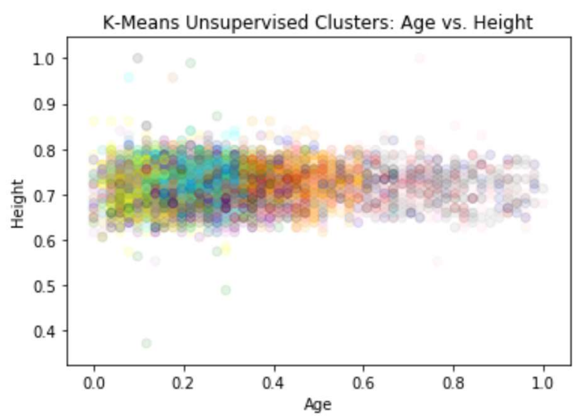



```
#unsupervised method (kmeans (or try neural network- random forest))
from sklearn.cluster import KMeans

ilist= []
inert= []

for i in range(1,100, 5):
    model = KMeans(n_clusters=i, random_state=1)
    model.fit(x_train)
    new_labels = model.predict(x_test)
    inertia = model.inertia_
    inert.append(inertia)
    ilist.append(i)

plt.plot(ilist,inert)
plt.xlabel("number of clusters")
plt.ylabel("inertia")
plt.show()
```

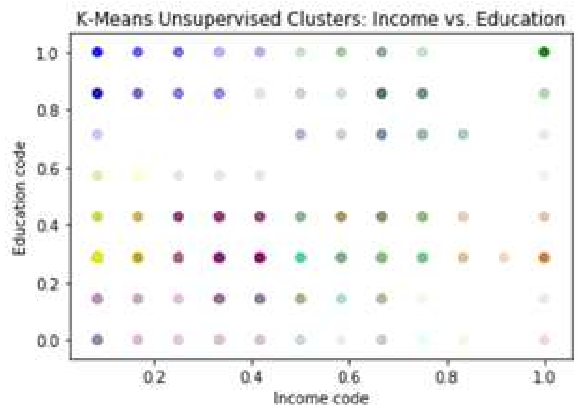
K-means- unsupervised clustering with 15 clusters separates out the dataset

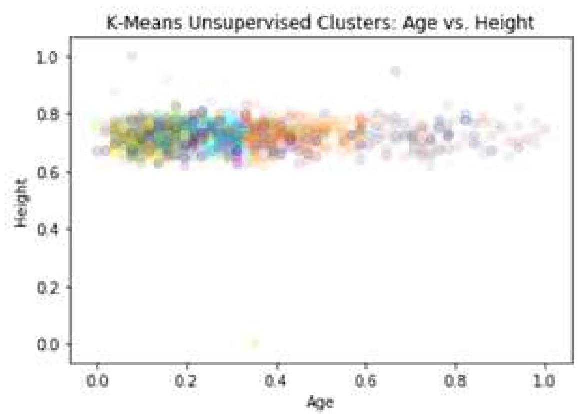
Training Set





Test Set





- 15 complex demographic groups are nicely separated with k-means unsupervised method

- To find good dating matches in the dataset, might want to identify people within the same k-means cluster.
  - Separate male/ female indices
  - Match up straight, gay and bisexual couples
  - Further segment by age

Conclusions

- The Okcupid dataset is composed largely of Millennials and GenX age individuals
  - Almost all baby boomers in the dating pool are straight
- There are more men in the dating pool than women
  - Men tend to be taller
  - Bisexual population is almost all women
  - Age differences present between men/women in the low-mid income group
- Age is a weak predictor of income
- Income can be predicted with higher accuracy if you add in additional parameters
- Sex can be predicted with 83% accuracy with K-Nearest Neighbor KNN classification
- K-Means can be used to partition people in dataset in an unsupervised way, based on numerous features
  - Matching up individuals with each of these groups (maybe using KNN <u>within</u> each group) would be a good way to suggest possible matches for individuals.