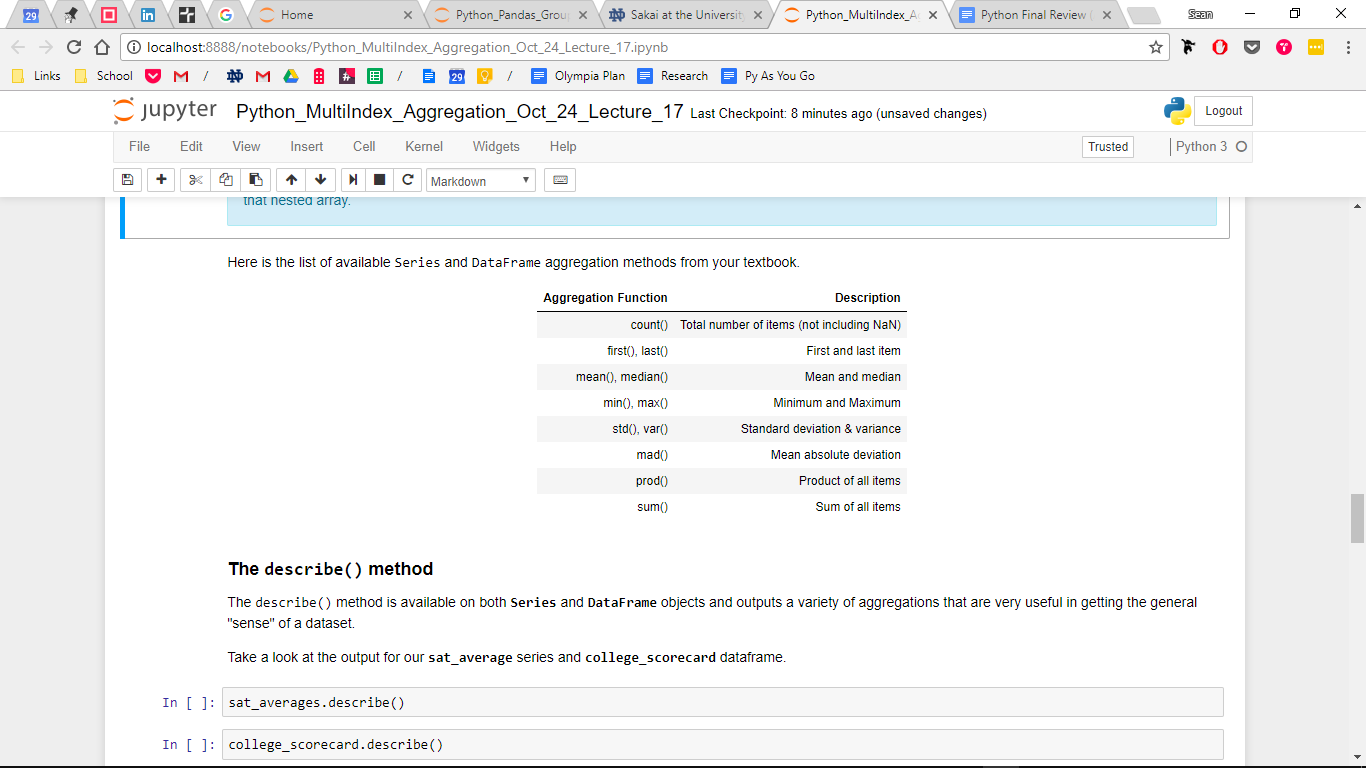
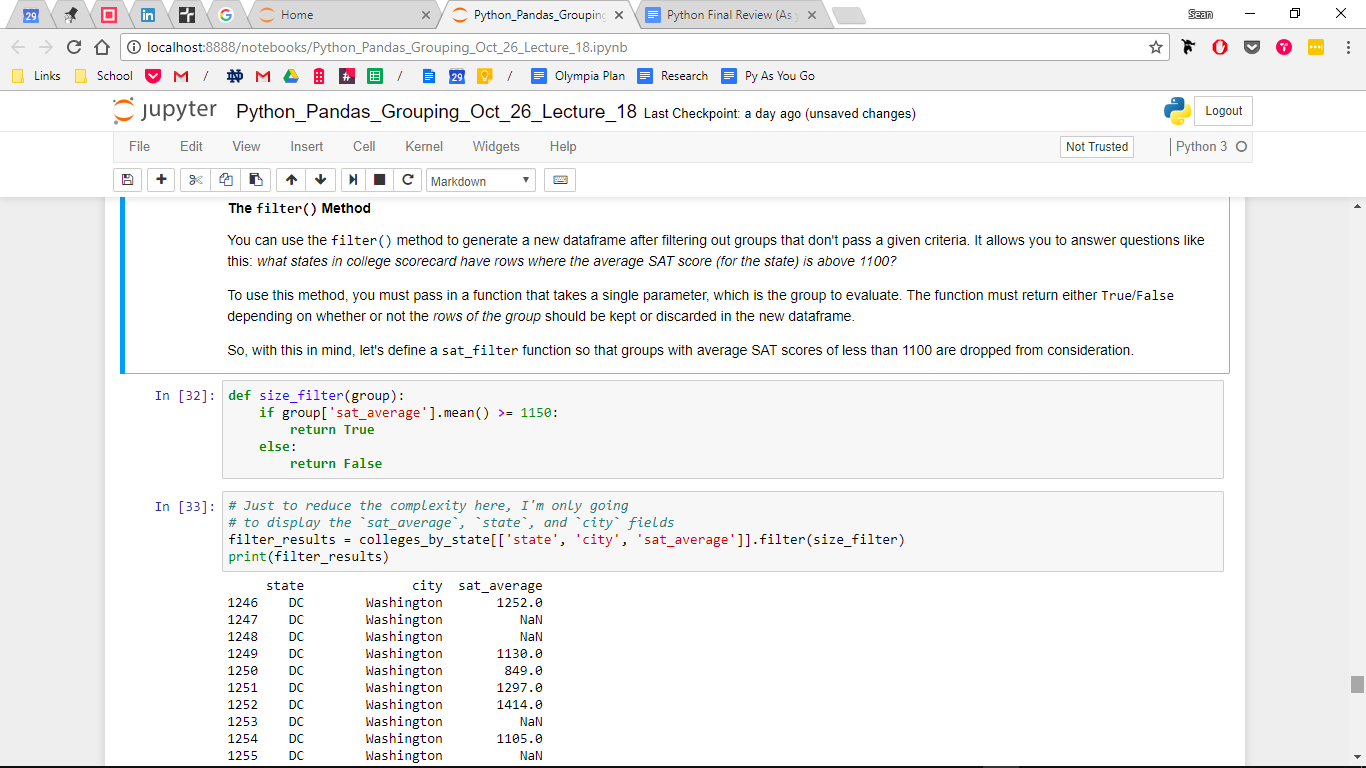
dOct 24/26 - Multiindexing, grouping, aggregate

* Multi/hierarchical index
  + Set one or multiple index: *df\_ind = df.set\_index(['regiment', 'company'])*
  + Ufunc on chosen index, unique values only: *df\_ind.mean(level = 'regiment')*
    - Also on both indices: *df\_ind.mean(level=['regiment','company'])*
* .describe
  + **By default, only numerical values**
  + Include the object datatype columns: *data.describe(include=[np.object])*
  + Exclude numeric datatypes: *data.describe(exclude=[np.number])*
  + *(include='all')* is also an option
  + Might need: *college\_scorecard.dtypes* to see what to inc/exclude
* Grouping
  + To group: *flights\_by\_airline\_month = flights.groupby(['AIRLINE', 'MONTH'])*
    - *flights.groups* to display
    - Use ([ for single and multiple columns
  + To group flights dataset by airline, then find mean distance traveled
    - *flights\_by\_airline\_month = flights.groupby(['AIRLINE', 'MONTH'])*
    - *flights\_by\_airline[['DISTANCE']].mean()*
  + Find mean for mult columns for each airline
    - *flights\_by\_airline[['DISTANCE', 'TAXI\_IN', 'TAXI\_OUT']].mean()*
  + Select data for only one index(airline in this case) and display median for 3 columns by month
    - *UA\_flights = flights[flights['AIRLINE'] == 'UA']*
    - *UA\_flights\_by\_month = UA\_flights.groupby(['MONTH'])*
    - *UA\_flights\_by\_month['DISTANCE','TAXI\_IN','TAXI\_OUT'].median()*
  + Display data for only one index: *summary\_by\_airline\_month.loc[‘United Airlines’]*
  + **Distinguish these**
    - Compute the median for the entire DataFrameGroupBy object and then select 'DISTANCE' column: *flights\_by\_airline.median()[['DISTANCE']]*
    - Select the 'DISTANCE' Column and then compute the median: *flights\_by\_airline[['DISTANCE']].median()*
* Aggregate
  + To count via aggregate: *flights\_by\_airline.aggregate('count')*
  + Various aggregates for each column of dataset:
    - Produce per state and city, minimum and maximum for the sat\_average column and average for the full\_time\_retention\_rate\_4\_year column.
    - *college\_score\_by\_statecity = college\_scorecard.groupby(['state', 'city'])*
    - *college\_score\_by\_statecity.aggregate(*

*{'sat\_average': [np.min, np.max],*

*'full\_time\_retention\_rate\_4\_year': np.mean})*

* + To apply friendlier labels to columns, use rename:
    - *flights\_by\_airline['DISTANCE'].aggregate([np.mean, np.min, np.max]).rename(columns={'mean': 'Avg. Distance', 'amin': 'Shortest Distance', 'amax': 'Longest Distance'})*
  + Continuous example:
    - Different Ufuncs for each column:
      * *var = flights\_by\_airline\_month.aggregate({'DISTANCE': [np.min, np.max], 'TAXI\_IN': np.mean})*
    - Renaming the column names when you have Hierarchy in column names
      * *var.columns.set\_levels(['Min Dist', 'Max Dist', 'Avg Taxi'], level = 1, inplace= True)*
  + **
* Advanced topics - **not on test**
  + Filter
    - What states in college scorecard have rows where the average SAT score (for the state) is above 1100?
      * **
    - Doesn’t group by state - just creates new df
  + Transform
    - **Dont really get, doesn’t seem needed - apply works just as well as i understand?**
  + Apply
    - Versatile
    - Count number of time a city shows up in state data using apply:

*def city\_counts(dataframe):*

*return dataframe['city'].value\_counts()*

*colleges\_by\_state.apply(city\_counts)*

* + - Create new column that centers (subtracts mean) another column’s data

*def center\_default\_rate(dataframe):*

*dataframe['center\_year\_1\_default\_rate'] = (*

*dataframe['year\_1\_default\_rate'] - dataframe['year\_1\_default\_rate'].mean())*

*return dataframe*

*college\_loan\_defaults\_by\_state.apply(center\_default\_rate)*

* Great example is airline alliances from Halloween notes

**October 31**

Matplotlib

* Starting off
  + *import matplotlib as mpl*
  + *import matplotlib.pyplot as plt*
  + Pick a style: *plt.style.use('seaborn')*
  + *%matplotlib inline*
  + Create figure and axes: *figure,* ***axes*** *= plt.subplots()*
  + X and Y axes, in order
    - ***axes****.plot(nd\_football\_roster.index, nd\_football\_roster['Height'])*
      * Note: the bolded bits in the above 3 lines have to match
  + More options you can add to end of above argument inside parentheses:
    - Alpha controls transparency: *alpha=.25*
    - *color='CadetBlue'* or *color='#f4429e'* or *color=(1.0, 0, 0)*
    - Grayscale: *color='.1'*
    - Linestyle: *linestyle='dotted'* or *linestyle='-'*
    - *linewidth=4.5*
    - *marker='\*', markersize=20* (can also do D for diamond or o for circles)
    - *label='Squad A'* this label will show up in the legend
  + Options that come in a separate line
    - *axes.set\_ylim(280, 340)* and *axes.set\_xlim(70, 90)*
    - *axes.set\_title(label='2017 ND Football Player Heights', loc='left', color='gold', alpha=.4)*
      * Loc indicates location. Can have multiple titles
    - *axes.set\_ylabel('Height(in)', color='red')* and *axes.set\_xlabel('Jersey #', alpha=.7)*

**November 2nd**

* Plot Legends
  + Initialize: *axes.legend()*
  + Inside the parentheses:
    - *loc='lower right'* for location
    - title='Legend'
    - *frameon=True, edgecolor='black', facecolor='gold'* options for a box w/ fill
    - *ncol=2* number of columns
    - *fontsize=14*
* Scatter Plots
  + Initialize:
    - figure, axes = plt.subplots()
    - axes.scatter(x variable, y variable)
      * Add other stuff to comma in:
      * Can assign color map based on a specific variable, whether or not it’s actually your y, with:
        + *c=data[‘column’]*
        + *cmap='coolwarm'*
      * Can also just assign gradient w colormap: *cmap='coolwarm'*
    - Colorbar
      * First, store entire axes.scatter into scatter\_image variable
      * *figure.colorbar(*

*Scatter\_image,*

*label='Temperature')*

* Histogram
  + One dimensional; just pass a variable:
    - *figure, axes = plt.subplots()*
    - *axes.hist(nd\_football\_roster['Height'])*
      * Add more bins inside parentheses: *bins=15*
      * Set range to change xlim and # of bins at the same time: *range=(.5, 2.5)*
    - Sample code
      * *figure, axes = plt.subplots()*
      * *image = axes.hist(*
      * *flights['DISTANCE'],*
      * *bins = 15,*
      * *range = (0,3000),*
      * *color='CadetBlue')*

Nov 7th

* Two-dimensional histograms
  + Create
    - *figure, axes = plt.subplots()*

*image = axes.hist2d(*

*nd\_football\_roster['Height'],*

*nd\_football\_roster['Weight']*

*cmap='coolwarm',*

*cmin=1)*

* Add colorbar
  + *figure.colorbar(results[3], label='Count of Occurrence')*
  + For histograms, have to pass 3rd index, just cause
* Get rid of clutter by setting *cmin=1* inside same parentheses after setting dataframes for histogram; this changes the minimum amount of counts for a data point to be visually encoded
* Bc 2 dimensions, range is complicated here:
  + *Range = [(a,b) , (a,b)*
* Interactive Plots
  + NEED THIS CODE
    - *%matplotlib notebook*

*import matplotlib as mpl*

*import matplotlib.pyplot as plt*

* Subplots
  + Create 1x2 subplot grid:
    - *figure, axes = plt.subplots(1, 2)*
      * Can add in *sharey=True* to have same y axis
    - *print(type(axes), axes, sep='\n')*
  + Specify which subplot to work with when inserting a DF
    - *axes[0].scatter(*
    - *nd\_football\_roster.index, nd\_football\_roster['Height'])*
* Adjust Plot Size
  + *figure.set\_size\_inches(10, 4)* beneath the rest of your code
  + Can also add *figsize=(12,5)* argument to *figure, axes = plt.subplots* parentheses
* Add Text
  + *axes.text(175, 15,'WAY TOO COLD',*
  + *size=20,* Adjust the size
  + *alpha=.8,* Adjust opacity
  + *color='red',* Change the color
  + *rotation=5* Add a rotation *)*
* Seaborn
  + Starting off
    - *import seaborn as sns*
    - Can now add *normed=True* to axes.hist parentheses, for example, to convert to probability scale
  + Plot distribution (hist + curve) of data
    - *sns.distplot(seattle\_weather\_2015\_2016['low\_temp'], ax= axes)*
  + Pairplot
    - USEFUL FOR STARTING WORK W DATA TO UNDERSTAND WHAT’S GOING ON
    - *sns.pairplot(seattle\_weather\_2015\_2016)*
  + Jointplot
    - Shows together and individual graphs
    - *sns.jointplot("low\_temp", "avgwindspeed", data=seattle\_weather\_2015\_2016)*

Nov 9th

* Side trick: To get number of flights each airline has in the dataset: *flights\_by\_airline.size()*
* Bar Plots
  + Bar Plots in Matplotlib (have to do certain way - it doesn’t know how to handle discrete, string datatypes - aka airlines as X axis)
    - *figure, axes = plt.subplots()*
    - *axes.bar(range(len(num\_flights\_airline.index)), num\_flights\_airline)*
    - Set ticks at every number: *axes.set\_xticks( range(len(num\_flights\_airline.index)))*
    - Rename ticks to airlines: *axes.set\_xticklabels(num\_flights\_airline.index)*
  + Easier bar plots
    - *figure, axes = plt.subplots()*
    - *df.plot(ax=axes, kind='bar', color='orange')*
  + Plot 3 bars, in a particular “sequence”
    - *width = 0.2*
    - *axes.bar(num\_flights\_month\_airline.loc['AA'].index - width, num\_flights\_month\_airline['AA'],*
    - *width=width,*
    - *label = "American")*
    - Repeat two more, one just index and one index+width
* Web Scraping
  + HTML comes in as a list: *nba\_data\_list = pd.read\_html("https://www.basketball-reference.com/draft/NBA\_2017.html")*
  + Access df table by pulling 0th element: *nba\_df = nba\_data\_list[0]*
  + Cleaning
    - Rename columns: *nba\_df.columns = ['Rk', 'Pk',*...
    - Drop unnecessary rows: *nba\_df.drop([30,31], axis=0, inplace= True)*
  + **MORE FROM THIS CLASS THAT YOU MISSED, INCLUDING *A P I ’S***

Nov 16th

* General notes
  + Don’t forget df.**loc**[‘AA’] to call the explicit index
  + Set column as permanent index: *goog.set\_index('Date', inplace=True)*
  + .nunique() gets number of unique elements
  + New df with certain columns: *goog\_new = goog****[[****'Open','High','Low','Close']]*
* Date and time
  + Start
    - *from datetime import datetime, date, time*
    - *from datetime import timedelta*
    - *now = datetime.now()*
    - *print(now)*
  + Assign date: *soon = datetime(2017,day=20, month=11)*
  + Add 10 seconds: *now + timedelta(seconds=600)*
  + *print(now.strftime('%a, %d %b %Y %H:%M:%S'))* leads to:
    - Thu, 16 Nov 2017 15:38:46
  + Convert str to datetime: *(datetime.strptime('2017-12-4', '%Y-%m-%d')*
  + Dateutil Parser
    - *from dateutil.parser import parse*
    - *print(parse('April 4th, 2017 at 11:30am'))*
* Pandas time series
  + Convert to datetime in pandas: *date = pd.to\_datetime("4th July, 2017")*
  + Pull out day only: *date.strftime('%A')*
  + Add one day: *date + pd.to\_timedelta(1, 'D')*
    - List of other letters to use

|  |  |
| --- | --- |
| **Code** | **Description** |
| D | Calendar Day |
| W | Week |
| M | Month |
| Y | Year |
| H | Hour |
| T | Minute |
| S | Seconds |
| B | Business day |
| BM | Business Month end |
| BQ | Business Quarter end |
| BA | Business Year end |
| BH | Business Hours |

* + Converting the date and replacing it with the same column
    - *goog['Date'] = pd.to\_datetime(goog['Date'])*
  + Latest date in column: *goog['Date'].max()*
  + Select range of dates in index, even if in a different format!
    - *goog.loc['Nov 7th 2015':'Nov 16th 2015']*
      * Add :4 at the end to get every 4th day
  + Select entire month or year: *goog.loc['Dec 2015']*
  + Average stock price in month of january: *goog.loc['Jan 2014']['Close'].mean()*
* Plotting timeseries data
  + Start
    - *figure, axes = plt.subplots()*
    - *goog\_new.plot(ax=axes)*
  + Plot only one month: *goog\_new.loc['Nov 2008'].plot(ax = axes)*
  + Plot only one column: *goog\_new['Close'].plot(ax = axes)*
  + Plot time slice of one column
    - *goog\_new.loc['2007 Dec':'2009 May']['Close'].plot(ax = axes)*
  + Plot slice of 2 columns:
    - *goog\_new.loc['2007 Dec':'2009 May'][['High', 'Low']].plot(ax = axes)*
* Timeseries operations
  + Similar to groupby - think of group by month if you sample by month
  + resample() will compute groupby and get mean for each segment for {d,w,m,y}, whichever you select
    - HAVE TO DO .mean() or .median() at the end!!!
  + asfreq() will get you value at the end of that {d,w,m,y}
  + Examples:
    - Avg google price for entire years: *goog.resample('A').mean()*
    - Exact price on last business day of every year: *goog.asfreq('BA')*
    - *goog.resample('5D').mean()* for every 5 days
  + Now can plot w df.resample(‘BA’).mean().plot()
* Rolling window
  + Rolling avg of last 7 observations (will get NaN for the first 6): *goog.rolling(7).mean()*
  + *flights[‘DISTANCE’].resample(****‘D’****).mean().rolling(5).mean().plot()* will do rolling for whatever time block you choose
    - First avg for each day, then avg for each 5 days. Need the first mean to make it one value

Nov 21 - Machine Learning

* Other random thoughts
  + These class ppt notes were really interesting. Machine learning as attempting to find decision boundaries and reduce dimensionalities to simpler structure; like segmenting in marketing.
  + Machine learning + Google trends (“is my model correct?”) would be dope
  + The concept of pairplot, where we just say “throw up a bunch of tests and see if we can spot trends” - **what are more advanced ways of doing this?**
* Scikit Learn
  + Start
    - *import sklearn as skl*
    - *from sklearn.datasets import \_\_\_name of file\_\_\_* if necessary
  + Pairplot with one variable determining hue: *sns.pairplot(bp\_data, hue='high\_bp')*
  + Split input features and outcome variable (create x and y from larger data)
    - *bp\_data\_X = bp\_data.drop('high\_bp', 1)*
    - *bp\_data\_Y = bp\_data['high\_bp']*
  + Split data into train and test
    - Random state will randomly split data - can put any number
    - Train size = what fraction to use for training
    - *from sklearn.model\_selection import train\_test\_split*
    - *bp\_train\_X, bp\_test\_X, bp\_train\_Y, bp\_test\_Y = train\_test\_split(bp\_data\_X, bp\_data\_Y, random\_state=42, train\_size = 0.75)*
  + Basic Classifier (focus of later classes)
    - *from sklearn.naive\_bayes import GaussianNB*
    - *model = GaussianNB()*
    - *model.fit(bp\_train\_X, bp\_train\_Y)*
  + Predict on test data
    - *bp\_predict\_Y = model.predict(bp\_test\_X)*
    - *import sklearn.metrics as sklmetrics*
    - *sklmetrics.accuracy\_score(bp\_test\_Y, bp\_predict\_Y)*
    - The number that comes as a result here is the likelihood that the model can predict successfully, from x to y. BUT IT CHANGES BASED ON HOW DATA WAS SPLIT!
  + Plot confusion matrix (true pos, false neg, false pos, true neg in order)
    - *conf\_mat = sklmetrics.confusion\_matrix(bp\_test\_Y, bp\_predict\_Y, labels =[0,1])*
    - Plot with seaborn
      * *sns.heatmap(conf\_mat, square=True, annot=True, cbar = False)*
      * *plt.xlabel("Predicted Value")*
      * *plt.ylabel("True Value")*
  + Dimensionality Reduction and Iris Clustering example with drawn numbers - DOPE

Nov 28 - Classification w SciKitLearn

* Getting started:
  + Assign X and Y:
    - *got\_data\_X = got\_data.drop('dead',1)*
    - *got\_data\_Y = got\_data['dead']*
  + Split X and Y into train and test:
    - *from sklearn.model\_selection import train\_test\_split*
    - *got\_train\_X, got\_test\_X, got\_train\_Y, got\_test\_Y = train\_test\_split(got\_data\_X, got\_data\_Y, random\_state=42, train\_size = 0.7)* “use 70% of the data for training”
    - Check length: *print(len(got\_data\_X), len(got\_train\_X), len(got\_test\_X))*
* Logistic Regression
  + Start
    - *from sklearn.linear\_model import LogisticRegression*
    - *log\_regression\_model = LogisticRegression(class\_weight=’balanced’)*
    - *log\_regression\_model.fit(got\_train\_X, got\_train\_Y)*
  + Predict on test data
    - *got\_predict\_Y = log\_regression\_model.predict(got\_test\_X)*
    - *import sklearn.metrics as sklmetrics*
    - *sklmetrics.accuracy\_score(got\_test\_Y, got\_predict\_Y)*
  + Confusion Matrix
    - *conf\_mat = sklmetrics.confusion\_matrix(got\_test\_Y, got\_predict\_Y, labels =[0,1])* (can type conf\_mat to see a basic graphical version for info)
    - *sns.heatmap(conf\_mat, square=True, annot=True, cbar = False, xticklabels = ['Alive','Dead'], yticklabels = ['Alive','Dead'])*
    - *plt.xlabel("Predicted Value")*
    - *plt.ylabel("True Value")*
  + We also looked at which components were most significant in contributing to ultimate binary prediction - maybe use in group proj
* Decision Tree
  + *from sklearn.tree import DecisionTreeClassifier*
  + *dec\_tree\_model = DecisionTreeClassifier()*
  + *dec\_tree\_model.fit(got\_train\_X, got\_train\_Y)*
  + *got\_predict\_Y = dec\_tree\_model.predict(got\_test\_X)*
  + *print(sklmetrics.accuracy\_score(got\_test\_Y, got\_predict\_Y))*
  + Conf Matrix
    - *conf\_mat = sklmetrics.confusion\_matrix(got\_test\_Y, got\_predict\_Y, labels =[0,1])*
    - *print(conf\_mat)*
    - *sns.heatmap(conf\_mat, square=True, annot=True, cbar = False, xticklabels = ['Alive','Dead'], yticklabels = ['Alive','Dead'])*
    - *plt.xlabel("Predicted Value")*
    - *plt.ylabel("True Value")*
* Activity
  + Drop objects (categorical) datatypes
    - *Bank\_data.dtypes* to check
    - *bank\_data.drop(['marital','education','contact'], axis=1, inplace=True)*
    - *bank\_data.head()*
  + When class\_weight is extremely unbalanced (like with cold calling, vast majority of calls are fails - this will make it look like your model is “accurate”), use the blue text above after LogisticRegression
    - To check if inbalanced, *print(bank\_data.success.value\_counts())*
      * Success was the column here

11/30 - **I missed class**

* GetDummies technique - handle categorical data by splitting into its options
  + **MUST DELETE ONE CATEGORY**
* Look for categories with *data[‘column’].unique()*
* Create
  + *bank\_data\_with\_dummies = pd.get\_dummies(bank\_data)*
  + *bank\_data\_with\_dummies.head()*
* Remove additional variables (just removing one)(**should automatically delete original, check)**
  + *bank\_data\_with\_dummies.drop(['marital\_divorced','education\_unknown','contact\_unknown'], axis = 1, inplace=True)*
* Parameters of classification models
  + Logistic Regression: Avoid overfit w variable selection (regularization - subset of input into regression, less variables)
    - **Penalty** (common values of 'l1','l2')
    - **C** (weight to regularization, common values .0001 to 10000 in factors of 10)
  + Decision Tree
    - **max\_depth** (# of levels)
    - **Max\_features** (how many features to consider when splitting a node. Common are ['auto','log2', None])
* General steps:
  + Split input x and outcome y
  + Split into train and test
  + Create model for logistic (cross-validate; test each combo of penalty and C)
    - *# penalty = 'l1', C = 0.1 A = 0.80545320560058953*
    - *# penalty = 'l1', C = 1 A = 0.80397936624907884*
    - *# penalty = 'l1', C = 10 A = 0.80103*
    - *# penalty = 'l2', C = 0.1 A = 0.80250552689756816*
    - *# penalty = 'l2', C = 1 A = 0.80250552689756816*
    - *# penalty = 'l2', C = 10 A= 0.80103168754605747*
    - *#Creating the model*
    - *log\_reg\_model\_parameter = LogisticRegression(class\_weight='balanced', penalty = 'l2', C = .1)*
    - *#Training the model*
    - *log\_reg\_model\_parameter.fit(bank\_dum\_train\_X, bank\_dum\_train\_Y)*
    - *# Predict on the test Data*
    - *bank\_dum\_predict\_Y = log\_reg\_model\_parameter.predict(bank\_dum\_test\_X)*
    - *# Test the accuracy of the model*
    - *sklmetrics.accuracy\_score(bank\_dum\_test\_Y, bank\_dum\_predict\_Y)*
    - At the end, keep the values that worked the best
  + Do the same for decision tree
  + However, we can’t use test data to choose parameters - have to use scikitlearn
* GridSearchCV()
  + Searches thru all parameter possibilities and selects best
  + Gridsearch parameters
    - **estimator**: The classifier you want to learn the parameters, LogisticRegression, DecisionTreeClassifier, etc.
    - **param\_grid**: Dictionary (dict) of parameters and their values to be searched over.
    - **cv**: How many times you want it to run. Usually 3 for smaller data and 10 for large data.
    - **n\_jobs**: Usually you specify this as 1. You can parallelize the process of this search by specifying a value more than 1. **Do not have the n\_jobs set to more than 3**, for the first time users. Especially, on a laptop or lab machine or on Vocareum, you will end up stalling the machine and it has to be rebooted. For more experienced students, in the class, check the number of processing cores of the machine before you increase the number.
  + Set-up
    - *from sklearn.model\_selection import GridSearchCV*
    - Cross-validation w gridsearch
      * *# The model you want to set the parameters for*
      * *model = DecisionTreeClassifier/LogisticRegression(class\_weight='balanced')*
      * *# The parameters to search over for the model*
      * *params = {'max\_depth':[2,3,4], {'penalty':['l1','l2'],*
      * *'max\_features':['auto','log2',None]} 'C':[0.01, 0.1, 1, 10, 100]}*
      * *# Prepare the GridSearch for cross validation*
      * *grid\_search\_dec\_tree = GridSearchCV(model, # Note the model is DecisionTreeClassifier as stated above*
      * *param\_grid=params, # The parameters to search over.*
      * *cv=3, # How many hold out sets to use*
      * *n\_jobs = 1 # Number of parallel processes to run.*
      * *)*
      * *# Do the cross validation on the training data*
      * *grid\_search\_dec\_tree.fit(bank\_dum\_train\_X, bank\_dum\_train\_Y)*
      * *# Select the best model*
      * *best\_dec\_tree\_cv/best\_log\_reg\_cv = grid\_search\_dec\_tree.best\_estimator\_*
      * *# Print the best parameter combination*
      * *print(grid\_search\_dec\_tree.best\_params\_)*
    - Test
      * *# Finally test the performance of the best model on the test data*
      * *bank\_dum\_pred\_Y = best\_dec\_tree\_cv.predict(bank\_dum\_test\_X)*
      * *#Print the accuracy*
      * *print(sklmetrics.accuracy\_score(bank\_dum\_test\_Y, bank\_dum\_pred\_Y))*
      * *conf\_mat = sklmetrics.confusion\_matrix(bank\_dum\_test\_Y, bank\_dum\_pred\_Y)*
      * *print(conf\_mat)*
      * *# Confusion matrix*
      * *sns.heatmap(conf\_mat, fmt='d',square=True, annot=True, cbar = False, xticklabels = ['Failure','Success'],*
      * *yticklabels = ['Failure','Success'])*
      * *plt.xlabel("Predicted Value")*
      * *plt.ylabel("True Value")*
    - Function for feature importance
      * *def plot\_feature\_importance(model, Xnames, cls\_nm = None):*
      * *# Measuring important features*
      * *imp\_features = pd.DataFrame(np.column\_stack((Xnames, model.feature\_importances\_)), columns = ['feature', 'importance'])*
      * *imp\_features[['importance']] = imp\_features[['importance']].astype(float)*
      * *imp\_features[['abs\_importance']] = imp\_features[['importance']].abs()*
      * *# Sort the features based on absolute value of importance*
      * *imp\_features = imp\_features.sort\_values(by = ['abs\_importance'], ascending = [1])*
      * *# Plot the feature importances of the forest*
      * *plt.figure(figsize=(10,6))*
      * *plt.title(cls\_nm + " - Feature Importance")*
      * *plt.barh(range(imp\_features.shape[0]), imp\_features['importance'],*
      * *color="b", align="center")*
      * *plt.yticks(range(imp\_features.shape[0]), imp\_features['feature'], )*
      * *plt.ylim([-1, imp\_features.shape[0]])*
      * *plt.xlabel('Importance')*
      * *plt.ylabel('Feature')*
      * *plt.tight\_layout()*
      * *plt.savefig(cls\_nm + "\_feature\_imp.png", bbox\_inches='tight')*
      * *plt.show()*
      * **Then, after:** *plot\_feature\_importance(best\_dec\_tree\_cv, bank\_dum\_train\_X.columns, cls\_nm='Best CV Decision Tree')*

12/5 - **I effectively missed class**

* Clustering
  + *from sklearn.cluster import KMeans*
  + *model = KMeans(n\_clusters=3)***n is number of clusters**
  + *model.fit(cluster\_data)*
  + *# Assigning the cluster numbers to the each data point in DF*
  + *cluster\_data['cluster'] = model.predict(cluster\_data)*
  + *cluster\_data.head()*
* Plot clusters w color
  + *fig, ax = plt.subplots(figsize = (10,8))*
  + *ax.scatter(cluster\_data['X1'], cluster\_data['X2'], c=cluster\_data['cluster'], cmap = 'viridis')*
  + *ax.set\_xlabel("X1")*
  + *ax.set\_ylabel("X2")*
  + *ax.set\_title("Clusters identified by the KMeans Algorithm")*

**Final exam review**

* All imports:
  + Data Manipulation
    - import pandas as pd
    - import numpy as np
  + Visualization
    - import matplotlib as mpl
    - import matplotlib.pyplot as plt
    - import seaborn as sns
    - %matplotlib inline
  + Machine Learning
    - import sklearn as skl
    - from sklearn.linear\_model import LogisticRegression
    - from sklearn.tree import DecisionTreeClassifier
    - from sklearn.model\_selection import train\_test\_split
    - import sklearn.metrics as sklmetrics
    - from sklearn.model\_selection import GridSearchCV
* How many users have rated the movie 'Back to the Future' lower than 3?
  + *movies\_ratings[(movies\_ratings['title'] == 'Back to the Future') & (movies\_ratings['rating'] < 3)]['userId'].nunique()*
* What are the top 10 movies which have the highest rating?
  + *movie\_avg\_rat = movies\_ratings.groupby(['movieId','title'])[['rating']].mean()*
  + *movie\_avg\_rat.sort\_values('rating', ascending=False).iloc[:10]*
* Create new column of average ratings for that movie, given long list of all movie ratings
  + *avg = pd.merge(movies\_ratings, movie\_avg\_rat, left\_on=['movieId','title'], right\_index=True, suffixes=('', '\_avg'))*
* Plotting with a groupby
  + *movies\_by\_year = movies\_ratings.groupby('year')*
  + *movies\_by\_year.size().plot(ax = axes)*
  + Ends up with year on x and # of movies in list (aka number of ratings) in y
  + Similar:
    - *figure, axes = plt.subplots()*
    - *flights.loc['july 4 2015']****[['****TAXI\_IN', 'TAXI\_OUT'****]]****.plot(ax=axes)*
  + Similar again
    - *goog\_new = goog[['Open','High','Low','Close']]*
    - *figure, axes = plt.subplots()*
    - *goog\_new.plot(ax=axes)*
    - plot slices of high and low stock prices: *goog\_new.loc['2007 Dec':'2009 May'][['High', 'Low']].plot(ax = axes)*