Lesson 11 – Python statistics in EDA

**Questons for Mentor:**

**Intro to EDA:**

* EDA is an essential step in data science method process
* Plotting a histogram
* Always label axes
* Seaborn is a data viz library
  + Can get style to be seaborn default using sns.set() function
* Avoid binning bias
  + Data can look different based on different binning
* Bee swarm plot
  + In seaborn library
  + Syntax: sns.swarmplot()
* ECDFs – Empirical cumulative distribution function
  + X value is quanitity we’re measuring
  + Y value is # of values with value smaller than corresponding x value
  + Can plot multiple ECDFs on same view
  + Function:
    - def ecdf(data):
    - """Compute ECDF for a one-dimensional array of measurements."""
    - # Number of data points: n
    - n = len(data)
    - # x-data for the ECDF: x
    - x = np.sort(data)
    - # y-data for the ECDF: y
    - y = np.arange(1, n+1) / n
    - return x, y
* Introduction to summary statistics: the sample mean and median
  + Mean has syntax: np.mean()
  + Median has syntax: np.median() – also not overly impacted by outliers like mean
* Percentiles, outliers and box plots
  + Percentiles Syntax: np.percentile(df[‘column’], [25, 50, 75]) – will calc 25th 50th and 75th percentiles
  + Box plots
    - Show percentiles with whiskers showing 1.5 IQR from 25th and 75th percentiles
    - Items outside whiskers can be outliers, but may not be erroneous
    - Syntax: sns.boxplot()
* Variance
  + Syntax: np.var(data)
  + It is square of mean of differences
  + Square root of variance is standard deviation
  + Standard dev uses more reasonable scale and is more realistic look at variability of data points
* Covariance
  + Measure of negative or positive correlation
  + Can compute with built in numpy functions
  + Indexing of covariance is very tricky
    - Multiple dimensions of indexes
* Pearson correlation coefficient
  + Comparision of variability due to codependence (covariance) and independent variability (std dev)
  + def pearson\_r(x, y):
  + """Compute Pearson correlation coefficient between two arrays."""
  + # Compute correlation matrix: corr\_mat
  + corr\_mat = np.corrcoef(x, y)
  + # Return entry [0,1]
  + return corr\_mat[0,1]
* Probabilistic reasoning
  + Statistical inference – using measured data to gather probabilistic conclusions
* Hacker Statistics
  + Figure out how to simulate with data
  + Simulate
  + Probability is the number of times you get what you want over the total number of simulations
  + Np.random.random() gets a random number
  + Np.random.seed() will give us a seed random number – same random number when we run same code
* PMF – Probability mass function
  + Np.random.binomial(n of trials, probability, size=\*)
* CDF – plotted before going from smallest value up to the largest
* Poisson processes and the Poisson distribution
  + Poisson process = timing of next event has nothing to do with previous timing
  + Poisson distribution = number r of arrivals of a Poisson process in a given time interval with an average rate of ?
  + Np.random.poisson() takes a random sample from poisson distribution
* Probability density functions
  + Continuous variables can take on any value, not just discrete values
  + Areas under PDF curve give probability
* Normal Distribution
  + Continuous variable that is symmetrical and has a single peak
  + Peak is mean
  + Np.random.normal(mean, std, size=\*)
  + Sometimes, things we think are normal distributions, really aren’t
  + Think of lightness of tails – it there are too many major outliers, normal distribution might not be right
* Exponential distribution
  + The waiting time between arrivals of a poisson process is exponencially distributed
  + Syntax: np.random.exponential(mean, size=\*)
* Linkedin Learning
  + P value – probability of null hypothesis happening
  + Uses for models
    - Reveal facts and trends about populations
    - Predict future behavior
    - Predict method allows you to repeat model function
  + Session walks through:
    - Fitting models
      * Step by step of least squares model
    - Goodness of fit
      * How to assess how “good” a model really is
      * Mean squared error of residuals
      * R^2 = explained variance / total variance
      * F statistic: explanatory power of fit parameters compared to random fit vectors (hypothetical explanatory variables)
      * Anova = analysis of variance
    - Cross validation
      * Cross validating across models and with shuffled data
    - Logistic regression
      * Standard linear model -> logistic regression -> probability
      * Maximum likelihood estimation
    - Bayesian inference
      * Using prior knowledge, new data to get to new conclusions
* Data Visualization in Python
  + Axes() creates coordinate axes within a figure with which subsequent plots are drawn
  + Axes() uses figure units, positional location
  + Subplot() creates grid of axes, multiple plots with different plots
  + Axis() command controls extents of axes,
    - allows you to control extent of x and y axis in same function
    - axis((xlim1, xlim2, ylim1, ylim2))
  + Plt.xlim() and plt.ylin() controls limits of x and y axis
  + Annotate() lets you add text to body of the chart and can include arrows
    - Syntax: plt.annotate(s, xy, xytext, arrowprops)
    - S is annotation label, xy is coordinates of arrow pointing, xytext is
  + Two dimensional histogram – plots joint variation of 2 variables instead of one
    - Plt.hist2d(x, y)
  + Seaborn – data viz package on top of matplotlib
    - Works best with dfs
  + Sns.lmplot() plots a regression (scatter with regression line)
  + Sns.regplot() is a more complicated regression plot
  + Sns.lmplot() is just a higher level interface to sns.regplot()
  + Hue argument allows you to add another level of detail (i.e. origin of cars with the weight and hp data)
  + Can split out in subplots row-wise or column-wise using the row= or column= argument within sns.lmplot()
  + Strip plots, swarm plots, violin plots
  + Strip plot is just a line of points on a plot for one variable
    - Syntax: stripplot()
    - Can add more detail by showing variable for different criteria (i.e. days)
  + Swarm plot is a strip plot, but same values are spread out horizontally so you can see the repeats
    - Orient=h will make the plot horizontal
  + Violin plot is similar to box plot but has rounded distribution showing density of distribution
  + Joint plot scatter with histograms to the side and top
  + Pair plot shows scatter of all variables against each other
  + Heat maps show a grid that have darker and lighter shades of color depending on the data
* Hypothesis testing in python
  + Optimal parameters
    - Parameter values that bring the model in closest agreement with the data
      * Calc mean and standard dev straight from data
  + Linear regression by least squares
    - Residual is the space between a data point and regression line
    - Find line where the sum of the squares of residuals is minimal
      * Least squares model
    - Np.polyfit(x, y, degree) performs least squares calc with polynomials
    - Dependent variable = response variable
    - Independent variable = explanatory variable
  + Importance of EDA
    - Anscombe’s quartet
      * 4 grids with almost exact slope, intercept etc that look very different
      * Look before you leap, the numbers don’t tell the whole story
  + Bootstrapping
    - Resampling can give us distro of summary statistics from sample for statistical inference
    - Can resample using Np.random.choice(list, size=n)
    - Bootstrap replicate is a single value of a statistic computed from a bootstrapped sample
  + Bootstrap confidence intervals
    - Normed=True means that the area under the bars will equal to 1 so as to portray the probability of each bin
    - Confidence interval = if we repeated measurements over and over again, p% of the observed values would lie within the p% confidence interval
      * Syntax: np.percentile(data, [2.5, 97.5])
    - def bootstrap\_replicate\_1d(data, func):
    - """Generate bootstrap replicate of 1D data."""
    - bs\_sample = np.random.choice(data, len(data))
    - return func(bs\_sample)
    - Standard error of the mean = standard dev of data / sqrt of # of data points
    - def draw\_bs\_reps(data, func, size=1):
    - """Draw bootstrap replicates."""
    - # Initialize array of replicates: bs\_replicates
    - bs\_replicates = np.empty(size)
    - # Generate replicates
    - for i in range(size):
    - bs\_replicates[i] = bootstrap\_replicate\_1d(data, func)
    - return bs\_replicates
    - Pulling bootstrap replicates for linear regression
    - def draw\_bs\_pairs\_linreg(x, y, size=1):
    - """Perform pairs bootstrap for linear regression."""
    - # Set up array of indices to sample from: inds
    - inds = np.arange(len(x))
    - # Initialize replicates: bs\_slope\_reps, bs\_intercept\_reps
    - bs\_slope\_reps = np.empty(size)
    - bs\_intercept\_reps = np.empty(size)
    - # Generate replicates
    - for i in range(size):
    - bs\_inds = np.random.choice(inds, size=len(inds))
    - bs\_x, bs\_y = x[bs\_inds], y[bs\_inds]
    - bs\_slope\_reps[i], bs\_intercept\_reps[i] = np.polyfit(bs\_x, bs\_y, 1)
    - return bs\_slope\_reps, bs\_intercept\_reps
  + Pairs bootstrap
    - Resample data in pairs (x, y)
    - Compute slope and intercept from resampled data
  + Hypothesis testing
    - Assessment of how reasonable the observed data are assuming a hypothesis is true
    - Hypothesis we’re testing – null hypothesis
    - Scrambling order of entries in array is permutation
    - Np.random.permutation(data) will let you get permuted data replicates
      * def permutation\_sample(data1, data2):
      * """Generate a permutation sample from two data sets."""
      * # Concatenate the data sets: data
      * data = np.concatenate((data1, data2))
      * # Permute the concatenated array: permuted\_data
      * permuted\_data = np.random.permutation(data)
      * # Split the permuted array into two: perm\_sample\_1, perm\_sample\_2
      * perm\_sample\_1 = permuted\_data[:len(data1)]
      * perm\_sample\_2 = permuted\_data[len(data1):]
      * return perm\_sample\_1, perm\_sample\_2
    - Generating permutation replicates
      * def draw\_perm\_reps(data\_1, data\_2, func, size=1):
      * """Generate multiple permutation replicates."""
      * # Initialize array of replicates: perm\_replicates
      * perm\_replicates = np.empty(size)
      * for i in range(size):
      * # Generate permutation sample
      * perm\_sample\_1, perm\_sample\_2 = permutation\_sample(data\_1, data\_2)
      * # Compute the test statistic
      * perm\_replicates[i] = func(perm\_sample\_1, perm\_sample\_2)
      * return perm\_replicates
  + Test statistics and p-values
    - Single number that can be computed from observed data and from data you simulate under null hypothesis
    - Serves as a basis of comparison between the two
    - Example would be difference in means from hypo to test (should be zero is null hypothesis is correct)
    - P-value – probability of obtaining a value of your test stat that is at least as extreme as what was observed, under the assumption the null hypothesis is true
      * Not the probability the null hypothesis is true
      * Example calc:
        + p = np.sum(perm\_replicates >= empirical\_diff\_means) / len(perm\_replicates)

in other words, sum of instances where the replicates were >= null hypothesis, divided by number of replicates

* + - * Example: Find fraction of replicates that are greater than the differences between the mean
      * Example: sum(bootstrapping test >= or <= observed val) / len(bootstrapping test)
    - Difference of Means function
      * def diff\_of\_means(data\_1, data\_2):
      * """Difference in means of two arrays."""
      * # The difference of means of data\_1, data\_2: diff
      * diff = np.mean(data\_1) - np.mean(data\_2)
      * return diff
  + Bootstrap hypothesis tests
    - Clearly state null hypothesis
    - Define test stat
    - Generate many sets of simulated data assuming null hypo is true
    - Compute test stat for each simulated data set
    - P-val is fraction of your simulated data sets for which the test stat is as least as extreme as for the real data
* A/B Testing
  + Used by orgs to see if a strategy causes positive change
  + Statistical significance does not mean practical significance
* Test of correlation
  + State hypothesis
  + Simulate data
  + Use pearson correlation coefficient as test stat
  + Compute p val as fraction of replicates that have pearson corr coef at least as large as observed
* Finch beaks and the need for statistics
  + EDA
  + Parameters estimates
  + Hypothesis testing