STAT 177, CLASS 7

Richard Waterman

July 2020

OBJECTIVES

OBJECTIVES

- Statistical comparisons
- Hypothesis testing
- Seaborn regression plots
    - Graphing a simple regression model
    - Regression over subgroups
    - Graphically checking model assumptions
    - Smoothing with lowess
- Fitting the regression with ‘statsmodels’
- Prediction from regression
- Multiple regression
    - One line fits all
    - Parallel lines
    - Interaction (different slopes)

STATISTICAL COMPARISONS AND A TASTE OF "INFERENCE"

STATISTICAL COMPARISONS

- Graphics are useful to give us a sense of what is going on in the data, but there is often ambiguity in conclusions.
- Formal statistical comparison procedures allow us to compare groups in terms of an outcome of interest.
- We will illustrate with the PRSM data set, which has data on loan performance for loans given to small merchants.
- We will perform a two-sample t-test on the difference in average FICO score between Original and Repeat Loan.Type.

GETTING THE DATA

- Start by setting up the libraries and reading in the data.

```
import os
import numpy as np
np.set_printoptions(precision=5, suppress=True)
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm # This is the statistical modeling library we will use.

os.chdir('C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\477s2020\\DataSets')
prsm_data = pd.read_csv("PRSM_data.csv", index_col=0)
prsm_data.columns
```

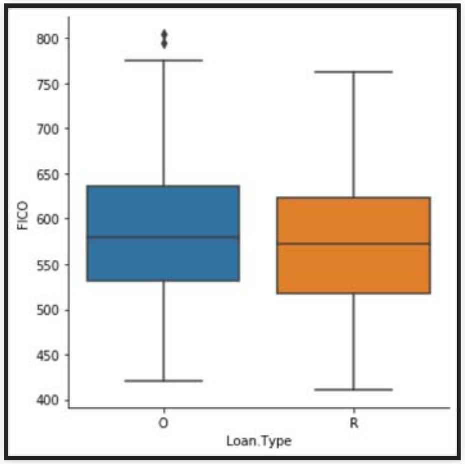
```
Index(['PRSM', 'Amt.Repaid.at.6.Months', 'Nominal.Loan.Amount',
       'Total.Amt.to.be.Repaid', 'Repayment.Percentage', 'Commission.Upfront',
       'Validated.Monthly.Batch', 'Historical.Monthly.Credit.Card.Receipts',
       'Months.of.History', 'Loan.Type', 'FICO', 'Years.In.Business',
       'Num.of.Credit.Lines', 'Satisfied.Accounts',
       'Num.of.Paid.off.Credit.Lines', 'Current.Delinquent.Credit.Lines',
       'Previous.Delinquent.Credit.Lines', 'Business.Entity.Type',
       'Years.in.File', 'Num.of.Legal.Items', 'Num.of.Trade.Lines',
       'Num.of.Derog.Legal.Item', 'Two.Digit.SIC.Code',
       'Two.Digit.SIC.Description', 'Population.in.Zip.Code',
       'Male.Population.in.Zip.Code', 'Female.Population.in.Zip.Code',
       'Average.House.Value.in.Zip.Code', 'Income.Per.Household.in.Zip....',
```

```
'Median.Age.in.Zip.Code', 'Time.Zone', 'Bus.Establishments.in.Zip.Code',
'Employment.in.Zip.Code', 'Annual.Payroll.in.Zip.Code',
```

GRAPH THE FICO SCORE OVER THE LEVELS OF LOAN.TYPE

- Loan type indicates whether the loan to the merchant was an original (“O”) or repeat (“R”) loan.
- FICO score is a measure of an individual’s creditworthiness.

```
sns.catplot(x='Loan.Type', y="FICO", kind="box", data=prsm_data); # The comparison boxplots
```



REMARKS

- It looks like the median FICO sore is higher for the Original loan types, but given that these were a sample from a population of loans, it is hard to make a definitive statement.
- There is inherent variability in FICO, due to the sampling process.
- We will focus on comparing *mean* FICO score across the two groups.

```
# Find the two means
prsm_data.groupby('Loan.Type')['FICO'].mean()
```

```
Loan.Type
O    582.847892
R    571.373134
Name: FICO, dtype: float64
```

THE TWO-SAMPLE T-TEST

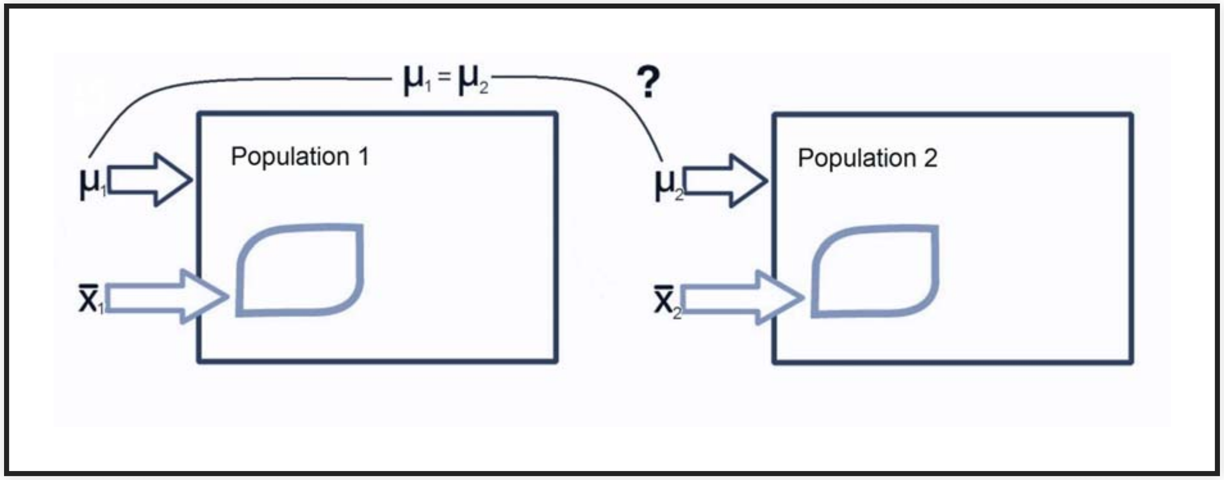
- Below is the formal output for comparing the two group means.
- We will review it shortly, but first develop some background for hypothesis testing.

```
# The two-sample t-test not assuming unequal variances.

# Extract the two columns of interest.
X1 = sm.stats.DescrStatsW(prsm_data['FICO'].loc[prsm_data['Loan.Type']=="O" ])
X2 = sm.stats.DescrStatsW(prsm_data['FICO'].loc[prsm_data['Loan.Type']=="R" ])

comp_means = sm.stats.CompareMeans(X1, X2)
print(comp_means.summary(usevar='unequal'))
```

```
                          Test for equality of means
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
subset #1     11.4748      4.726      2.428      0.015       2.195      20.754
==============================================================================
```

THE POPULATION/SAMPLE PARADIGM

- The graphic displays the idea of the population/sample paradigm that we follow in statistical hypothesis testing.



- We assume that we have a simple random sample of loans from the two populations.

THE HYPOTHESIS TESTING SETUP

- In English: the population means are the same, versus the population means are different.

$$H_0 : \mu_1 - \mu_2 = 0 \qquad v. \qquad H_1 : \mu_1 - \mu_2 \neq 0$$

- Though you could write the null hypothesis as $\mu_1 = \mu_2$, it is more elegant to think of $\mu_1 - \mu_2$ as the parameter of interest.
- To estimate the difference in population means, we use the difference in sample means: $\overline{x}_1 - \overline{x}_2$.
- The intuition is that if the difference in sample means, $\overline{x}_1 - \overline{x}_2$, is ***large*** then we have evidence against the null hypothesis.

THE HYPOTHESIS TESTING SETUP

- But what do we mean by ***large*** ? We need a reference to indicate whether the difference in sample means is large or small, on a *statistical* scale.
- The standard error of an estimate measures the sample to sample variability of the estimate.
- So we divide the difference in means, by the estimated std. error.
- This lets us count how many standard errors the observed difference in sample means is away from 0 (the null hypothesis value).

THE HYPOTHESIS TESTING SETUP

- If the standard error counter is a “long” way from zero, then the event is surprising given the null hypothesis is true.
- But statisticians aren’t meant to be surprised so we reject the null hypothesis when the standard error counter is large.
- But again, how large is large?
- That’s where the assumptions and theory come in; if we define **large** as 2 standard errors, then using this cut-off, the probability of rejecting the null, when the null is true is approximately 0.05.
- The “0.05” is called the Type 1 error rate of the hypothesis test.

CONFIDENCE INTERVALS

- A confidence interval provides a range of feasible values for the true population parameter.
- Here the population parameter is $\mu_1 - \mu_2$.
- We often work with 95% confidence intervals.
- Values of the parameter that lie inside the confidence interval are consistent with the observed data.
- Values of the parameter that are outside the confidence interval are inconsistent with the observed data.
- So we take the values inside the confidence interval as a working set of possible values for the parameter of interest.

P-VALUES

- p-values provide a measure of the credibility of the null hypothesis.
- They are calculated assuming the null hypothesis is true.
- They range in value from 0 to 1.
- The smaller the p-value the more evidence against the null, which really says that the data is inconsistent with the null hypothesis.
- Practitioners often use a cut-off of 0.05 for the p-value, to determine if the test is significant.
- When the p-value is less than 0.05, then we reject H_0 .

REVIEWING THE T-TEST OUTPUT

```
comp_means.summary(usevar='unequal')
```

Test for equality of means

	coef	std err	t	P>|t|	[0.025	0.975]
subset #1	11.4748	4.726	2.428	0.015	2.195	20.754

INTERPRETING THE OUTPUT

- ‘coef’ is the difference in sample means, 11.4748. (582.847892 -571.373134).
- ‘std err’ is the standard error of the difference in means, here it is 4.726.
- ‘t’ is the standard error counter, 11.4748/4.726 = 2.428. It is usually called the ‘t-statistic’.
- ‘P>|t|’ is the p-value, here it is 0.015.
- ‘[0.025, 0.975]’ are the end points of the 95% confidence interval, (2.195, 20.754).
- The bottom line is that what we have observed is a rare event if the null is true, so we reject the null hypothesis and declare a significant difference in populations means.
- Why is it a ‘rare’ event under the null? Because (and they are equivalent):
  - The confidence interval does not contain 0.
  - The t-statistic is greater than 2.
  - The p-value is less than 0.05.

REVIEW THE CODE FOR THE TWO-SAMPLE T-TEST

- ‘DescrStatsW’ is a function that summarizes a variable.
- ‘CompareMeans’ calculates the elements required for the two-sample t-test.
- The summary method ‘.summary’, then provides a display of the results.
- The “usevar=‘unequal’” is an option that says we do not assume the population variances are equal.

```
# The two-sample t-test not assuming unequal variances.
# Extract the two columns of interest.
X1 = sm.stats.DescrStatsW(prsm_data['FICO'].loc[prsm_data['Loan.Type']=="O" ])
X2 = sm.stats.DescrStatsW(prsm_data['FICO'].loc[prsm_data['Loan.Type']=="R" ])

comp_means = sm.stats.CompareMeans(X1, X2)
print(comp_means.summary(usevar='unequal'))
```

```
                          Test for equality of means
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
subset #1     11.4748      4.726      2.428      0.015       2.195      20.754
==============================================================================
```

REGRESSION

REGRESSION

- Regression models let us relate a predictor(s) (x-variable) to an outcome (y-variable).
- They can be used both for *interpretation* and *prediction* .
- We will go to the Stats notes to introduce regression in more detail, then implement regression in Python.

USING SEABORN TO VISUALIZE THE REGRESSION

USING SEABORN TO VISUALIZE THE REGRESSION

- We start by using a small dataset that relates the size of a house in square feet to its valuation. All the houses are in the same neighborhood, sharing the same school district, etc.
- Note that the outcome variable (Valuation) is continuous.
- Regular regression is for a continuous y-variable.

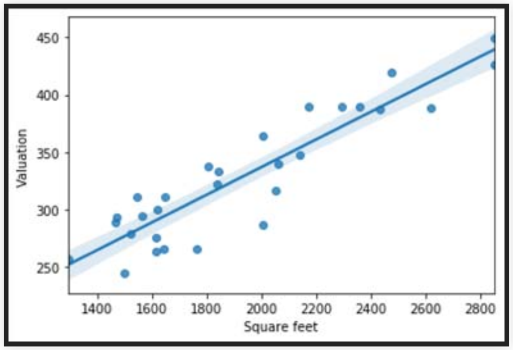
```
house_data = pd.read_csv("real_housing.csv")
print(house_data.head(5))
```

```
   Valuation  Square feet
0        311         1544
1        293         1470
2        311         1650
3        333         1843
4        294         1565
```

BASIC REGRESSION PLOT

- Seaborn has two regression plotting functions, ‘regplot’ and ‘lmplot’.
- ‘regplot’ does simple regression in one graph, whereas ‘lmplot’ let’s you do faceting, that is using multiple panels to display the regression, conditional on other variables.
- Below is the default regplot visualization, with the points, regression line and confidence bands for the regression.

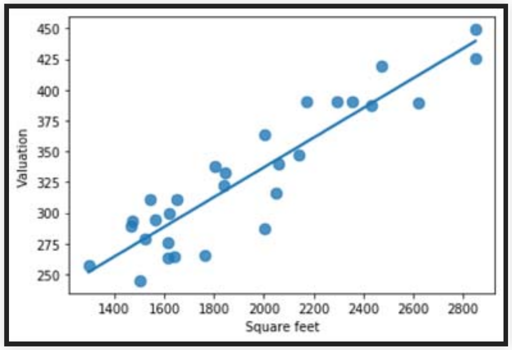
```
sns.regplot(x="Square feet", y="Valuation", data=house_data);
```



CHANGING PLOTTING CHARACTERISTICS

- You can change features and the style of the plot through arguments, and by passing arguments down to the underlying scatter and plot functions.
- Below the points are made larger and the confidence bands removed.

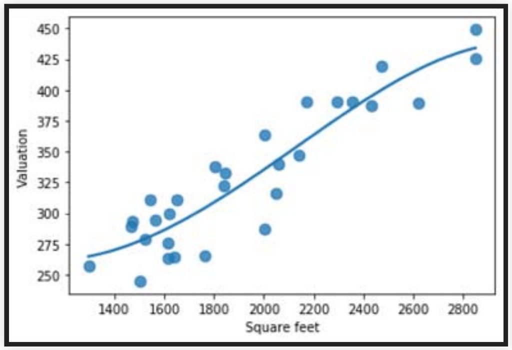
```
sns.regplot(x="Square feet", y="Valuation", data=house_data, ci=None, scatter_kws={"s": 75});
```



A POLYNOMIAL FIT TO THE DATA

- Using the ‘order’ argument, you can fit a polynomial to the data.
- Setting ‘order’ equal to 1 gives a line, order=2 is a quadratic, 3 is a cubic and so on.
- The polynomials allow ‘curvature’ to be accommodated in the model, but should not be used for extrapolation.

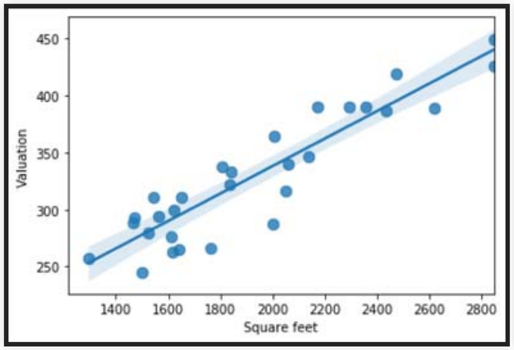
```
sns.regplot(x="Square feet", y="Valuation", order=4, data=house_data, ci=None, scatter_kws={"s": 75});
```



USING A “ROBUST” FIT

- An alternative to the “least squares” model fitting criteria is to use a “robust” fit.
- This downweights points with large residuals and could be a good choice if the underlying data is outlier prone.
- Here it won’t make much of a difference, because we don’t have outliers.
- The argument to do the robust fit is ‘robust’.

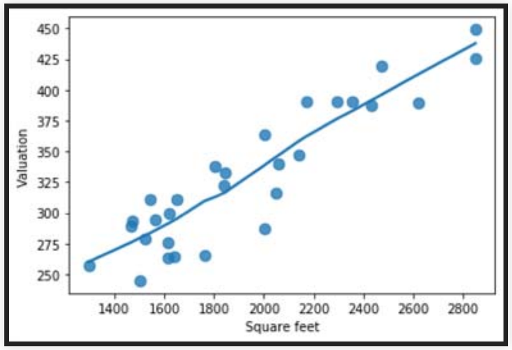
```
sns.regplot(x="Square feet", y="Valuation", robust=True, data=house_data, ci=95, scatter_kws={"s":
  75});
```



A LOWESS SMOOTH

- The "lowess" technique is an example of a "scatterplot smoother".
- It is a ***data driven*** method that seeks to capture the relationship between x and the mean of y, regardless of the underlying true functional form.

```
sns.regplot(x="Square feet", y="Valuation",  data=house_data, ci=None, lowess=True, scatter_kws={"s":
  75});
```



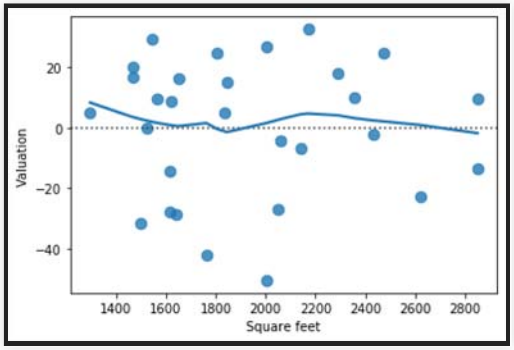
VIEWING THE RESIDUALS FROM THE REGRESSION

- Part of standard operating procedure, should be to review the residuals from the regression.
- Ideally there should be no pattern.
- If there is a pattern, then you may want to change the form of the regression function, for example use a log curve.

VIEWING THE RESIDUALS FROM THE REGRESSION

- The residual plot below is a “good” one, as there is no discernible pattern.
- Seaborn has a ‘residplot’ function to plot the residuals from the linear regression.
- There are alos arguments to use a polynomial fit, or lowess smoother for the fit.

```
sns.residplot(x="Square feet", y="Valuation",  data=house_data,  lowess=True, scatter_kws={"s": 75});
```



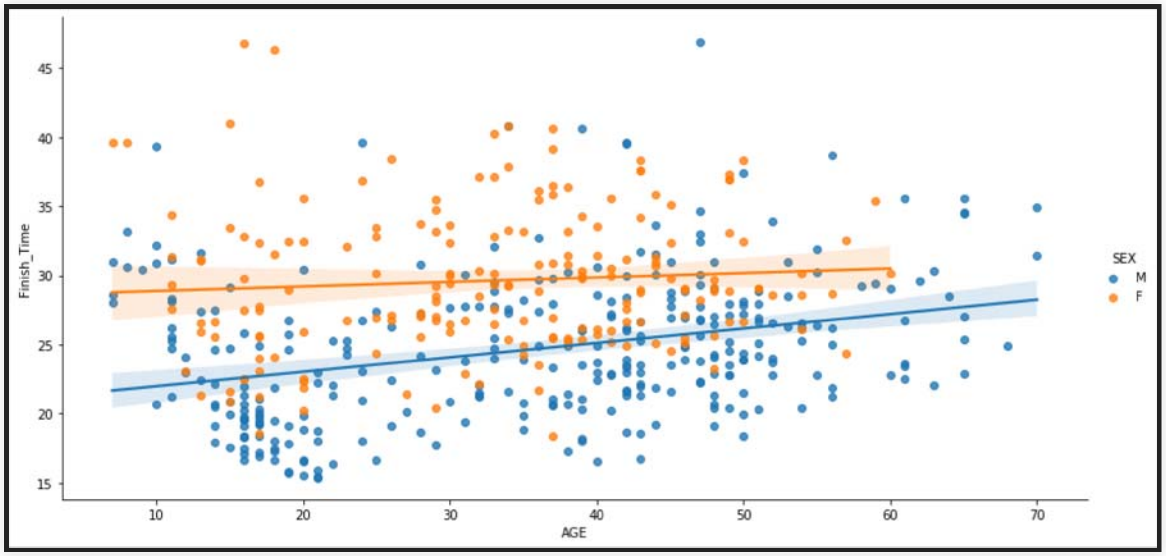
PLOTTING REGRESSIONS OVER A GROUPING VARIABLE

- When there are categorical variables available, it makes sense to explore regression lines for the levels of the categorical variable.
- We will use the “Run5K” dataset, which has the categorical variable SEX, either M or F.
- By using the ‘hue" argument to ’lmplot’ you get a regression line for each of M and F, on the same graph.
- This makes it very easy to compare the lines.

A REGRESSION LINE FOR EACH GROUP

- Notice that the male line is above the female line, and the slopes are different.
- There maybe a differential impact of aging on race time, according to sex.

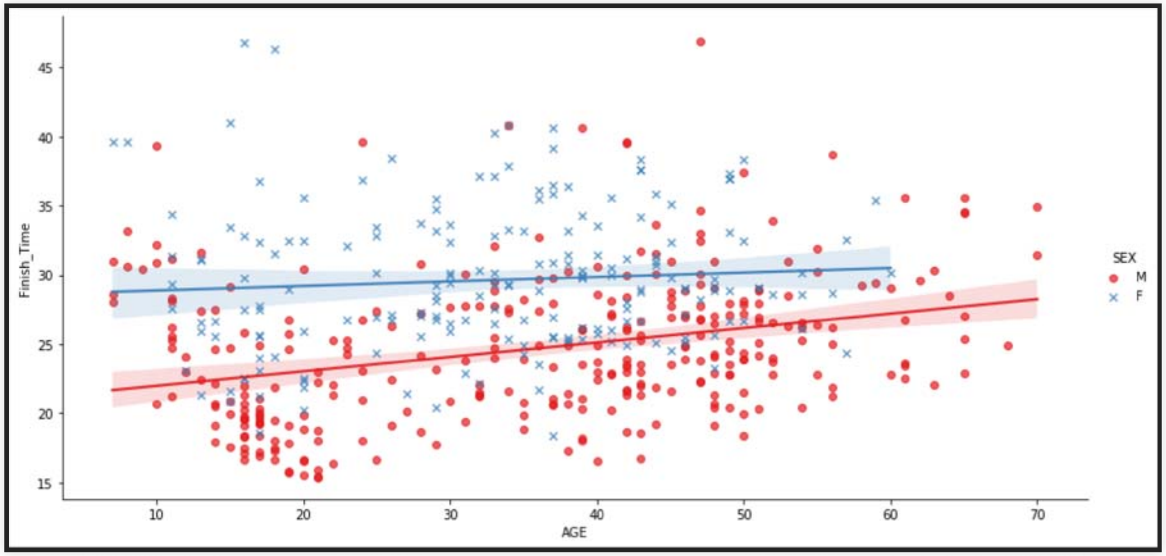
```
race_data = pd.read_csv("Run5K.csv")
race_data['Finish_Time'] = race_data['Time_mins'] + race_data['Time_secs']/60 # Calculate the actual
  race time.
sns.lmplot(x="AGE", y="Finish_Time", hue="SEX", data=race_data, height=6, aspect=2);
```



TWEAKING THE PLOT ARGUMENTS

- Note the *marker* and *pallet* arguments, with additional arguments (scatter_kws, line_kws) being passed down to the underlying plot functions.

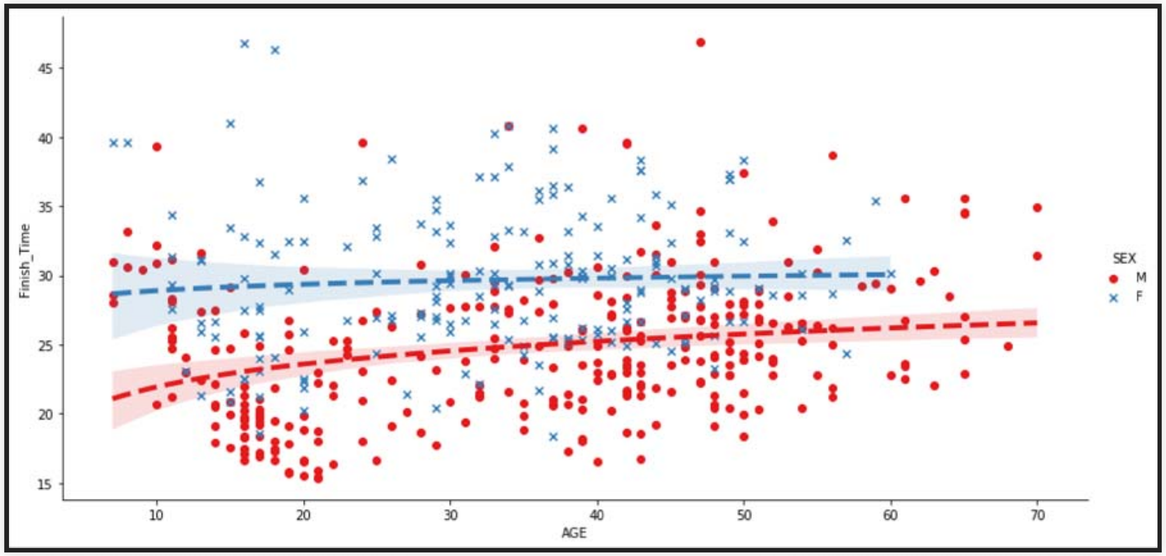
```
sns.lmplot(x="AGE", y="Finish_Time", hue="SEX", data=race_data, markers=["o", "x"],
           palette="Set1",scatter_kws={'alpha':0.7}, line_kws={'alpha':0.9} ,height=6, aspect=2);
```



FITTING USING LOG(X)

- Many relationships are logarithmic rather than linear.
- Using the ‘logx = True’ options, lets you plot the best fitting log curves.

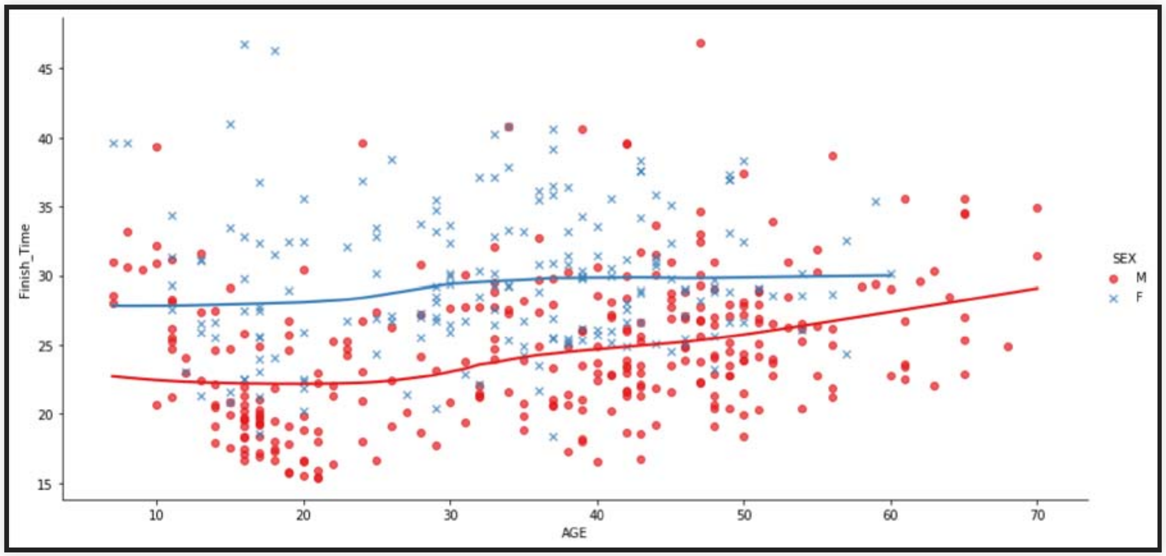
```
sns.lmplot(x="AGE", y="Finish_Time", hue="SEX", data=race_data, markers=["o", "x"], palette="Set1",
           scatter_kws={'alpha':1.0}, line_kws={'alpha':1, 'linestyle':'dashed', 'linewidth':4},
           logx=True, height=6, aspect=2);
```



PUTTING TWO LOWESS SMOOTHS

- You can also view two lowess smooths for a ‘non-parametric’ (no model) fit.

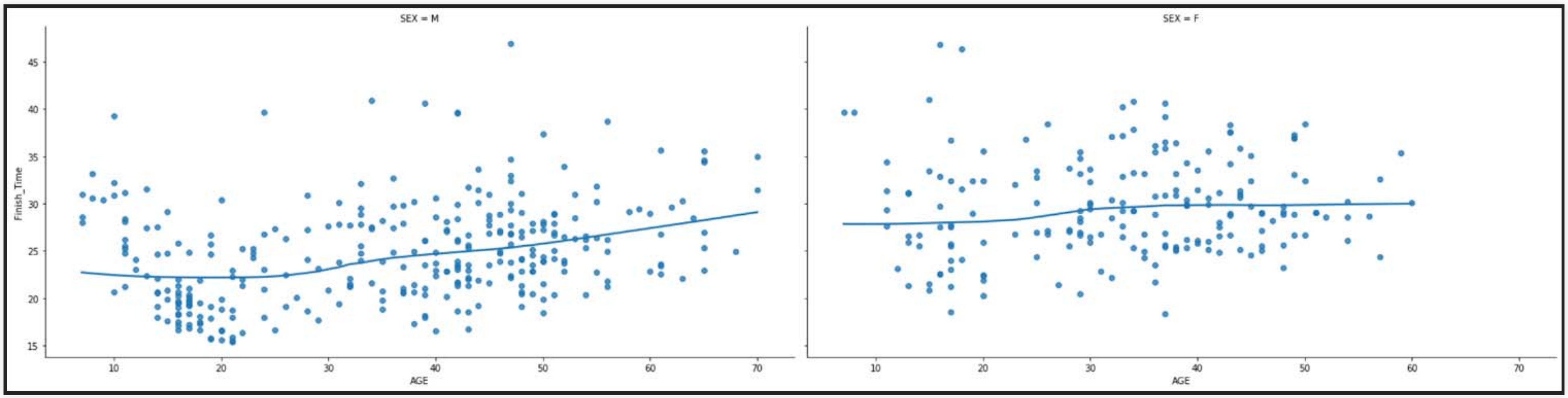
```
sns.lmplot(x="AGE", y="Finish_Time", hue="SEX", data=race_data, markers=["o", "x"], palette="Set1",
           scatter_kws={'alpha':0.7}, lowess=True, height=6, aspect=2      );
```



SIDE BY SIDE PLOTS

- By using the ‘col’ and ‘row’ arguments you can create a grid of plots.
- This allows you to look at the regression, conditionally on other variables.

```
sns.lmplot(x="AGE", y="Finish_Time", col="SEX", data=race_data, palette="Set1", lowess=True, height=6,
  aspect=2);
```



BINARY OUTCOME DATA

BINARY OUTCOME DATA

- When the outcome variable takes on two levels (binary), we use what is called a 'logistic regression' to model the probability of being in each of the two categories.
- The key feature of a logsitic regression is that the predictions always lie between 0 and 1.
- If you used a regular regression line, the line would eventually go above 1 and below 0, giving nonsensical predictions for the probabilities.
- On the next slide we subset the outpatient data so that there are only two outcomes, 'Arrived' and 'No Show'.
- We also turn the outcome variable, Status, into a 0/1 variable by using the 'pd.get_dummies' function.
- There was also a gross outlier that I dropped as well.

CREATING A BINARY OUTCOME FOR STATUS

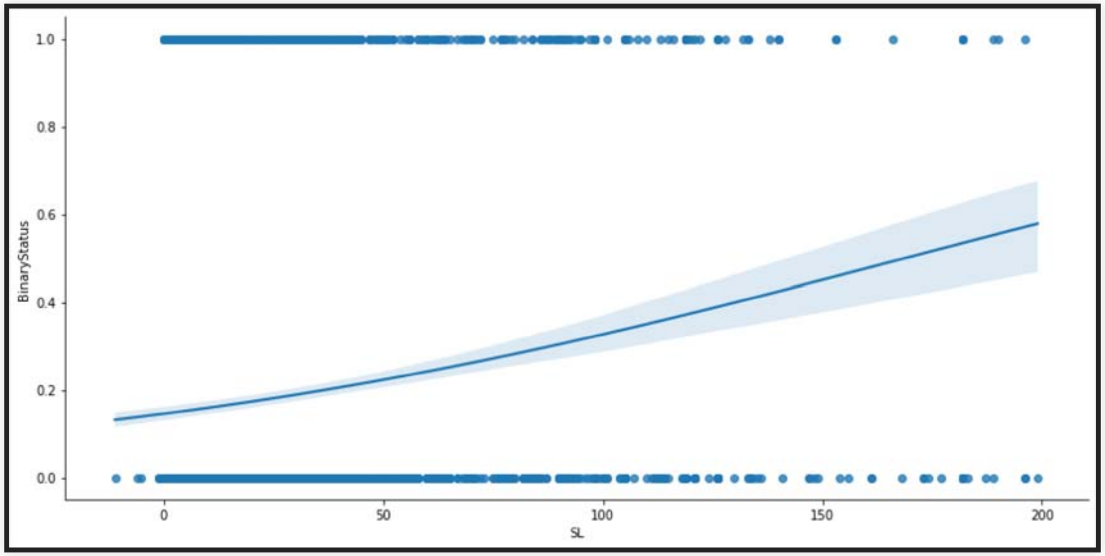

```
op_data = pd.read_csv("Outpatient.csv", parse_dates=['SchedDate', 'ApptDate'])
op_data['ScheduleLag']  = op_data['ApptDate'] - op_data['SchedDate']
op_data['SL'] = op_data['ScheduleLag'].dt.days # Create a new variable that has schedule lag in days.

# Create a dichotomous  Status variable.
op_subset = op_data.loc[(op_data['Status'] == "Arrived") | (op_data['Status'] == "No Show")]
op_subset['Status'].value_counts()

# Create a 0/1 version of the variable with the 'get_dummies' function.
op_subset.insert(op_subset.shape[1], 'BinaryStatus', pd.get_dummies(op_subset["Status"],
  drop_first=True))
print(op_subset.head(4))
# Drop the gross outlier (371 days)
op_subset = op_subset[op_subset['SL'] < 250]
```

```
      PID  SchedDate   ApptDate             Dept Language Sex  Age  \
0  P10092 2012-07-27 2012-10-05             DERM  ENGLISH   F  80+
2  P10962 2012-02-02 2012-02-10  OTOLARYNGOLOGY  ENGLISH   M   12
4  P10320 2012-10-25 2012-12-11      NEPHROLOGY  SPANISH   F   45
6  P10410 2013-10-31 2013-11-03    ORTHOPAEDICS  ENGLISH   M   54

                Race   Status ScheduleLag  SL  BinaryStatus
0  AFRICAN AMERICAN  Arrived     70 days  70             0
2  AFRICAN AMERICAN  Arrived      8 days   8             0
4          HISPANIC  No Show     47 days  47             1
6  AFRICAN AMERICAN  No Show      3 days   3             1
```

PLOTTING A LOGISTIC REGRESSION

- The curve models the probability of being a No Show as a function of Schedule lag.
- The key argument is ‘logistic=True’.

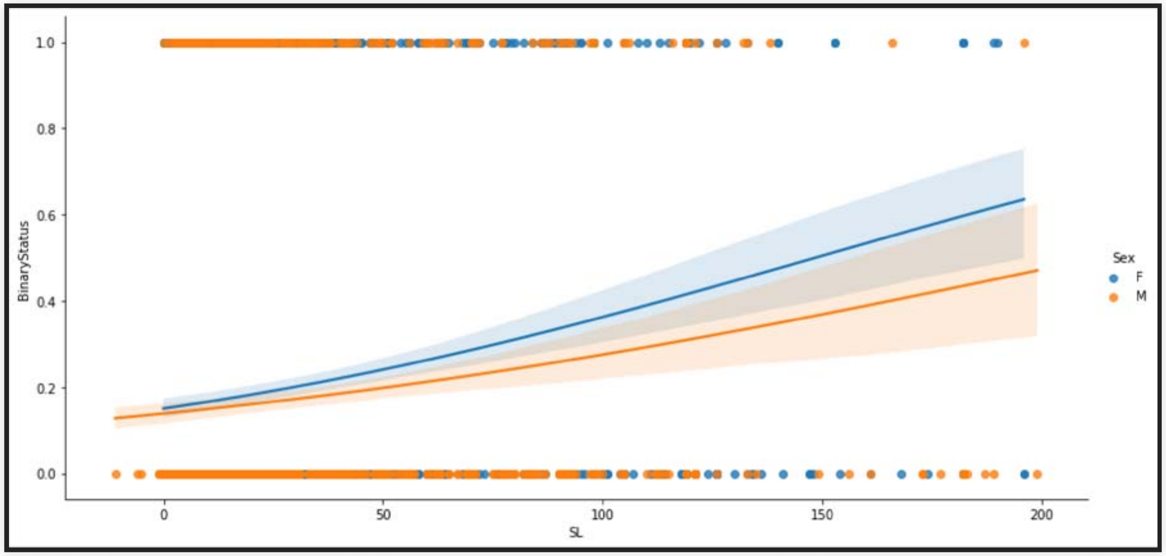
```
sns.lmplot(x="SL", y="BinaryStatus", data=op_subset, logistic=True, height=6, aspect=2);
```



A LOGISTIC CURVE FOR EACH GROUP

- Using the ‘hue’ argument allows us to plot the logistic over the levels of SEX.
- Females seem to have a higher probability of being a No Show, than males though there is a lot of overlap in the confidence bands.

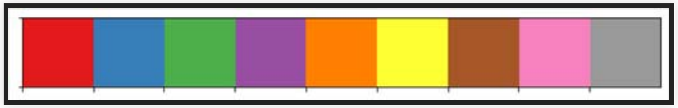
```
sns.lmplot(x="SL", y="BinaryStatus", data=op_subset, hue='Sex', logistic=True, height=6, aspect=2);
```



PLAYING WITH THE COLOR WIDGETS

- Just for fun, below is a seaborn color widget, but you need to do this in Jupyter to actually use it!

```
sns.choose_colorbrewer_palette("q");
```



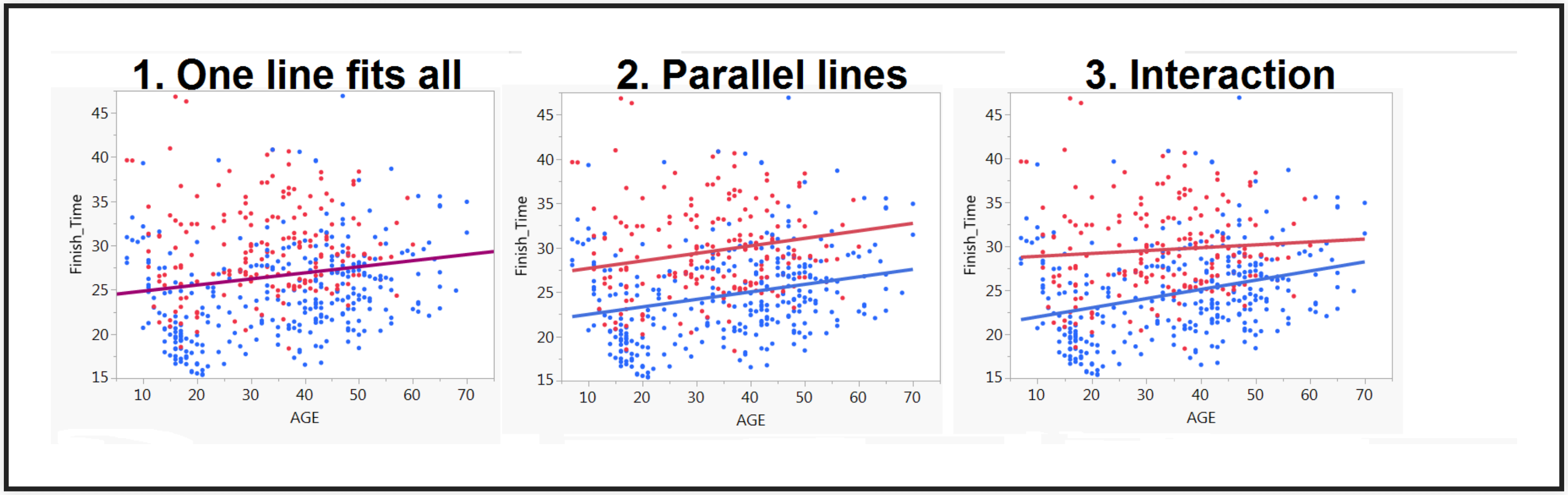
NUMERICAL REGRESSION RESULTS AND PREDICTION

NUMERICAL REGRESSION RESULTS AND PREDICTION

- All the plots we have being doing are simply illustrative of the relationships between the variables.
- In order to do formal hypothesis testing we need to get at the underlying regression lines.
- Likewise, if we want to do prediction we need the actual formulas for the regression lines.
- To do this, we will switch to the ‘statmodels’ library, where we can fit and query these models.

THREE MODELS

- We will fit and discuss the following three models:
  - One line fits all.
  - Parallel lines.
  - Interaction.



THE SIMPLE REGRESSION MODEL

- The models can be specified using a special formula syntax that is borrowed from the ‘R’ language.
- On the next slide we fit the simple regression model (one line fits all) with the model syntax:

```
formula='Finish_Time ~ AGE'
```

FITTING A SIMPLE REGRESSION MODEL

```
import statsmodels.formula.api as smf
olsmod1 = smf.ols(formula='Finish_Time ~ AGE', data=race_data) # Define the model.
olsres1 = olsmod1.fit() # Fit the model.
print(olsres1.summary()) # View the results.
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Finish_Time   R-squared:                       0.030
Model:                            OLS   Adj. R-squared:                  0.028
Method:                 Least Squares   F-statistic:                     15.53
Date:                Thu, 23 Jul 2020   Prob (F-statistic):           9.28e-05
Time:                        15:03:58   Log-Likelihood:                -1579.4
No. Observations:                 501   AIC:                             3163.
Df Residuals:                     499   BIC:                             3171.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     24.1633      0.656     36.812      0.000      22.874      25.453
```

DISCUSSION

- There is a lot of output!
- But note the coefficient estimates for the Intercept and AGE variable, 24.16 and 0.0668 respectively.
- These two numbers define the fitted regression line.
- There are also confidence intervals for the estimates.
- The R^2 of only 3%, indicates that AGE does not explain much variability in Finish_Time at all.
- But that does not imply that there isn't a story to be told about the general impact of aging.
- There is also a big underlying subtlety here, and that is that there maybe a lot of self selection in the age groups.
- Maybe lots of young people run, but only fit and good old people enter races!
- Regression does not address this subtlety in any way.

PREDICTION

- To do prediction we create a new data frame and pass it in to the '.predict' method.
- We get predictions for the three fictional runners.

```
Xnew = pd.DataFrame({'AGE': [32,43,16]}) # Create new data to predict at.
yprednew = olsres1.predict(Xnew)
print(yprednew)
```

```
0    26.366174
1    27.123395
2    25.264760
dtype: float64
```

ADDING THE SEX VARIABLE TO THE REGRESSION

- When we add the SEX variable to the model, then we have moved from “Simple regression” to “Multiple regression”.
- But because the SEX variable is categorical, it must be made numeric.
- We could use the ‘pd.get_dummies’ function to do this, as below, but the good news is that it will all happen behind the scenes if we use the formula interface to specify the model.

```
print(pd.get_dummies(data=race_data['SEX']))
```

```
     F  M
0    0  1
1    0  1
2    0  1
3    0  1
4    0  1
..  .. ..
496  0  1
497  1  0
498  1  0
499  1  0
500  0  1
```

```
[501 rows x 2 columns]
```

THE PARALLEL LINES MODEL REGRESSION MODEL

- Note the addition of the SEX variable to the formula:

```
formula='Finish_Time ~ AGE + SEX'
```

- We add additional variables to the model, simply by using the plus sign.
- A regression for data mining can have hundreds of variables!

FITTING THE PARALLEL LINES MODEL

```
olsmod2 = smf.ols(formula='Finish_Time ~ AGE + SEX', data=race_data) # Specify model.
olsres2 = olsmod2.fit() # Fit model.
print(olsres2.summary()) # Summarize model.
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Finish_Time   R-squared:                       0.221
Model:                            OLS   Adj. R-squared:                  0.217
Method:                 Least Squares   F-statistic:                     70.45
Date:                Thu, 23 Jul 2020   Prob (F-statistic):           1.14e-27
Time:                        15:03:59   Log-Likelihood:                -1524.7
No. Observations:                 501   AIC:                             3055.
Df Residuals:                     498   BIC:                             3068.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     26.8253      0.637     42.139      0.000      25.575      28.076
```

INTERPRETING COEFFICIENTS

- The key coefficient is the "SEX[T.M] = -5.1852' which indicates that for a given AGE, men are expected to be about 5 minutes faster than women.
- The regression equation has two cases, one for Men and the other for Women:
  - Men: Finish_Time = 26.8235 - 5.1852 + 0.0846 * Age
  - Women: Finish_Time = 26.8235 + 0.0846 * Age
- Women are described as the 'baseline' or 'reference' category.
- The categorical variable coefficient gives the difference in height between the two lines.
- Also note that R^2 increased dramatically from the initial model of 3% to 22% here.

THE INTERACTION MODEL

- This is the model that allows for a different slope for each group.
- Generically we say “the impact of X1 on Y depends on the level of X2”.
- The formula now contains the colon ‘:’ to specify the interaction term:

```
Finish_Time ~ AGE + SEX + AGE:SEX
```

FITTING THE INTERACTION MODEL

```
olsmod3 = smf.ols(formula='Finish_Time ~ AGE + SEX + AGE:SEX', data=race_data) # Specify model.
olsres3 = olsmod3.fit() # Fit model.
print(olsres3.summary()) # Summarize model.
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Finish_Time   R-squared:                       0.227
Model:                            OLS   Adj. R-squared:                  0.222
Method:                 Least Squares   F-statistic:                     48.65
Date:                Thu, 23 Jul 2020   Prob (F-statistic):           1.39e-27
Time:                        15:03:59   Log-Likelihood:                -1522.6
No. Observations:                 501   AIC:                             3053.
Df Residuals:                     497   BIC:                             3070.
Df Model:                           3
Covariance Type:            nonrobust
================================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept     28.5391      1.053     27.110      0.000      26.471      30.607
```

INTERPRETING COEFFICIENTS

- In this regression the key term is the interaction term ‘AGE:SEX[T.M] = 0.0717’.
- This tells us that the **slope** for the Male line is 0.0717 higher than the female line.
- Again we can write down the regression line for each group:
  - Male: Finish_Time = 28.5391 -7.6032 + 0.0326 * AGE + 0.0717 * AGE
  - Female: Finish_Time = 28.5391 + 0.0326 * AGE
- With these equations we could do prediction ‘by hand’, but it is better to use Python!

PREDICTION IN THE MULTIPLE REGRESSION

- Once again we need to create a prediction data frame of the x-variables, which we do below.
- We then use the predict method for the model of interest.

```
Xnew = pd.DataFrame({'AGE': [21,43,21], 'SEX':['M','M','F']})
#Xnew = sm.add_constant(Xnew)
print(Xnew, "\n")
yprednew = olsres3.predict(Xnew)
print(yprednew)
```

```
   AGE SEX
0   21   M
1   43   M
2   21   F

0    23.126859
1    25.422101
2    29.224650
dtype: float64
```

THE ROOT MEAN SQUARED ERROR (RMSE)

- To create an approximate 95% prediction interval for the Finish_Time, so long as the prediction is within the range of the x-variables, you can simply add +/-2RMSE to the prediction (assuming the residuals are normally distributed).
- You get the RMSE for the model by extracting the ‘mse_resid’ attribute and square rooting it.
- Here, it is about 5, so the prediction intervals will be +/- about 10 minutes!
- Not very precise.

```
import math
math.sqrt(olsres3.mse_resid)
```

```
5.074349067236697
```

SUMMARY

SUMMARY

- Statistical comparisons.
- Hypothesis testing.
- Seaborn regression plots.
- Fitting the regression with ‘statsmodels’.
- Prediction from regression.
- Multiple regression:
    - One line fits all.
    - Parallel lines.
    - Interaction (different slopes).

NEXT TIME

NEXT TIME

- The machine learning world view