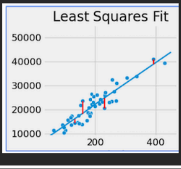
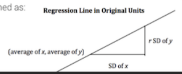
Fitting The Models

- Linear Regression (Least Squares Regression)
  - For our loss, the way we figure out how far many data points are from our metric is by looking at the residuals (how far they are from our line) of every data point. Minimize the sum of the residuals (ie. set the derivative in the x direction = 0 and y direction = 0)



Recall the least squares linear regression model is defined as:

$$\hat{y} = \boxed{(\frac{r\sigma_y}{\sigma_x})} \times x + \boxed{(\bar{y} - \frac{r\sigma_y}{\sigma_x}\bar{x})}$$


Regression Line in Original Units
r SD of y
(average of x, average of y)
SD of x

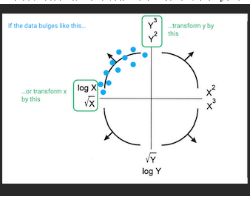
$$\hat{y} = r \times x \; [both\ measured\ in\ standard\ units]$$

Correlation Coefficient $r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma_x}\right)\left(\frac{y_i - \bar{y}}{\sigma_y}\right)$

$$\frac{\text{SD of fitted values}}{\text{SD of } y} = |r| \;\Rightarrow\; \boxed{\frac{\text{variance of fitted values}}{\text{variance of } y} = r^2}$$

y in standard units = r *x in standard units*
slope = *r* (y_std / x_std)
intercept = y.mean - (slope * x.mean)
Ex. $mpg = \Theta_0 + \Theta_1 \cdot horsepower$

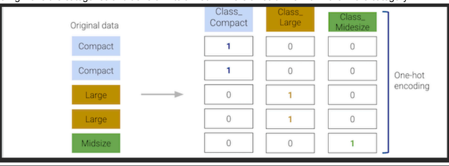
- Linear Regression With Transformation: If your data bulges in a direction, transform x and/or y in that direction to make data more linear and then perform linear regression. Keep going with powers of X or logs of Y until it looks linear



`data["Log(Age)"] = np.log(data["Age"])`

Map Back: Just remember to do `y = e^log(x)` when trying to get the y val from a model like this (ie. plug log(x) in as input instead of just x)

  - Feature Engineering: process of transforming raw features into more informative features that can be used in modeling (deep learning is the process of automating this). Think of original dataset as a matrix and then apply a transformation to the matrix
    - (1) Look for uninformative features you can remove, (2) use bulge diagram for quantitative features, (3) put data in standard units, (4) take categorical features and incorporate them into model
    - One-Hot Encoding: transform qualitative data to features for modeling. Take the categories and transform to a matrix where all columns with 1 are in the category



Original data	Class_ Compact	Class_ Large	Class_ Midsize
Compact	1	0	0
Compact	1	0	0
Large	0	1	0
Large	0	1	0
Midsize	0	0	1

One-hot encoding

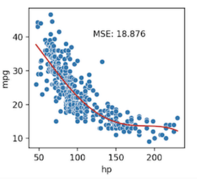
```
ohe = OneHotEncoder()
ohe.fit(hybrid[["class"]])
encoded_class = ohe.transform(hybrid[["class"]]).todense()
encoded_class_df = pd.DataFrame(encoded_class, columns=ohe.get_feature_names_out(), index = hybrid.index)
```

This fits a coefficient for every single category in the dataset:

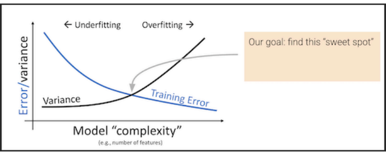
$$\text{MSRP} = \theta_1\text{acceleration} + \theta_2\text{mpg} + \theta_3(acceleration)^2 + \theta_4\sqrt{mpg} + \theta_5(classCompact) + \theta_6(classLarge) + \theta_7(classMidsize)\ldots + \theta_8(classMinivan) + \theta_9(classPickupTruck) + \theta_{10}(classSUV) + \theta_{11}(classTwoSeater)$$

  - Adding more polynomial features: what if we let our curve have higher degrees (more places it can change) to decrease training error

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3) + \theta_4(\text{hp}^4)$$



- Overfitting: if you try to have 0 MSE your model becomes useless. To correct for this, look at the variance of the model (ie. if it's given many different small samples from one dataset it should make roughly the same model every time). To solve:



1. Cross Validation Holdout Method: take the data we're given and split into train and test set. Then split train into train and validation. During model training, use validation to make sure we don't overfit. Assess the final performance with the test set
   - If training error > test error, then we've overfit!
   - Once we use our validation to figure out best degree, use both training and val to train model!
2. K-Fold Cross Validation: what if we accidentally select a validation set that isn't representative of the overall data? We can check by testing K different validation sets (and retraining the model K different times). 5-10 folds is optimal
   - Cross-Validation Error: average MSE across all K-validation sets, goal is to make this as low as possible. Pick whichever model has the lowest Cross-Validation Error

- Multiple Linear Regression: Must be a linear combination in Θ (check if you can separate into a vector of non-Thetas and a vector of Thetas)

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p$$

Ex.

$$\begin{bmatrix}\hat{y}_1\\ \hat{y}_2\\ \hat{y}_3\end{bmatrix} = \begin{bmatrix}1 & x_{11} & x_{12} & x_{13}\\ 1 & x_{21} & x_{22} & x_{23}\\ 1 & x_{31} & x_{32} & x_{33}\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \theta_3\end{bmatrix}$$

Instead of a line, fit to a plane (for 2 variables)

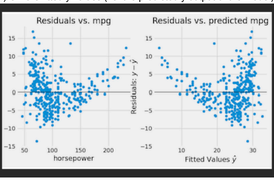
Evaluating The Models

- Evaluating Linear Regression
  - Root Mean Square Error: the square of the root of MSE which is the average loss we've been minimizing to determine optimal model parameters (the lower the better)

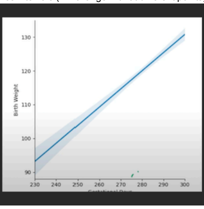
$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
RMSE = np.sqrt(np.mean((predicted_mpg - vehicle_data['mpg'])**2))
```

  - Look at residual plot (difference between actual and predicted values). If residuals are randomly scattered you're doing well. If there's some pattern than you shouldn't be using a Linear Model.
    - Put the residuals on the y-axis. For x-axis either use the input feature (ie. `horsepower`) OR the fitted y-values (i.e. the predicted y output of the model)



  - Confidence Intervals
    - Estimate Confidence Interval for Regression Line: because of sample variability the line may not be truly accurate so give a confidence interval
      1. Fit the regression line to sample, Bootstrapping: Sample with replacement from distribution, Fit the regression line
      2. We'll get a ton of slopes. Then plot a histogram of all slopes. Find the 95th percentile of the distribution. Answer: Confidence Interval = Slope between `[0.46, 0.612]`
         How do you know if your line is statistically significant or not? Do bootstrapping with (null = the line has slope of 0) (ie. no correlation between X and Y). Do bootstrapping a bunch of times and graph distribution of regression lines. If the confidence interval encompasses 0, not statistically significant (ie. can't reject null). If it doesn't encompass 0 then it is statistically significant (ie. reject null). BUT CAN'T MAKE CAUSATION CLAIMS
    - Estimate Confidence Interval for Predictions of Regression Line (Mean Response): Just do same bootstrapping thing but look at the line's prediction at a certain value each time (instead of the line's slope). Graph this in distribution and find 95th percentile
      - Confidence Band: Make a confidence interval at every single point and then plot all confidence intervals (will change with at different points)



- Evaluating Multiple Linear Regression
  1. Ordinary Least Squares: just use mean squared error on multiple linear regression (From linear algebra, we can get a nice formula to get our best theta values). If this goes down we're happy
  2. Multiple R-Squared (coefficient of determination): analog for root mean squared error. If this goes up we're happy

$$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y} = \frac{\sigma_{\hat{y}}^2}{\sigma_y^2}$$

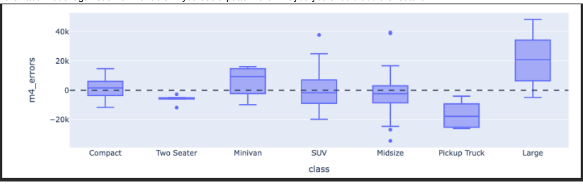
$$\text{Adjusted } R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

N: sample size
p: number of features/predictors

  Caveat: as we add more features ^{2} increases so usually there's some sort of penalty to balance out

  3. Look at residuals (if there's patterns then you need to add more datapoints or do better)
     - Put the residuals on the y-axis. For Multiple Linear Regression plot the residuals vs the fitted y-values

- To choose new features to add, plot residuals from the fitted model against a new variable. If you see a pattern then maybe you should add the feature



Implies that we should add car class to our model (if all boxes were centered around median, we wouldn't need to add)

Examples

Ex. Course has midterm (avg=70, SD=10) and hard final (avg=50, SD=12). Correlation 0.75. What is the avg final score for students who scored 90 on the midterm?
y in standard units = r * x in standard units
((y-50)/12) = .75((x-70)/10) = .75((90-70)/10) -> y = 50 + 18 = 68% on the exam

- Code (For linear regression and multiple linear regression)

```
# automatically
my_model = lm.LinearRegression() #creates the "idea" of a linear regression model
my_model.fit(X, y) #takes in two inputs: the design matrix, `X`, and the target variable, `y` and figure out optimal theta
predicted_mpg = my_model.predict(vehicle_data[["horsepower"]]) # for multiple linear regression pass array of many vars in
my_model.score(X,y)
my_model.intercept_
my_model.coef_ #returns all theta values (my_model.coef_[0] = slope

# manually
def fit_line(df, x_col, y_col):
    x_std = np.std(df[x_col])
    y_std = np.std(df[y_col])
    r = correlation(df, x_col, y_col)
    slope = r * (y_std / x_std)
    intercept = np.mean(df[y_col]) - (slope * np.mean(df[x_col]))
```

[Penalized Regression]
[Logistic Regression]