Week 2, 01-14-2022

UCLA CS148 Section 1C

Checking in!

- How's everyone doing?
  - Social Directory
  - Project 1

- Material for the week:
- https://drive.google.com/drive/folders/16BOVX0C1mm1nCAtQMdpdbkdba47uyICA?usp=sharing

Me this week:

But hey a bit
Of good news:

Best Zoom Background Contest Time!

- Let's get ready to rumble!
- Webcams on let's show off those backgrounds!
- Best background, (as decided by the votes of your peers!) will win a Starbucks Giftcard
  - Link to vote here:
    - https://forms.gle/rQoRwJQBjzdJ6jrk7

HWs and Projects and Bears oh my!

- **Project1 DUE Wednesday the JAN 19 before lecture**

- Submit via Gradescope

When you haven't started your project yet, and it's due at 10am tomorrow:

Scavenger Hunt Discussion!

- 

Project 1

Doing Data Science: General Thoughts - Think Cooking!

- More Art than Science
- Build out a toolkit
  - Understand the ‘ingredients’ you’re given and generally how best to do it.
  - Beyond that, it’s a bit of an artform
- Throw the kitchen sink at it and see what sticks
- There’s a reason Kaggle exists for DS!
- Project 1 = Bake your first cake

Let's talk Project 1!

- General Thoughts:
  - Assumption that you've never done datascience projects like this before
  - Have you dive right in
  - Provide a detailed example walk through that you can basically replicate for your own code
- Methodology:
  - No one 'right' way to implement, consider the provided code a suggestion rather than a requirement
  - We want to see you go through the steps but are agnostic to methodology
  - Not graded on model performance!

Project 1 Spec

- As a first step it's critical to understand the data you're working with, both to assess opportunities to leverage it, and also to determine any issues you may have to resolve before completing your model
- Get Acquainted with the data you have
  - Load the dataset
  - Display the first few rows of the data
  - Generate statistics of the loaded data
  - Visualize Data
    - Plot histograms
    - Plot relationships
    - Map Data
  - Calculate Correlation Matrix

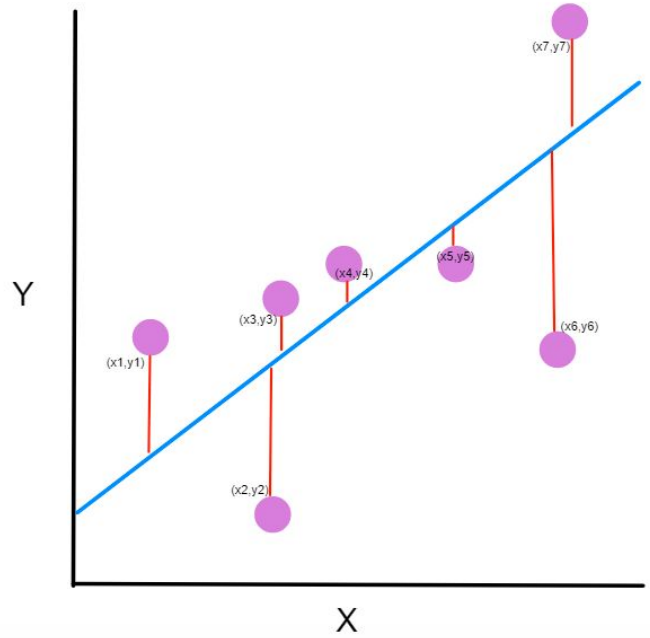
How do you pronounce it...

Project 1 Spec (Cont.)

- Once you know what you're working with, you devise and implement a strategy for how you'll be preparing your data for insertion into your learning model
- Data Prep
  - Drop columns from data
  - Augment data with new features
  - Impute missing data elements
  - Divide dataset into Train and Test cohorts
    - Why do we do a Train/Test split in DS projects? Shouldn't the more data the better?
- Once you have your process in place standardize it with a pipeline!

Project 1 Spec (cont.)

- Now that your data is all set, it's finally time to do the 'Machine Learning' component of your data science project
- Here you'll feed your data into a learning mechanism and it will attempt to train a model that best fits the data it is given
- Train your learning model!
  - Linear Regression
    - Train it, then test it, then report out results
  - How do we measure the success of our learning model?
    - Depends on the model!
  - Root Mean-Square Error
    - Question: Why do we take the Square of our error terms?



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

Submission Guidelines

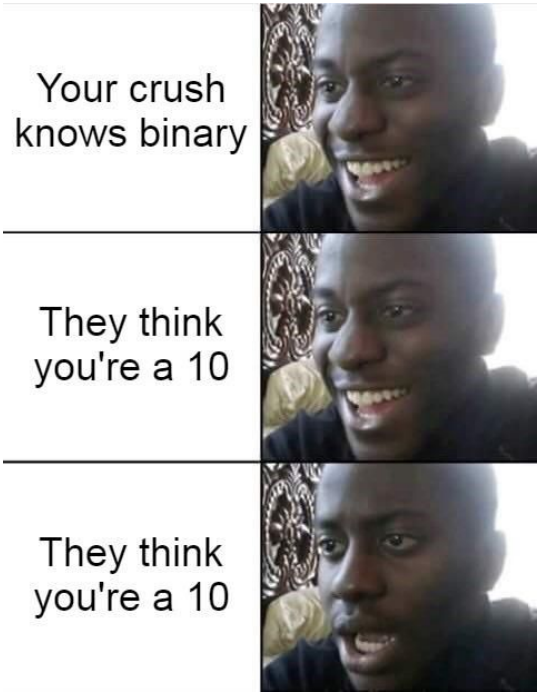
- Remember! For your final submission,export your Jupyter Notebook as a PDF and upload that to Gradescope
  - We will be grading the outputs (and looking over the code)
- Due date: Project 1 is **DUE Wednesday the OCT 13th before lecture**

me irl

Code Specifics

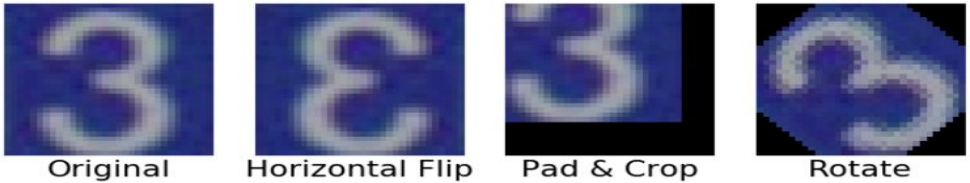
Let's Talk Project / Python Specifics

- Augmenting data
- Imputing data
- Data types and how to handle them:
  - Categorical data
  - Numeric Data (Scaling)
- Pipeline
- Splitting Data
- Interpreting Results



Augmenting Data

- Can augment Both Rows and Columns
  - (We're just augmenting Columns in Project 1)
  - Goal: Richer features and deeper/more balanced datasets
- Augmenting rows
  - Computer Vision Datasets
  - Timeseries
- Augmenting Columns
  - Log transformations
  - Feature/Label mismatch
    - Typical with time series data! (E.g., Data collected continuously, but labels at intervals)
  - Categorical transformations
    - E.g., converting lat/long to neighborhoods, or time-blocking



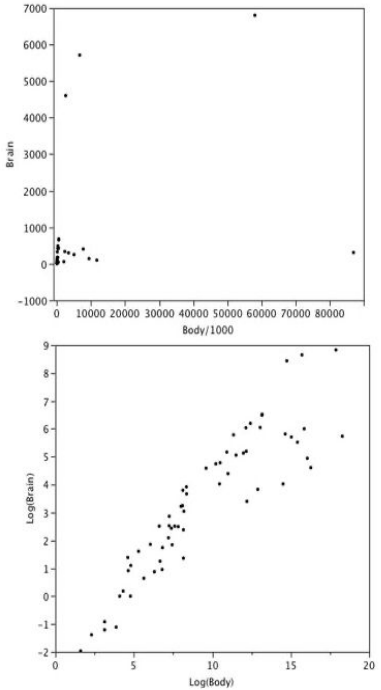
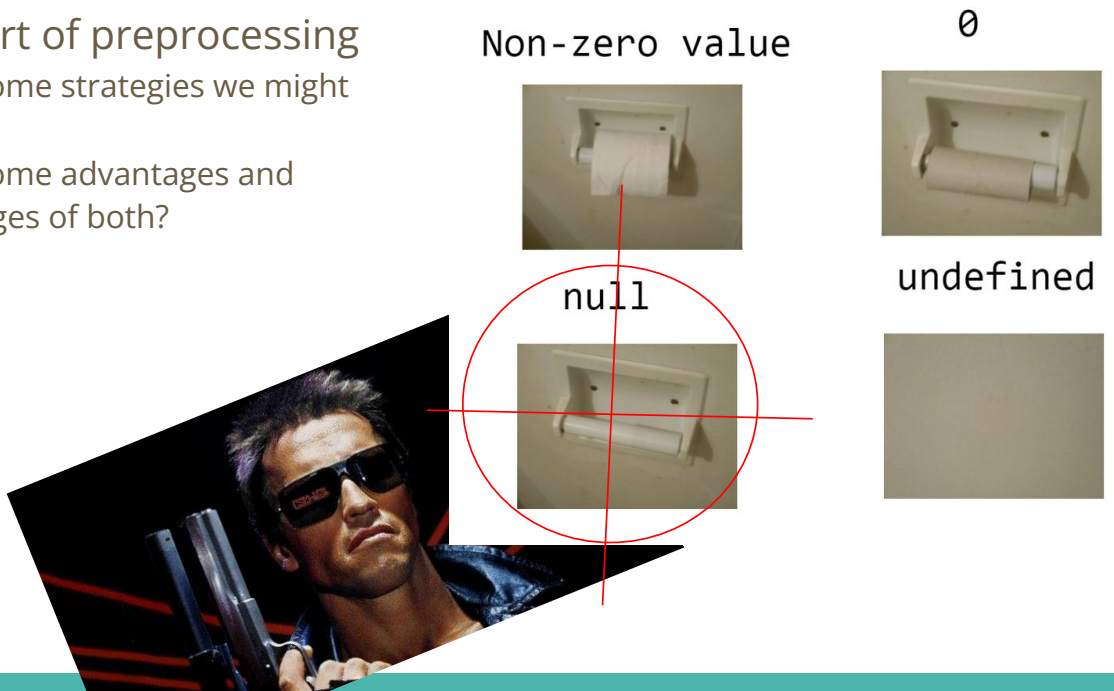


Figure 1. Scatter plots of brain weight as a function of body weight in terms of both raw data (upper panel) and log-transformed data (lower panel).

Imputing Data

- Need to resolve Null Values in dataset as part of preprocessing
  - What are some strategies we might take?
  - What are some advantages and disadvantages of both?



Dealing with Categorical Features

- Problem: many features will come in the form of categorical variables
- First problem: Converting strings to ints:
  - Solution: Label encoder
  - Data looks good! So can we stop there?
  - No! Multiple categories =/= ordinal ordering
  - How can we handle this?
    - One Hot Encoding!

	name	neighborhood	type	price
0	barneys beanery	westwood	alcohol	16
1	roccos	westwood	alcohol	18
2	sharetea	santa monica	boba	4
3	junpi	santa monica	boba	5
4	ichi	santa monica	boba	10
5	cabo cantina	brentwood	alcohol	6
6	bruhaus	brentwood	alcohol	11
7	qs billiards	brentwood	alcohol	25

In [51]:
```
# label encoding the data
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data['neighborhood']= le.fit_transform(data['neighborhood'])
data['type']= le.fit_transform(data['type'])
data.head(8)
```

Out[51]:

	neighborhood	type	price
0	2	0	16
1	2	0	18
2	1	1	4
3	1	1	5
4	1	1	10
5	0	0	6
6	0	0	11
7	0	0	25

One Hot Encoding

- Idea: Change each unique categorical value into its own binary column!
  - E.g, the values of westwood and brentwood each get their own column with 1 = present, and 0 = absent.


```
In [52]: from sklearn.compose import ColumnTransformer, make_column_transformer

In [53]: preprocess = make_column_transformer(
             (OneHotEncoder(categories='auto', sparse=False), ['neighborhood'])
         )

In [54]: preprocess.fit_transform(data)

Out[54]: array([[0., 0., 1.],
                [0., 0., 1.],
                [0., 1., 0.],
                [0., 1., 0.],
                [0., 1., 0.],
                [1., 0., 0.],
                [1., 0., 0.],
                [1., 0., 0.]])
```

	name	neighborhood	type	price
0	barneys beanery	westwood	alcohol	16
1	roccos	westwood	alcohol	18
2	sharetea	santa monica	boba	4
3	junpi	santa monica	boba	5
4	ichi	santa monica	boba	10
5	cabo cantina	brentwood	alcohol	6
6	bruhaus	brentwood	alcohol	11
7	qs billiards	brentwood	alcohol	25

Scaling/Normalizing Numeric Data

- Datasets often contain features that highly vary in magnitudes, units, and range.
- Normalisation should be performed when the scale of a feature is irrelevant or misleading and not performed when the scale is meaningful.
  - E.g., algorithms which use Euclidean Distance measure are sensitive to Magnitudes. Feature scaling helps to weigh all the features equally.
- Standard Scaler
  - The StandardScaler assumes your data is normally distributed within each feature and will scale them such that the distribution is now centred around 0, with a standard deviation of 1.
  - Other methods include MinMax (shrink data to a 0-1 range), and Normalizer (shrink to within 1 unit of cartesian coordinate space)

Scaling in Python

```
In [55]: from sklearn.compose import ColumnTransformer, make_column_transformer
         from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```
In [56]: preprocess = make_column_transformer(
             (StandardScaler(), ['price']),
         )
```

```
In [57]: preprocess.fit_transform(data)
```

```
Out[57]: array([[ 0.60259525],
                [ 0.89476265],
                [-1.15040912],
                [-1.00432542],
                [-0.27390693],
                [-0.85824173],
                [-0.12782324],
                [ 1.91734854]])
```

	name	neighborhood	type	price
0	barneys beanery	westwood	alcohol	16
1	roccos	westwood	alcohol	18
2	sharetea	santa monica	boba	4
3	junpi	santa monica	boba	5
4	ichi	santa monica	boba	10
5	cabo cantina	brentwood	alcohol	6
6	bruhaus	brentwood	alcohol	11
7	qs billiards	brentwood	alcohol	25

- Hmmm...doesn't this look familiar?

Yes! Now that's starting to look a bit like a pipeline!

```
In [70]: from sklearn.preprocessing import StandardScaler, OneHotEncoder
         from sklearn.compose import ColumnTransformer, make_column_transformer

In [71]: preprocess = make_column_transformer(
             (StandardScaler(), ['price']),
             (OneHotEncoder(categories='auto'), ['neighborhood', 'type'])
         )

In [72]: preprocess.fit_transform(data)

Out[72]: array([[ 0.60259525,  0.        ,  0.        ,  1.        ,  1.        ,
                  0.        ],
                [ 0.89476265,  0.        ,  0.        ,  1.        ,  1.        ,
                  0.        ],
                [-1.15040912,  0.        ,  1.        ,  0.        ,  0.        ,
                  1.        ],
                [-1.00432542,  0.        ,  1.        ,  0.        ,  0.        ,
                  1.        ],
                [-0.27390693,  0.        ,  1.        ,  0.        ,  0.        ,
                  1.        ],
                [-0.85824173,  1.        ,  0.        ,  0.        ,  1.        ,
                  0.        ],
                [-0.12782324,  1.        ,  0.        ,  0.        ,  1.        ,
                  0.        ],
                [ 1.91734854,  1.        ,  0.        ,  0.        ,  1.        ,
                  0.        ]])
```

	name	neighborhood	type	price
0	barneys beanery	westwood	alcohol	16
1	roccos	westwood	alcohol	18
2	sharetea	santa monica	boba	4
3	junpi	santa monica	boba	5
4	ichi	santa monica	boba	10
5	cabo cantina	brentwood	alcohol	6
6	bruhaus	brentwood	alcohol	11
7	qs billiards	brentwood	alcohol	25

What is this!?

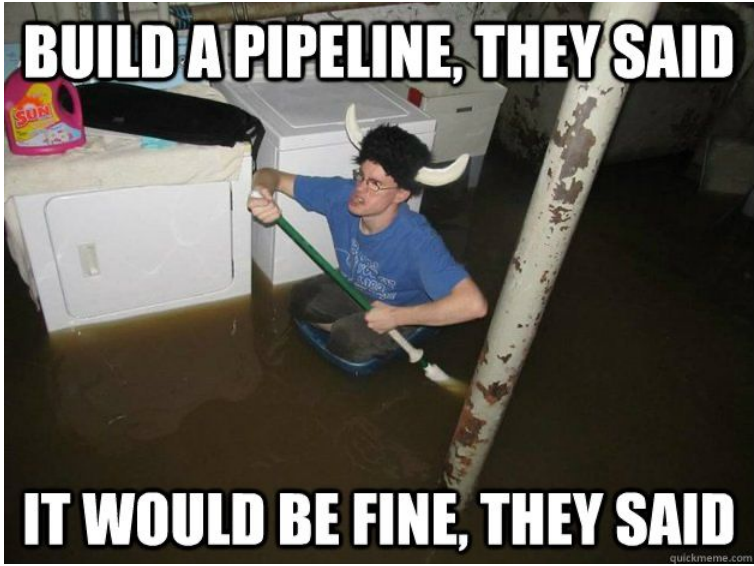
In [31]:

```
# This cell implements the complete pipeline for preparing the data
# using sklearns TransformerMixins
# Earlier we mentioned different types of features: categorical, and floats.
# In the case of floats we might want to convert them to categories.
# On the other hand categories in which are not already represented as integers must be mapped to integers before
# feeding to the model.

# Additionally, categorical values could either be represented as one-hot vectors or simple as normalized/unnormalized
# Here we encode them using one hot vectors.

# DO NOT WORRY IF YOU DO NOT UNDERSTAND ALL THE STEPS OF THIS PIPELINE. CONCEPTS LIKE NORMALIZATION,
# ONE-HOT ENCODING ETC. WILL ALL BE COVERED IN DISCUSSION

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.base import BaseEstimator, TransformerMixin


imputer = SimpleImputer(strategy="median") # use median imputation for missing values
housing_num = housing_unlabeled.drop('ocean_proximity', axis=1) # remove the categorical feature
# column index
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

#
class AugmentFeatures(BaseEstimator, TransformerMixin):
    '''
    implements the previous features we had defined
    housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
    housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
    housing["population_per_household"]=housing["population"]/housing["households"]
    '''
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = AugmentFeatures(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

housing_prepared = full_pipeline.fit_transform(housing_unlabeled)
```

Select a model and train

Pipelining in Datascience

- Doing datascience is a series of trials and errors
  - Trying out different feature sets, learning models, parameters etc.
- Once you've settled on an approach for processing your data, and successfully validated your model, the next steps is to migrate it to production for future data inputs with a standardized single function call.
  - In Project 1 once you've created your model that accurately predicts AirBNB rental prices, you might want to migrate the model into AirBNB's website.



Pulling everything together

- Define your strategy for imputing values
- Augment features as needed!
- Define scaling as needed
- Nestle numeric pipelines and categorical (if multiple steps)
- Full pipeline runs the processing steps for both our num and cat features and sticks them together!

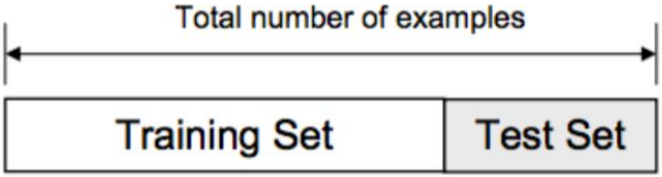
```
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', AugmentFeatures()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
numerical_features = list(housing_num)
categorical_features = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, numerical_features),
        ("cat", OneHotEncoder(), categorical_features),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Splitting a Dataset-Sometimes Breakups just happen...

- Train Test Split

```
# create training and testing vars
X_train, X_test, y_train, y_test = train_test_split(df, y,
test_size=0.2)
print X_train.shape, y_train.shape
print X_test.shape, y_test.shape


(353, 10) (353,)
(89, 10) (89,)
```

```
# fit a model
lm = linear_model.LinearRegression()


model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```



Interpreting Results

- How do we score? Have to choose our goalposts:
  - MSE vs. RMSE vs. MAE vs. R^2
    - Have to decide what you prioritize!
    - What score do you think is best for our airBNB problem?
- We've got a result, is it any good?
  - Almost always the answer is: 'It depends!'
    - A little easier with classification exercises, but not obvious there either!
  - Unlike stats we don't have industry-wide accepted outcomes (i.e., no p-value equivalents!)
  - Instead the quality of the result will heavily depend on specific problem
    - For our airBnb dataset, what do you think as acceptable error rate would be?