Feature processing V2:

I'm not getting quite where I want so I'm going to use sklearn's library to do some more intelligent binning.

https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features

Feature processing of the miscellaneous attributes. Need to do things like 1 hot encoding, etc.

https://www.kdnuggets.com/2020/07/easy-guide-data-preprocessing-python.html

https://www.udemy.com/course/machinelearning/learn/lecture/19039248#notes

https://colab.research.google.com/github/google/eng-edu/blob/master/ml/cc/exercises/linear_regression_with_a_real_dataset.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=linear_regression_real_tf2-colab&hl=en

REALLY GOOD: https://www.kaggle.com/dmilla/introduction-to-decision-trees-titanic-dataset#Preparing-the-Titanic-dataset

EVEN BETTER: https://towardsdatascience.com/understanding-feature-engineering-part-1-continuous-numeric-data-da4e47099a7b

```
# Imports needed for the script
import numpy as np
import pandas as pd
import re
import xgboost as xgb
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import preprocessing as P
from sklearn.impute import SimpleImputer

import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls

# For handling "thirteen-> 13"
!pip install word2number
from word2number import w2n
```

```
    Requirement already satisfied: word2number in /usr/local/lib/python3.6/dist-packages (1.1
```

```
# Data paths

MISC_TRAIN = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisio
MISC_TEST = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decision
MISC_EVAL = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decision
```

Load the data

```
# Loading the data
train = pd.read_csv(MISC_TRAIN)
test = pd.read_csv(MISC_TEST)
eval = pd.read_csv(MISC_EVAL)

# Merging the data
full_data = [train,test,eval]
full = pd.concat(full_data,keys=['train','test','eval'])
print(full.shape)

# Creating copy of this full dataset so I don't alter the original object
dataset = full.copy()

dataset.head(10)
```

```
(25000, 6)
```

		defendant_age	defendant_gender	num_victims	victim_genders	offence_category	o
train	**0**	62	female	1	male	theft	
	1	17	male	1	male	theft	
	2	not known	male	1	male	theft	
	3	not known	male	1	male	theft	
	4	52	male	1	female	theft	
	5	40	male	0	NaN	sexual	
	6	not known	male	1	female	theft	
	7	not known	female	1	male	theft	
	8	30	male	0	NaN	royalOffences	
	9	23	male	1	male	theft	

Functions to help process the age and the victim genders

```
# function for helping to process written numbers
def word_to_number(x):
  try:
    return w2n.word_to_num(x)
  except:
    # see if there is any number in phrase and return
    x = str(x)
    number = []
    num = ''
    for i in x:
      if i.isdigit():
```

```
      if i.isdigit():
        number.append(i)
    if len(number) > 0:
      return int(num.join(number))
    else:
      # Change any other results to big value that will be in its own bin
      x = float('nan')
      return x

def victims_processor(x):
  # takes in list of victim genders separated by ;
  # Split on ;
  x = str(x)
  x_split = x.split(';')
  # print(x_split)
  w_count = 0
  m_count = 0
  i_count = 0
  for gender in x_split:
    if re.search('.*f',gender):
      w_count += 1
    elif re.search('.*mal',gender):
      m_count += 1
    else:
      i_count += 1
  return np.argmax([m_count,w_count,i_count])
```

Processing age column

Here's the discretizer I'll use: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html#sklearn.preprocessing.KBinsDiscretizer

Psych! This doesn't handle NaN values.... maybe old way was fine?

Yes, used qcut instead.

```
# AGE

X_ages = dataset['defendant_age'].copy()
# Convert all ages to numbers
# print(X_ages.unique())
X_ages = X_ages.apply(word_to_number)
# print(X_ages.unique())
# print(X_ages.head(10))

# QCUT with 10 bins
bins = 10
# dataset = dataset.astype({'defendant_age' : 'category'})
X_ages_qcut = pd.qcut(X_ages,
                        q=bins,
                        precision=0)
```

```
                                precision=0)
# print(X_ages_qcut)
onehot_ages = pd.get_dummies(X_ages_qcut,dummy_na=True)
onehot_ages.head(10)
```

		(7.0, 17.0]	(17.0, 19.0]	(19.0, 21.0]	(21.0, 23.0]	(23.0, 26.0]	(26.0, 29.0]	(29.0, 33.0]	(33.0, 39.0]	(39.0, 47.0]	(47.0, 1112.0]	NaN
train	**0**	0	0	0	0	0	0	0	0	0	1	0
	1	1	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	1
	3	0	0	0	0	0	0	0	0	0	0	1
	4	0	0	0	0	0	0	0	0	0	1	0
	5	0	0	0	0	0	0	0	0	1	0	0
	6	0	0	0	0	0	0	0	0	0	0	1
	7	0	0	0	0	0	0	0	0	0	0	1
	8	0	0	0	0	0	0	1	0	0	0	0
	9	0	0	0	1	0	0	0	0	0	0	0

Processing gender column

Simple one hot encoding here ala get_dummies

```
print(dataset['defendant_gender'].value_counts())
one_hot_gender = pd.get_dummies(dataset['defendant_gender'],dummy_na=False)
one_hot_gender.head(10)
```

```
male              19035
```

Processing num victims

```
# QCUT with 2 bins
print(dataset['num_victims'].value_counts())
bins = 2
num_vic_qcut = pd.qcut(dataset['num_victims'],
                        q=bins,
                        precision=0)

one_hot_num_victims = pd.get_dummies(num_vic_qcut,dummy_na=False)
one_hot_num_victims.head(10)
```

```
1     19211
0      3553
2      1742
3       340
4        85
5        36
6        14
7         7
8         4
12        3
9         3
10        2
Name: num_victims, dtype: int64
```

		(-1.0, 1.0]	(1.0, 12.0]
train	**0**	1	0
	1	1	0
	2	1	0
	3	1	0
	4	1	0
	5	1	0
	6	1	0
	7	1	0
	8	1	0
	9	1	0

Processing victim genders

```
# Need to parse and process the victims
# print(dataset['victim_genders'].value_counts())
victim_gender = dataset['victim_genders'].apply(victims_processor)
one_hot_vic_gender = pd.get_dummies(victim_gender,dummy_na=False)
```

```
one_hot_vic_gender.columns = ['majority male','majority fem','majority indeterminate']
one_hot_vic_gender.head(10)
```

		majority male	majority fem	majority indeterminate
train	**0**	1	0	0
	1	1	0	0
	2	1	0	0
	3	1	0	0
	4	0	1	0
	5	0	0	1
	6	0	1	0
	7	1	0	0
	8	0	0	1
	9	1	0	0

Processing offense category

```
print(dataset['offence_category'].value_counts())
one_hot_offence = pd.get_dummies(dataset['offence_category'],dummy_na=False)
one_hot_offence.head(10)
```

```
theft              17703
deception           1745
royalOffences       1143
```

Processing offense subcategory

```
kill                 881
```

```
# print(dataset['offence_subcategory'].value_counts())
# one_hot_sub_offence = pd.get_dummies(dataset['offence_subcategory'],dummy_na=True)

# Might be cool to bin the infrequent ones into a bigger "infrequent" category
def subcat_processor(x):
  x = str(x)
  if re.search('grandLarceny|simpleLarceny|theftFromPlace|pocketpicking|stealingFromMaster|coi
    return x
  else:
    return 'various'

processed_subcats = dataset['offence_subcategory'].copy().apply(subcat_processor)
one_hot_sub_offence = pd.get_dummies(processed_subcats,dummy_na=False)
one_hot_sub_offence.head(10)
# processed_subcats.value_counts()
```

		animalTheft	burglary	coiningOffences	embezzlement	forgery	fraud	grandLarceny
train	**0**	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0
	8	0	0	1	0	0	0	0
	9	0	0	0	0	0	0	1

Combine into one dataframe

```
processed_df = pd.concat([onehot_ages,one_hot_gender,one_hot_num_victims,one_hot_vic_gender,on
processed_df.head()
```

		(7.0, 17.0]	(17.0, 19.0]	(19.0, 21.0]	(21.0, 23.0]	(23.0, 26.0]	(26.0, 29.0]	(29.0, 33.0]	(33.0, 39.0]	(39.0, 47.0]	(47.0, 1112.0]	Na
train	**0**	0	0	0	0	0	0	0	0	0	1	
	1	1	0	0	0	0	0	0	0	0	0	
	2	0	0	0	0	0	0	0	0	0	0	

Interactions

Create a few feature combinations that I think may be useful

Ideas:

1. old women may be less likely to be charge (but then again they may be very sparse)
2. maybe when defendant is male and victim is female
3. maybe when victim is female and crime is sexual

Maybe it would be easier to just let a neural net figure these things out for me?

```
# Ok lets look at male on female crimes

male_perp = processed_df['male'].copy().to_numpy()
fem_vict = processed_df['majority fem'].copy().to_numpy()
m_v_f = []
for i,sample in enumerate(male_perp):
  if sample == 1:
    if fem_vict[i] == sample:
      m_v_f.append(1)
    else:
      m_v_f.append(0)
  else:
    m_v_f.append(0)

processed_df['male vs fem'] = np.array(m_v_f)
processed_df.head(10)
```

		(7.0, 17.0]	(17.0, 19.0]	(19.0, 21.0]	(21.0, 23.0]	(23.0, 26.0]	(26.0, 29.0]	(29.0, 33.0]	(33.0, 39.0]	(39.0, 47.0]	(47.0, 1112.0]	Na
train	**0**	0	0	0	0	0	0	0	0	0	1	0

```
# female victim and crime is sexual

crime_sex = processed_df['sexual'].copy().to_numpy()

sex_crime_f = []
for i,sample in enumerate(crime_sex):
  if sample == 1:
    if fem_vict[i] == sample:
      sex_crime_f.append(1)
    else:
      sex_crime_f.append(0)
  else:
    sex_crime_f.append(0)

processed_df['sex crime fem'] = np.array(sex_crime_f)
processed_df.head(10)
```

		(7.0, 17.0]	(17.0, 19.0]	(19.0, 21.0]	(21.0, 23.0]	(23.0, 26.0]	(26.0, 29.0]	(29.0, 33.0]	(33.0, 39.0]	(39.0, 47.0]	(47.0, 1112.0]	Na
train	**0**	0	0	0	0	0	0	0	0	0	1	0
	1	1	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	1
	3	0	0	0	0	0	0	0	0	0	0	1
	4	0	0	0	0	0	0	0	0	0	1	0
	5	0	0	0	0	0	0	0	0	1	0	0
	6	0	0	0	0	0	0	0	0	0	0	1
	7	0	0	0	0	0	0	0	0	0	0	1
	8	0	0	0	0	0	0	1	0	0	0	0
	9	0	0	0	1	0	0	0	0	0	0	0

Load other datasets, tack on labels, and export as numpy array

```
# Split sets back apart on key
train = processed_df.loc['train']
test = processed_df.loc['test']
eval = processed_df.loc['eval']
```

```
# I'm only bothering with tf-df cause that's what performed the best last time

TRAINING_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-d
TESTING_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-de
EVAL_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decis

EVAL_IDS = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions
```

```
# Function for combining data
def load_data(file_path,data_to_append,file_name,glove=True):
    with open(file_path) as f:
      raw_data = [line.split() for line in f]
    # I want to convert this to a numpy array
    N = len(raw_data) # Num examples
    D = None
    if glove:
      D = len(raw_data[0]) # num dimensions (of first example...), need to adjust to hardcode
    else:
      D = 10001 # num dimensions for tfidf
    print("num_examples:",N,"num_dimensions:",D)
    np_data = np.zeros((N,D))
    # np_labels = np.zeros((N,))
    for index,instance in enumerate(raw_data):
      # Store label in numpy array
      label = int(instance[0])
      if label == 0:
        label = -1
      np_data[index][0] = label
      # Store data
      for dim,feat in enumerate(instance[1:]):
        feat_index = int(feat.split(":")[0])
        feat_value = float(feat.split(":")[1])
        np_data[index][feat_index] = feat_value
        # np_data[index][dim] = feat_value

    # Now, convert to pandas df
    df = pd.DataFrame(np_data)
    df_label = pd.DataFrame(np_data[:,0])
    # Add my data
    df = pd.concat([df,data_to_append],axis=1)
    # df = df.to_numpy()

    # Also, add labels to just the data_to_append
    df_misc = pd.concat([df_label,data_to_append],axis=1)
    # df_misc = df_misc.to_numpy()

    # print("labels shape:",np_labels.shape)
    print("intance shape:",df.shape)
    print("instance:",df)

    print("misc",df_misc.shape) # 52

    # Save file
    save_path = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-deci
    save_path2 = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-dec

    df.to_csv(path_or_buf=save_path,index=False)
    df_misc.to_csv(path_or_buf=save_path2, index=False)
```

```
# %time load_data(TRAINING_PATH_TF,train,'tf-train-2',glove=False)
# %time load_data(TESTING_PATH_TF,test,'tf-test-2',glove=False)
%time load_data(EVAL_PATH_TF,eval,'tf-eval-2',glove=False)
```

```
num_examples: 5250 num_dimensions: 10001
intance shape: (5250, 10052)
instance:         0         1         2  ...  wounding  male vs fem  sex crime fem
0     1.0  0.000000  0.000000  ...         0            0              0
1     1.0  0.037681  0.000000  ...         0            0              0
2     1.0  0.000000  0.000000  ...         0            0              0
3     1.0  0.000000  0.000000  ...         0            0              0
4     1.0  0.000000  0.000000  ...         0            0              0
...   ...       ...       ...  ...       ...          ...            ...
5245  1.0  0.000000  0.000000  ...         0            0              0
5246  1.0  0.008551  0.000000  ...         0            0              0
5247  1.0  0.000000  0.055636  ...         0            0              0
5248  1.0  0.000000  0.000000  ...         0            0              0
5249  1.0  0.148382  0.000000  ...         0            1              0

[5250 rows x 10052 columns]
misc (5250, 52)
```

Scraps

```
# X_ages = dataset['defendant_age'].copy()
# # Convert all ages to numbers
# print(X_ages.unique())
# X_ages = X_ages.apply(word_to_number)
# print(X_ages.unique())
# X_ages = X_ages.to_numpy().reshape(-1,1)

# # si = SimpleImputer(strategy='constant', fill_value='MISSING')
# # si.fit_transform(X_ages)

# # Now, let's bin intelligently using sklearn
# est = P.KBinsDiscretizer(n_bins=[15], strategy='uniform', encode='onehot-dense').fit(X_ages)
# binned_ages = est.transform(X_ages)
# print(binned_ages[0:10])
```

```
dataset = full.copy()
processed_data = []

# Let's go column by column and consider best treatment
# First is age '
# AGE
# Convert all ages to numbers
# print(dataset['defendant_age'].unique())
dataset['defendant_age'] = dataset['defendant_age'].apply(word_to_number)
# print(dataset['defendant_age'].unique())

# CUT allows us to select specific bin ranges, which may make sense, maybe
# https://pbpython.com/pandas-qcut-cut.html
# I want all kids under 16 to be own group...
cut_bins = [0,16,25,35,50,110]
dataset['defendant_age'] = pd.cut(dataset['defendant_age'],bins=cut_bins) # set labels=False t
one_hot_age = pd.get_dummies(dataset['defendant_age'],dummy_na=True)

# Now, gender
# print(dataset['defendant_gender'].unique())
# print(dataset['defendant_gender'].value_counts())
one_hot_gender = pd.get_dummies(dataset['defendant_gender'],dummy_na=True)

# Now, number of victims, want to bin these
# print(dataset['num_victims'].value_counts())
victim_bins = [-1,0,1,2,3,4,6,100]
victim_cut = pd.cut(dataset['num_victims'],bins=victim_bins)
# print(victim_cut)
one_hot_num_victims = pd.get_dummies(victim_cut,dummy_na=True)

# Now, victim genders, this will be tough
# Need function to process these, I want to return majority male or female or indeterminate (i
# Actually, let's just return 1 if there is a female in the victim or a 0 if not
victim_gender = dataset['victim_genders'].apply(victims_processor)
# print(victim_gender.value_counts())
# print(dataset['victim_genders'].value_counts())

# Now, offence_category - how to I make sure these are same across all datasets - think maybe
# Then separate at end?
# print(dataset['offence_category'].value_counts())
# print(dataset['offence_subcategory'].value_counts())
one_hot_offence = pd.get_dummies(dataset['offence_category'],dummy_na=True)
one_hot_sub_offence = pd.get_dummies(dataset['offence_subcategory'],dummy_na=True)


processed_df = pd.concat([one_hot_age,one_hot_gender,one_hot_num_victims,victim_gender,one_hot

print(processed_df.shape)

processed_df.head()
# processed_df.iloc[17498:17510]

# Split sets back apart on key
train_processed = processed_df.loc['train']
test_processed = processed_df.loc['test']
eval_processed = processed_df.loc['eval']

# print(train_processed.shape)
# print(test_processed.shape)
# print(eval_processed.shape)
```

```
eval_processed.head()
```

```
(25000, 84)
```

	(0.0, 16.0]	(16.0, 25.0]	(25.0, 35.0]	(35.0, 50.0]	(50.0, 110.0]	NaN	female	indeterminate	male	NaN	(-1.0, 0.0]	(0 1
0	0	0	0	0	0	1	0	0	1	0	0	
1	0	0	1	0	0	0	0	0	1	0	1	
2	0	0	0	0	0	1	1	0	0	0	0	
3	0	0	0	0	0	1	1	0	0	0	1	
4	0	0	0	0	0	1	0	0	1	0	0	

5 rows × 84 columns

```
# let's add the misc attributes to the 3 other NLP datasets

# Let's load them all
TRAINING_PATH = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-deci
TESTING_PATH = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decis
EVAL_PATH = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decision

TRAINING_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-d
TESTING_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-de
EVAL_PATH_TF = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decis

TRAINING_PATH_BAG = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-
TESTING_PATH_BAG = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-d
EVAL_PATH_BAG = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-deci

EVAL_IDS = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisions


def load_data(file_path,data_to_append,file_name,glove=True):
    with open(file_path) as f:
      raw_data = [line.split() for line in f]
    # I want to convert this to a numpy array
    N = len(raw_data) # Num examples
    D = None
    if glove:
      D = len(raw_data[0]) # num dimensions (of first example...), need to adjust to hardcode
    else:
      D = 10001 # num dimensions for tfidf
    print("num_examples:",N,"num_dimensions:",D)
    np_data = np.zeros((N,D))
    # np_labels = np.zeros((N,))
    for index,instance in enumerate(raw_data):
      # Store label in numpy array
      label = int(instance[0])
      if label == 0:
        label = -1
      np_data[index][0] = label
      # Store data
      # Store data
      for dim,feat in enumerate(instance[1:]):
        feat_index = int(feat.split(":")[0])
        feat_value = float(feat.split(":")[1])
        np_data[index][feat_index] = feat_value
        # np_data[index][dim] = feat_value

    # Now, convert to pandas df
    df = pd.DataFrame(np_data)
    # Add my data
    df = pd.concat([df,data_to_append],axis=1)

    # print("labels shape:",np_labels.shape)
    print("intance shape:",df.shape)

    # Save file
    save_path = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-deci

    df.to_csv(path_or_buf=save_path,index=False)

    return df


# Glove
# glove_train = load_data(TRAINING_PATH,train_processed,'glove_misc_train')
# load_data(TESTING_PATH,test_processed,'glove_misc_test')
# load_data(EVAL_PATH,eval_processed,'glove_misc_eval')

# TF-IDF
# # tf_train = load_data(TRAINING_PATH_TF,train_processed,'tf_misc_train',False)
# load_data(TESTING_PATH_TF,test_processed,'tf_misc_test',False)
# load_data(EVAL_PATH_TF,eval_processed,'tf_misc_eval',False)

# # BOW
# bow_train = load_data(TRAINING_PATH_BAG,train_processed,'bow_misc_train',False)
# load_data(TESTING_PATH_BAG,test_processed,'bow_misc_test',False)
# load_data(EVAL_PATH_BAG,eval_processed,'bow_misc_eval',False)
```

```
num_examples: 2250 num_dimensions: 10001
intance shape: (2250, 10085)
num_examples: 5250 num_dimensions: 10001
intance shape: (5250, 10085)
num_examples: 17500 num_dimensions: 10001
intance shape: (17500, 10085)
num_examples: 2250 num_dimensions: 10001
intance shape: (2250, 10085)
num_examples: 5250 num_dimensions: 10001
intance shape: (5250, 10085)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

```
# # Now, append labels to misc attributes, just the attributes themselves
# train_processed.insert(0,'label',training_labels)
# test_processed.insert(0,'label',testing_labels)
# eval_processed.insert(0,'label',eval_labels)

# train_processed.head()

# # Export as csv files
# save_train = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decis
# save_test = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisi
# save_eval = '/content/drive/My Drive/Colab Notebooks/Machine Learning 2020/old-bailey-decisi


# train_processed.to_csv(path_or_buf=save_train,index=False)
# test_processed.to_csv(path_or_buf=save_test,index=False)
# eval_processed.to_csv(path_or_buf=save_eval,index=False)
```

```
# Ok now, let's do some processing via: https://www.kaggle.com/dmilla/introduction-to-decision

# Copy original dataset in case we need it later when digging into interesting features
# WARNING: Beware of actually copying the dataframe instead of just referencing it
# "original_train = train" will create a reference to the train variable (changes in 'train' w
train = pd.read_csv(MISC_TRAIN)
original_train = train.copy() # Using 'copy()' allows to clone the dataset, creating a differe

# Feature engineering steps taken from Sina and Anisotropic, with minor changes to avoid warni
full_data = [train] # will add test and eval here later
# full_data = full
processed_data = []

# Let's go column by column and consider best treatment
# First is age '

# function for helping to process written numbers
def word_to_number(x):
  try:
    return w2n.word_to_num(x)
  except:
    # see if there is any number in phrase and return
    x = str(x)
    number = []
    num = ''
    for i in x:
      if i.isdigit():
        number.append(i)
    if len(number) > 0:
      return int(num.join(number))
    else:
      # Change any other results to NaN
      x = float("nan")
      return x

def victims_processor(x):
  # takes in list of victim genders separated by ;
  # Split on ;
  x = str(x)
  x_split = x.split(';')
  # print(x_split)
  for gender in x_split:
    if gender=='female':
      return 1
  return 0

# AGE
for idx,dataset in enumerate(full_data):
    # Convert all ages to numbers
    # print(dataset['defendant_age'].unique())
    dataset['defendant_age'] = dataset['defendant_age'].apply(word_to_number)
    # print(dataset['defendant_age'].unique())

    # Print average and std of age
    # age_avg = dataset['defendant_age'].mean()
    # age_std = dataset['defendant_age'].std()
    # print("average age:",age_avg,"age std:",age_std)
    # print(dataset['defendant_age'].describe())

    # Now I want to bin these values
    # Mapping Age
    # dataset.loc[ dataset['defendant_age'].isna(), 'defendant_age'] = 0
    # dataset.loc[(dataset['defendant_age'] > 0) & (dataset['defendant_age'] <= 16), 'defendan
    # dataset.loc[(dataset['defendant_age'] > 16) & (dataset['defendant_age'] <= 32), 'defenda
    # dataset.loc[(dataset['defendant_age'] > 32) & (dataset['defendant_age'] <= 48), 'defenda
    # dataset.loc[(dataset['defendant_age'] > 48) & (dataset['defendant_age'] <= 64), 'defenda
    # dataset.loc[ dataset['defendant_age'] > 64, 'defendant_age'] = 5
    # dataset = dataset.astype({'defendant_age': 'int32'}) # need to assign to new variable
    # print(dataset['defendant_age'].head(5))

    # Can potentially do this better with "category" data type... or qcut
    # bins = 4
    # QCUT automatically orginizes bins
    # View bin ranges
    # dataset = dataset.astype({'defendant_age' : 'category'})
    # my_bins = pd.qcut(dataset['defendant_age'],
    #                          q=bins,
    #                         precision=0)
    # print(my_bins.value_counts())
    # dataset['defendant_age'] = pd.qcut(dataset['defendant_age'],
    #                         q=bins,
    #                         labels=False,
    #                         precision=0)
    # CUT allows us to select specific bin ranges, which may make sense, maybe
    # https://pbpython.com/pandas-qcut-cut.html
    # I want all kids under 16 to be own group...
    cut_bins = [0,16,25,35,50,110]
    dataset['defendant_age'] = pd.cut(dataset['defendant_age'],bins=cut_bins) # set labels=Fal
    # cut_labels = pd.cut(dataset['defendant_age'],bins=cut_bins)
    # then, set NaN's, or can specify in get dummys
    # dataset.loc[ dataset['defendant_age'].isna(), 'defendant_age'] = len(cut_bins)-1
    # dataset = dataset.astype({'defendant_age' : 'int64'})
    # print(dataset['defendant_age'].head(10))
    # print(dataset['defendant_age'].value_counts())
    # print(dataset.head())
    # Now convert to one hot encodiing
    # get_dummies
    # Pass ranges as labels?
    one_hot_age = pd.get_dummies(dataset['defendant_age'],dummy_na=True)


    # Now, gender
    # print(dataset['defendant_gender'].unique())
    # print(dataset['defendant_gender'].value_counts())
    one_hot_gender = pd.get_dummies(dataset['defendant_gender'])

    # Now, number of victims, want to bin these
    # print(dataset['num_victims'].value_counts())
    victim_bins = [-1,0,1,2,3,4,6,100]
    victim_cut = pd.cut(dataset['num_victims'],bins=victim_bins)
    # print(victim_cut)
    one_hot_num_victims = pd.get_dummies(victim_cut,dummy_na=True)

    # Now, victim genders, this will be tough
    # Need function to process these, I want to return majority male or female or indeterminat
    # Actually, let's just return 1 if there is a female in the victim or a 0 if not
    victim_gender = dataset['victim_genders'].apply(victims_processor)
    # print(victim_gender.value_counts())
    # print(dataset['victim_genders'].value_counts())

    # Now, offence_category - how to I make sure these are same across all datasets - think ma
    # Then separate at end?
    print(dataset['offence_category'].value_counts())
    print(dataset['offence_subcategory'].value_counts())

    processed_df = pd.concat([one_hot_age,one_hot_gender,one_hot_num_victims,victim_gender],ax
    # Assign dataset
    processed_data.append(processed_df)

# for dataset in full_data:
#   print(dataset['defendant_gender'].unique())
# print(train.head(10))
train_processed = processed_data[0]
train_processed = processed_data[0]
train_processed.head(10)
```

```
    deception        1140
    royalOffences     787
```

```
# ENCODING CATEGORICAL DATA - I'm using scikit learn here.... hope that's ok, double check
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthroug
features = np.array(ct.fit_transform(features))
```

```
    other                              547
    wounding                           530
    forgery                            510
    embezzlement                       496
    fraud                              477
    animalTheft                        369
    highwayRobbery                     321
    robbery                            308
    housebreaking                      279
    manslaughter                       271
    murder                             258
    shoplifting                        255
    rape                               240
    bigamy                             220
    receiving                          212
    perjury                            206
    mail                               122
    pettyLarceny                       104
    assault                             94
    sodomy                              88
    assaultWithIntent                   68
    arson                               67
    concealingABirth                    66
    libel                               65
```