**Assignment 2**

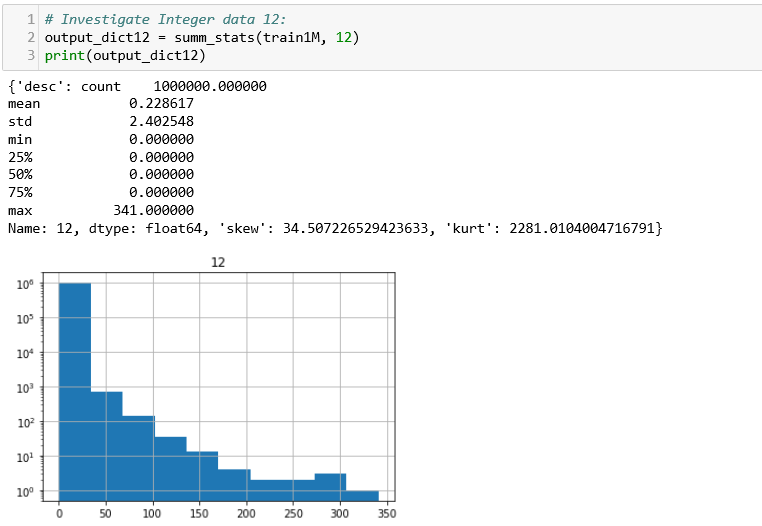
**Question 2**

2.1 Please refer to assign2.py

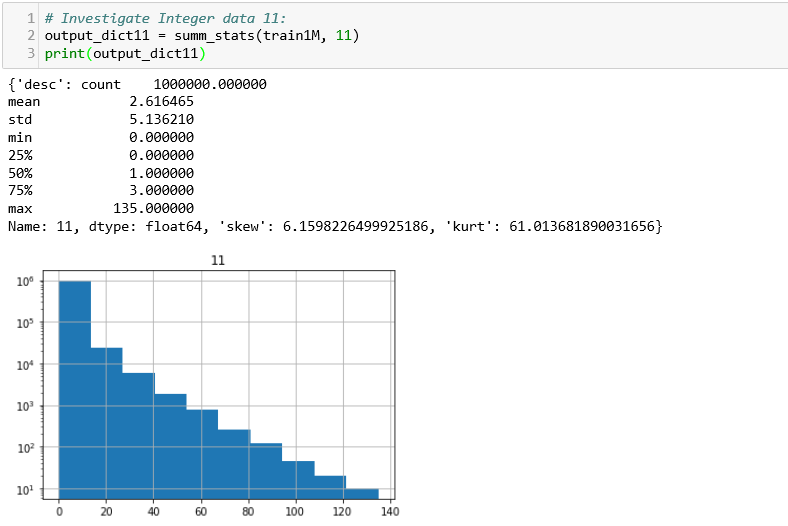
**2.2 Feature Summary Statistics**

Integer Feature Histograms:

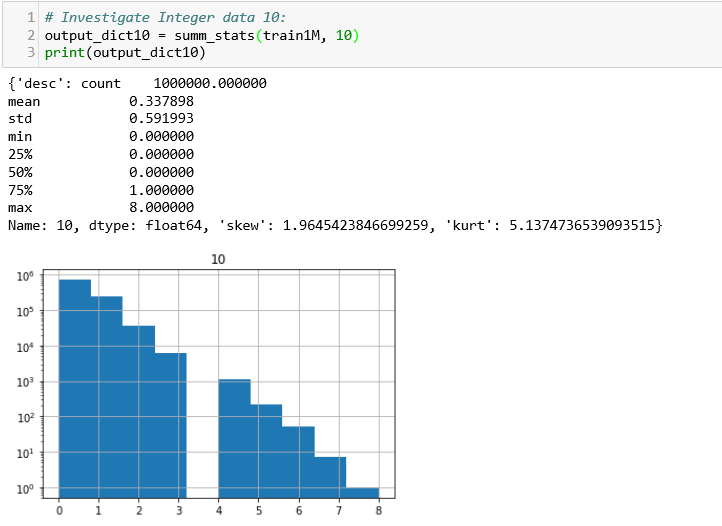
**Integer 12**



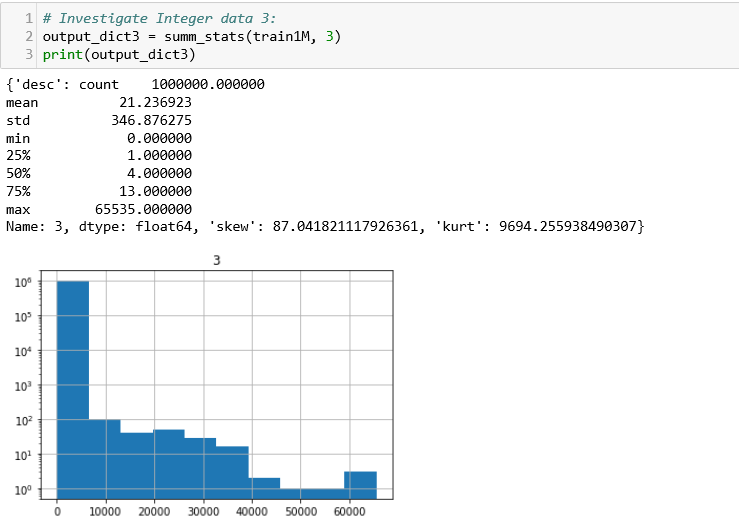
**Integer 11**



**Integer 10**

****

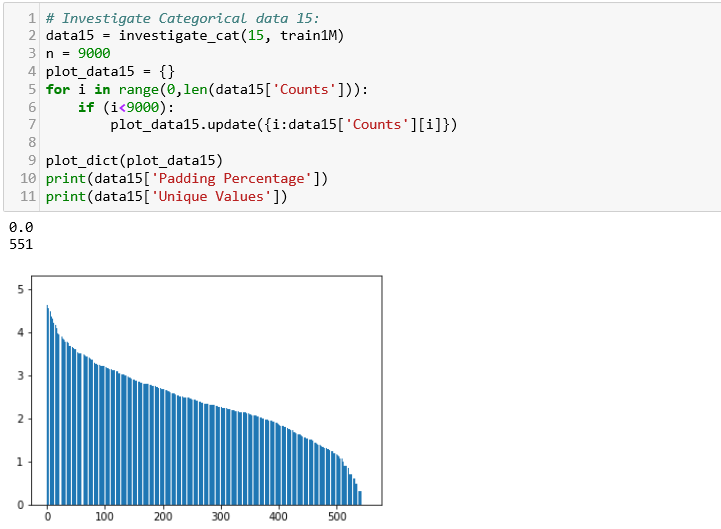
**Integer 3**

****

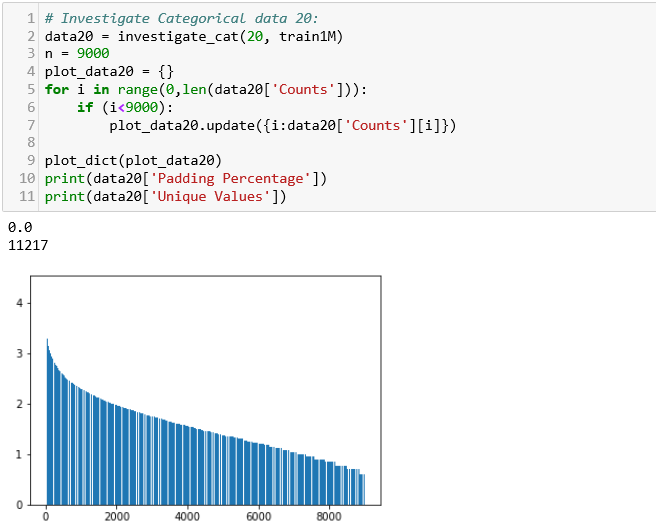
For the full syntax of the summ\_stats method please refer to the appendix.

Categorical Feature Histograms:

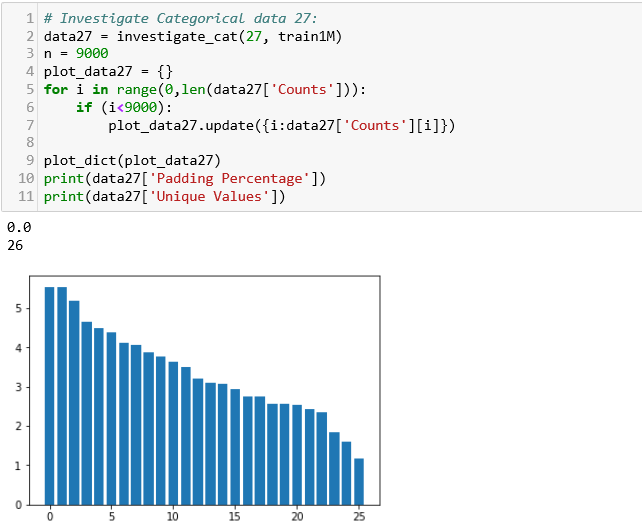
**Categorical Feature 15**



**Categorical Feature 20**

****

**Categorical Feature 27**



**Categorical Feature 30**



Where the full syntax of investigate\_cat and plot\_dict are available in the appendix.

**2.3 Feature Selection and Encoding**

When considering categorical features, I observed the following statistics:

* Number of missing values (padded as zeros in the preprocessing)
* Number of unique values the categorical feature can take

Features were dropped if *any* values were missing *or* if the number of unique values the feature can take exceeded 800. Since the intention was to use one hot encoding with 20 bits, we want to minimize information lost in the encoding process. With features with a high number of unique values, only the top 20 of feature values will be encoded, with the remainder grouped as a separate category. I believe this results in such high loss of information that it is not worth including in the encoding process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Categorical Feature Column Number (zero start)** | **Missing Values %** | **Unique Feature Values** | **Select/Reject** | **Select Reject Reason** |
| 14 | 0 | 1249 | Reject | High unique values |
| 15 | 0 | 551 | Select | Low missing values, low unique values |
| 16 | 3.3853 | 362872 | Reject | High unique values |
| 17 | 3.3853 | 141168 | Reject | High unique values |
| 18 | 0 | 274 | Select | Low missing values, low unique values |
| 19 | 12.159 | 16 | Reject | High missing values |
| 20 | 0 | 11217 | Reject | High unique values |
| 21 | 0 | 557 | Select | Low missing values, low unique values |
| 22 | 0 | 3 | Select | Low missing values, low unique values |
| 23 | 0 | 31849 | Reject | High unique values |
| 24 | 0 | 4916 | Reject | High unique values |
| 25 | 3.3853 | 322871 | Reject | High unique values |
| 26 | 0 | 3154 | Reject | High unique values |
| 27 | 0 | 26 | Select | Low missing values, low unique values |
| 28 | 0 | 9516 | Reject | High unique values |
| 29 | 3.3853 | 246473 | Reject | High unique values |
| 30 | 0 | 10 | Select | Low missing values, low unique values |
| 31 | 0 | 4093 | Reject | High unique values |
| 32 | 43.9476 | 1855 | Reject | High missing values, high unique values |
| 33 | 43.9476 | 4 | Reject | High missing values |
| 34 | 3.3853 | 291000 | Reject | High unique values |
| 35 | 76.2392 | 16 | Reject | High missing values |
| 36 | 0 | 15 | Select | Low missing values, low unique values |
| 37 | 3.3853 | 44970 | Reject | High unique values |
| 38 | 43.9476 | 73 | Reject | High missing values |
| 39 | 43.9476 | 32484 | Reject | High missing values, high unique values |

Upon observation it is also interesting to see that some features have equal numbers of missing values (one group highlighted in yellow, the other in orange). These features also beckon a rejection decision based on the possibility of high correlation.

This results in the resulting categorical feature set of columns: [14, 17, 20, 21, 26, 29, 35] for the training data (with zero start counting of columns). This is for training data *excluding* the training target.

For the categorical features selected above, I take the 20 most occurring feature values and any other values I group into a feature value of ‘Others’. Once I reduce the number of values the feature can take to these 21 values (the 20 top most occurring plus ‘Others’), I conduct label encoding into integers using LabelEncoder and subsequently apply a fit transform to encode each feature value into 21-bit lists. This information will be stored in the output of the preprocess\_cat\_data function. To find the *encoded* values from the fit transform, we can use the.toarray() function.

**Appendix: Calculating Summary Statistics and Plotting Histograms of Integer Features**

def summ\_stats(input\_data, col\_ind):

output\_dict = {}

desc = input\_data[col\_ind].describe()

skew = input\_data[col\_ind].skew(axis=0)

kurt = input\_data[col\_ind].kurtosis(axis=0)

output\_dict['desc'] = desc

output\_dict['skew'] = skew

output\_dict['kurt'] = kurt

%matplotlib inline

train1M.hist(column=col\_ind, log=True)

return output\_dict

**Appendix: Investigating Categorical Features**

def investigate\_cat(ind, data):

import matplotlib.pyplot as plt

cat\_kc = data[ind].value\_counts()

cat\_ind = cat\_kc.index.tolist()

cat\_count = np.log10(cat\_kc.values.tolist())

count\_dict = {}

for i in range(0,len(cat\_ind)):# if len(cat\_ind)<9000 else 9000)):

count\_dict.update({i:cat\_count[i]})

output\_dict = {}

if 0 in cat\_ind:

missing\_vals = cat\_kc.values[cat\_ind.index(0)]

else:

missing\_vals = 0

pct\_padding = missing\_vals/len(data)

output\_dict['Counts'] = count\_dict

output\_dict['Value Counts'] = cat\_kc

output\_dict['Padding Percentage'] = pct\_padding

output\_dict['Unique Values'] = len(cat\_ind)

return output\_dict

**Appendix: Plotting Categorical Features**

def plot\_dict(input\_dict):

%matplotlib inline

import matplotlib.pyplot as plt

lists = sorted(input\_dict.items())

x,y = zip(\*lists)

plt.bar(x,y)