**Approaches Followed while cleaning the 3gb of data**

Using **UltraEdit**, merged columns\_header and hit\_data file.

Using **delimit** tool, i converted the text file to CSV file and uploaded it into panda

**1st Approach** - Tried to load the data and read it

import csv

import pandas as pd

import numpy as np

import io

import pandas as pd

from sqlalchemy import create\_engine

x=pd.read\_csv("3gb\_test.csv",delimiter="\t",encoding='latin1',nrows=300000,dtype=str)

**Result:** I Can read maximum of 2 lakhs rows at a time after that I am getting memory error, as my machine doesn’t have enough space to load entire data in panda’s data frame.

**2nd Approach** – Since pandas cannot handle too much of data I tried to use pythons module dask

A Dask DataFrame is a large parallel dataframe composed of many smaller Pandas dataframes, split along the index.

Using this we can analyze data in the CSV without loading the entire CSV file into memory

It is similar to pandas dataframe but rather than loading the data into pandas, this approach keeps the dataframe as a sort of ‘pointer’ to the data file and doesn’t load anything until you specifically tell it to do so.

import dask.dataframe as dd

filename = '311\_Service\_Requests.csv' - # setup your dataframe so we can analyze the

311\_Service\_Requests.csv file. This file is assumed to be stored in the directory that you are working in

df = dd.read\_csv("3gb\_test.csv",delimiter="\t",encoding='latin1', dtype='str')

df.head(13) – shows only 13 rows \* 971 columns

I was trying to perform some computation operation

x=df[df.browser\_height=='640'] -------------------------------------------

x.browser\_height.count() # **Gives the count of total count of rows that have the height = 640**

x.compute() -----------------------------------------------------------------------

I was going through this doc - > <http://dask.pydata.org/en/latest/dataframe.html>

dusk can perform these operations –

* **Trivially parallelizable operations (fast):**
  + Elementwise operations: df.x + df.y, df \* df
  + Row-wise selections: df[df.x > 0]
  + Loc: df.loc[4.0:10.5]
  + Common aggregations: df.x.max(), df.max()
  + Is in: df[df.x.isin([1, 2, 3])]
  + Datetime/string accessors: df.timestamp.month
* **Cleverly parallelizable operations (fast):**
  + groupby-aggregate (with common aggregations): df.groupby(df.x).y.max(),df.groupby('x').max()
  + groupby-apply on index: df.groupby(['idx', 'x']).apply(myfunc), where idx is the index level name
  + value\_counts: df.x.value\_counts()
  + Drop duplicates: df.x.drop\_duplicates()
  + Join on index: dd.merge(df1, df2, left\_index=True, right\_index=True) or dd.merge(df1, df2, on=['idx', 'x']) where idx is the index name for both df1 and df2
  + Join with Pandas DataFrames: dd.merge(df1, df2, on='id')
  + Elementwise operations with different partitions / divisions: df1.x + df2.y
  + Datetime resampling: df.resample(...)
  + Rolling averages: df.rolling(...)
  + Pearson Correlations: df[['col1', 'col2']].corr()

**3rd Approach**

While doing research, I got to know that rather than loading entire data in dataframe which is giving us Memory error, we can read the data in chunks and then append the chunks to get the entire data.

Haven’t tried this, but it might work.

URL - <https://stackoverflow.com/questions/25962114/how-to-read-a-6-gb-csv-file-with-pandas>

**4th Approach**

Using Sqlite -

1. Create sqlite database
2. Iterate through the CSV file in chunks and store the data into sqllite
3. We can use pandas ‘sql’ tools to pull data from the database without worrying about memory constraints.