Interactive Visualizations of Mobile Device Market Leaders

***Abstract*—The goal of this assignment is to extract meaningful information from a dataset through a interactive visualization method. In our case, our dataset is about various mobile devices and their parent companies which competed against each other during the early 90s till the late 00s. We used Python libraries Plotly and Dash to extract valuable information to assist us in answering our group assignment questions i.e. we used our dashboard as a tool in creating different visualisations to investigate which devices or companies are the most successful in leading the market.**

I. Introduction

Throughout recent years, mobile devices have exploded in popularity and become essential to modern style living. Not only are they a key communication tool in the private, public, and business sectors, but they also provide entertainment in the form of social media, video streaming and mobile games. For many people, mobile devices are their first piece of technology providing them access to the internet, connecting even more people to the internet.

Mobile devices are not only profitable at the initial sale of the device, but by locking the user into the manufacturers software, companies can gain further long term profits through the collection of user data, advertising and app-store sales. Furthermore, by locking the customer into the company's ecosystem, further products can be sold which are designed to work particularly well with other products within that ecosystem (e.g. The Apple watch is designed to work particularly well with the iPhone). Therefore many companies compete in the mobile market by introducing ground breaking features (e.g. increased hardware performance), which later get adopted by competing devices once the technology becomes cheaper to implement. Other companies try to stand out by introducing novel form factors (e.g. larger screens), which tend to form new markets and new categories of mobile devices.

In this report we introduce an interactive visualisation of various mobile device parameters and their parent companies. We identify which devices and companies tried to create and lead new markets, and which ones were successful at doing so. We present our results in dashboard forms shown in figure 1.

II. Exploratory Data Analysis

A. Dataset

The main spreadsheet in our data set primarily includes mobile device names (nominal), release date and year (ordinal) and 12 ratio type attributes which can be split into two categories:

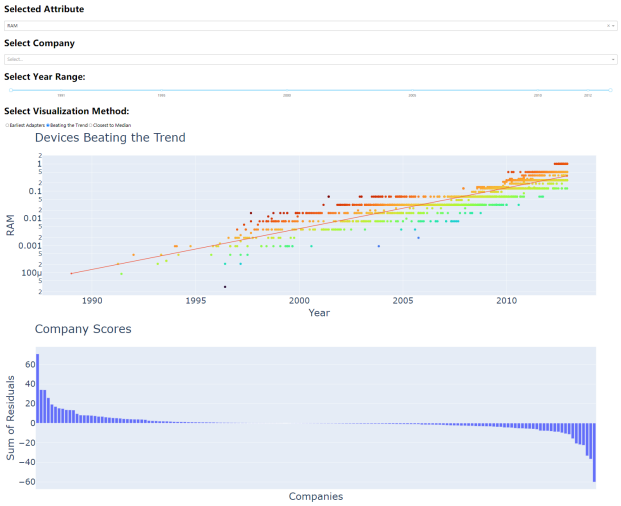


Figure 1: Main View of the Dash App

- **Performance Based Attributes** - RAM, CPU, Storage, Pixel Density - *A larger values is more desirable to consumers*
- **Dimensional Attributes** - Screen Width, Length and Diagonal, Display Diagonal, Length and Width, Volume and Mass - *A larger value is not necessarily more desirable to consumers*

B. Scatter Plot with Respect to the Attributes

To find trends or seasonality over time, we provide scatter plots of the feature data with respect to time. Some interesting pattern can be seen in the following scatter plot (Figure 3): RAM Capacity, Storage, CPU Clock and Pixel Density. These plots show horizontal periodic trend lay up along the y-axis. Such period can be explained by innovation diffusion theory and each period corresponds to an innovation diffusion period [3].

Diffusion of Innovation Theory states that *over time, an idea or product gains momentum and diffuses (or spreads) through a specific population or social system [4]*. When plotting the performance based attributes over time, we find a horizontal periodic trend lying along the y-axis which correspond to such innovation diffusion periods.

Businesses who implement new features early in a diffusion period are considered to be innovators and leading the market trend, and once the technology cheapens, the late majority will adapt those hardware features into their mobile device. This theory is depicted in figure 2.

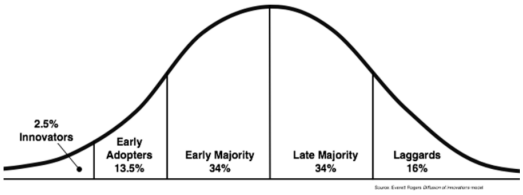


Figure 2: Innovation of Diffusion Plot [4]

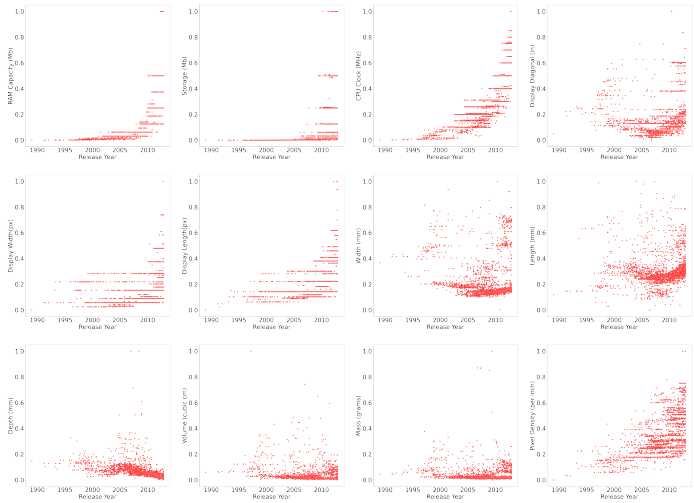


Figure 3: Scatter Plot with respect to the Features

C. Density Plot with Respect to the Features

Positively skewed distribution can be observed on width, length, depth, volume and mass features on Figure 4 . This is due to devices such as tablets, which have significantly longer, wider and thicker measurement compared to an average mobile phone device. A density plot of these five features reveals a normal distribution, and the median of these attributes represents the most popular dimensions that customers have for mobile devices.

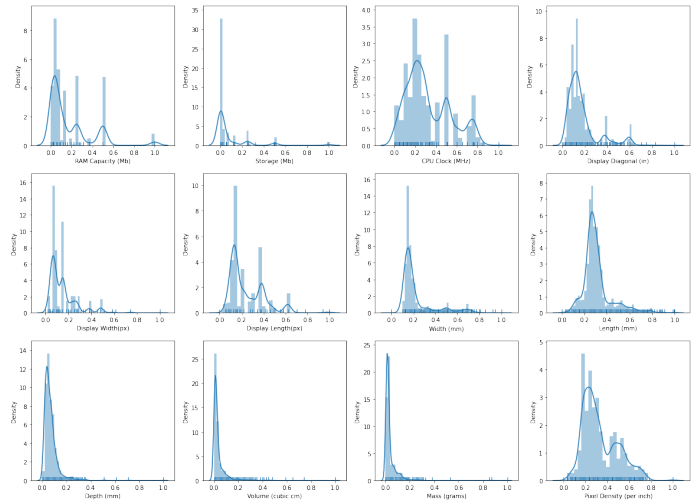


Figure 4: Density Plot in Respect to the Features

III. Data Preprocessing

The data was provided to us as a spreadsheet, with many of the attributes already been normalised to a range between 0 and 1. Although the company name for some devices were provided, some were missing and so the company name was extracted from mobile device name as well.

Simple scatter plots provided in figure 3 provides a first look at each attribute with respect to time. It is clear that many performance based attributes follow an exponential trend, which agrees with Moore's law claiming that these attributes double approximately every two years. As such, the axis for these attributes in our analysis will be arranged on a log scale. To avoid taking the log of 0 (which approaches to $-\infty$), we set all zero values to the next lowest value for each attribute to be log transformed.

The dataset contains 213 companies that produced different mobile devices from 1989 to 2012.There are a total of 3162 mobile device models and the first 259 models have missing company ID. To fix the missing first 259 mobile device models, we extracted the first word from the feature "Model" from the first 259 models and assigned the first word as the company name. The Model ID can be fetched from the second worksheet of the original dataset. We reassigned all 3162 mobile devices with new ID since some of the companies were taken over or merged with other companies. Additional interactive tools are provided Plotly. Including the ability to zoom in through click and drag on a subset of the visualisation.

IV. Interactive and Visual Variables

The interactive visualisation of the data was created in python with the use of Plotly and Dash libraries, which is shown in figure 1. The dashboard provides two drop-down menus for selecting the attribute that the user may wish to visualize, and an optional drop-down for selecting particular companies to focus on. The user may begin typing a company name in the company select field, and a list of company names that contain the typed in string will be provided as drop-down options. Multiple companies can be selected at a time in this way. A range slider is also provided which lets the user specify the time range to focus on.

A scatter plot is provided for visualising how various phone models compare with respect to the selected attribute, and a bar chart is provided for visualizing how successful companies are at beating the market with respect to the attribute. The exact behaviour of both these plots depend on the user's selected visualization method, which include earliest adopters, beating the trend and closest to median, which can be chosen by the use of radio buttons. Each company is given a score which is calculated by the sum of it's mobile device's scores, and is represented by the height of their corresponding bars in the bar chart. The bars in the bar chart are also sorted from left to right based on the company scores, which lets the user quickly identify the most successful company, and where each company stands in the ranking.

Whenever the user hovers their mouse over a point in the scatter plot, or over a bar in the bar chart, a hover label appears

Figure 5: Visual elements and KPI from dashboard

containing the corresponding mobile device or company name. This allows the user to quickly identify the device or company that the user is looking at. In the scatter plot, the cursor must be placed directly above the data point for the hover label to appear. In the bar chart, the user may place the cursor anywhere on the plot, and a label along with a dashed vertical line corresponding to the closest bar in the horizontal direction will appear, this allows for easier company selection especially when a bar is particularly small.

When no company is selected, all scatter points will be color coded depending on the chosen visualisation method , and all companies in the bar chart will be highlighted in dark blue. When at least one company is selected, only scatter points corresponding to the selected company will be highlighted by a dark blue-red color scheme, meanwhile all non-selected scatter points will be reduced in size, and their color set to light blue which blends in well with the plot background as seen in the first subplot in Figure 5. Additionally, the hover labels for non-selected scatter points will be disabled, and non-selected company's bars will turn light blue.

V. KGI and KPIs

Our Key Goal Indicator (KGI) is how successful companies and mobile devices are at leading the market. To measure this KGI, we provide three Key Performance Indicators (KPI), two of which are based on hardware attributes (*Beating the Trend* and *Earliest Adapters*). Meanwhile *Closest to Median* is based on dimensional attributes.

A. Performance Based KPIs

For performance based attributes, it is clear that a higher value is more desirable, and these features tend to follow the diffusion of innovation theory. Our KPI *Earliest Adopters* gives each mobile device a score for being the very first to implement a new or improved feature in the corresponding innovation diffusion period. Meanwhile *Beating the Trend* gives each mobile device a score based on where they stand in the innovation diffusion period.

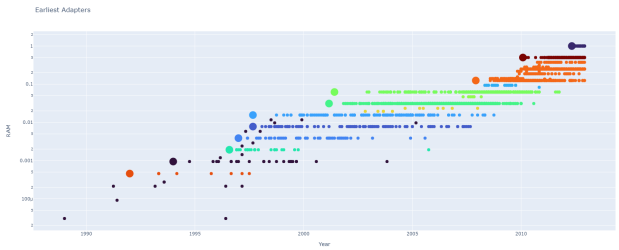


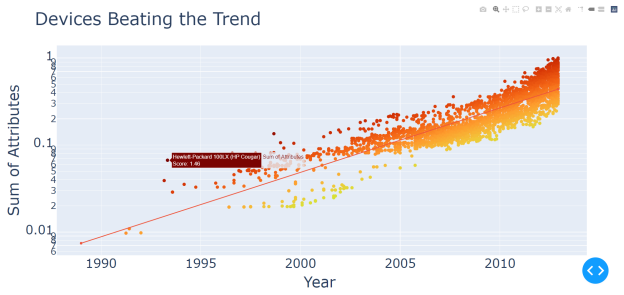
Figure 6: An example of best device in different innovation diffusion period

1) Beating the Trend: A linear fit is calculated for the selected attribute which is displayed in the scatter plot and each mobile device is given a score based on the residuals of the linear fit, then color coded accordingly. The companies are scored by the sum of all their device's scores. An additional attribute named *sum of attributes* is provided, which is a normalized score of the sum of all performance based attributes, which is used for identifying how mobile devices compare with respect to all performance attributes. The market leading device is represented by the scatter point having the most dark red color, and the hove label can be used to identify the name and exact score of that particular device.

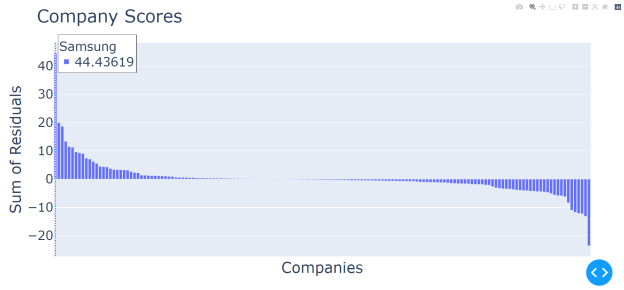
2) Earliest Adapters: When plotting the performance based attributes, it is frequently seen that mobile devices are clustered around discrete steps. This happens because mobile device manufacturers often follow standards such as RAM capacity doubling at each new generation, or manufacturers using the same hardware components. Each discrete step can be considered as a new market, and the first device in each step is considered to be the market leader.

To calculate a score based on earliest adapters, we applied the clustering algorithm DBSCAN [6] in one dimension of the chosen attributes to identify different innovation diffusion periods. This is shown in Figure 6. Each cluster is color coded in our scatter plot, and the earliest device in each cluster is marked by a larger scatter dot, which can be considered as the leader of the corresponding innovation diffusion period. A successful innovation can be identified by the number of devices in that given cluster, however it is too early to tell the success of the latest innovations in this way.

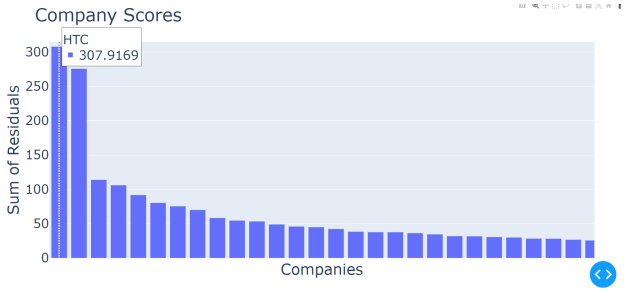
3) Closest to Median: For most of the attributes corresponding to the size and dimensions of the mobile device, a larger attribute value is not necessarily better. For these attributes we provide the *closest to median* KPI. We decided that median should be selected over mean to overcome any outlier influences. We decided to include Display Diagonal, Display Width and Display Length in this method rather than Earliest Adopter Method even though these features show a clear trend in their respective time series plots. For dimensional attributes such as size and shape of the mobile device, we can not assume that a larger value is necessarily better compared to a smaller one, and the identification of an optimal value can be very subjective. Nonetheless, we



(a) Visualisation showing *HP 100LX (HP Cougar)* as beating the trend over all attributes



(b) Bar chart showing Samsung as the top company at beating the trend



(c) Bar chart showing HTC as the top company at closest to median

Figure 7: Visualisations showing our answers to this assignments main questions

considered the median of each attribute to be the optimal value for each attribute, and we scored mobile devices based on their proximity to the median. More specifically, the list of normalised distances d from the median of each device is passed through the sigmoid function:

$$\text{score} = \frac{2}{1 + e^{4d}} \tag{1}$$

This way all scores are within a range of 0 to 1 and the closer a phone is to the median, the higher its score.

VI. Results

We present an interactive visualisation based on three KPIs (Figure 7) that allows the user to identify market leading companies and devices. Although the exact results varies depending on the user specified KGI, attribute and time range, the visualisations help identify clear market leaders.

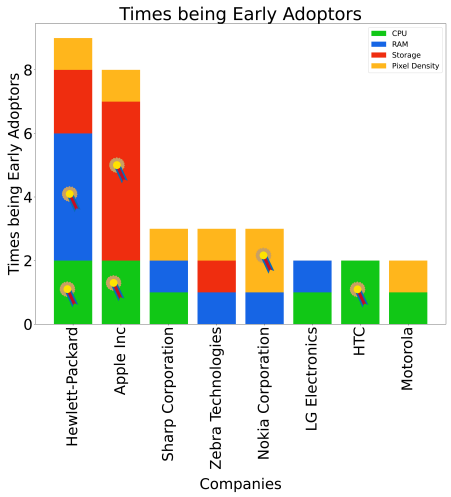


Figure 8: Number of times being earliest adopters

A. Question 1A

The overall best device will be Hewlett-Packard 100LX given it has the highest overall score of 1.46 among all devices, as shown in beating the trend with sum of attributes selected.

The best devices in terms of creating new markets in different innovation diffusion periods can be found through the interactive dashboard with earliest adapters selected. Here (Figure 6) each colour indicates the different innovation diffusion periods and the big dot is the earliest adaptor or the winner in this diffusion period. No specific device name will be provided in the report as the list will be too long and dependant on the chosen attribute and selected time range.

B. Question 1B

There were a total of 48 innovation diffusion periods with 14 companies that had been the earliest adopter at least once for all performance attributes. To save space on our report, we plotted only best eight companies on Figure 8 and it summarises the companies that becoming the early adopters of each quantities of specifications. Hewlett Packard (HP) is the overall most frequent earliest adopter with respect to all performance attributes, and Apple is the most frequent earliest adopter with respect to the *storage* attribute.

C. Question 1C

For performance related features, when the dashboard is set to *Beating the Trend*, and *Sum of Attributes* selected, we can see that Samsung is most consistently leading the market, followed by HP and Apple.

For dimensional related features, when the dashboard is set to *Closest to median*, HTC stands out as the company that manufactured the most mobile devices that were most consistent with the dimensions close to the median in all dimension.

Figure 8 shows the visualization that helped us answered this question.

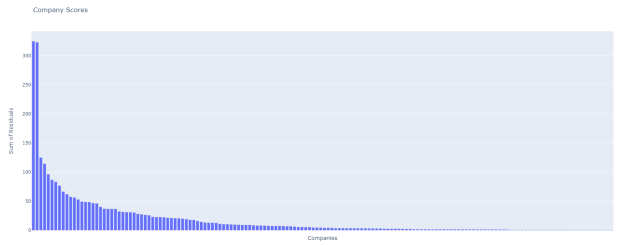


Figure 9: Company scores to define the overall most successful company

VII. Analysis of Graphics

The unique choice of visualisations corresponds to the choice of KPI required. The following analysis of the visualisation follows from Semiology of Graphics [5].

To reflect the time distance from the early adopter model to the latest which uses the same specification, it is best to convey them through lines in a two-dimensional plane. To do so, the linear shapes formed by scattered dots helps users to convey the distance from both ends of the line. To highlight the early adopter, this model is colour coded with a contrast colour of the others. Where users now can easily focus their eyes from the earliest adopter.

The bar plot conveys the number of times where a company was the earliest adopter. The choice of bar representation comes from the unit (i.e. number of times) is along the dimension one of the axes in the mapping. The information from different specification categories are conveyed through stacked bar chart. The height becomes the cumulative quantity of all categories. So the user can capture both information at once. To highlight the winners in each quantities of specifications. A gold award symbol highlights in figure 8.

The nearest to mean information express the message where strong mobile devices comes from the densely scattered area. The nearest to mean information, extracted by the linear regression plot. Unlike the other scatter plot, this visualisation conveys information through the vertical distance to the regression line. The axes choice is based on the mapping between the released year to the quantities of specifications. This shows the time evolution of innovation. To make the intensity of the residual present, or the distance to cluster, apparent to the users, the use of warm and cold colours are present. Note that this is not to express the quantitative value of the residual, but to let user perceive how the model is performing along others released from the same year. This visualisation is embedded within a dashboard which allows interaction. Users are allowed to drill down information from the overview, for example, the slider filters year range of interest. Users may select from the drop-down bars which switches between different specifications.

Furthermore, when users filter companies and the score of those chosen is highlighted by a darker colour. This contrasted with others by its luminence and it benefits to whom have colour blind and they may receive the information needed. The tooltip appeared when hover above a model point, shows the model name and the residual. This increases the knowledge level of users up to a higher affordance as a person whom has statistical background. Users can receive the information of the companies score from total residual. This already has an affordance for people with less mathematical background. Users can receive information already from the height of the bar.

In each plot, the user can click and drag on the plot to draw a box that the visualisation will then zoom into. Double clicking on the plot allows the user zoom back out to the original view of the plot. This is particularly useful when data points are bundled too close to each other. Zooming into the scatter plot in this way however does not change the year range that are used to calculate the score of each company displayed in the bar chart. To select a year range, the user should use the *Year Range* slider instead.

VIII. Conclusion

Our findings and results show that innovation is indeed mother of invention. Any established companies risked going down under in a matter of time if they either reject innovation or being slow to catch up with innovative competitors. However, not all innovations were success stories as can be seen on our plots that some earliest dots did not get much followers but the the successful ones did.

References

[1] J. McChesney, “Ghosts on the Radar — Why Radar Charts Are Easily Misread,” Towards Data Science, https://towardsdatascience.com/ghosts-on-the-radar-why-radar-charts-are-easily-misread-dba00fc399ef [online; accessed 24th March 2021].

[2] Will Kenton, “Early Adopter” Investopedia, https://www.investopedia.com/terms/e/early-adopter.asp#~text=The%20term%20%22early%20adopter%22%20refers,innovation%2C%20or%20technology%20before%20others.&text=Companies%20rely%20on%20early%20adopters,the%20product’s%20research%20and%20development [online; accessed 21st May 2021].

[3] Smart Insights, https://www.smartinsights.com/marketing-planning/marketing-models/diffusion-innovation-model/ [online; accessed 24th March 2021].

[4] Wayne W. LaMorte, Diffusion of Innovation Theory, https://sphweb.bumc.bu.edu/otlt/MPH-Modules/SB/BehavioralChangeTheories/BehavioralChangeTheories4.html, Boston University School of Public Health, [online; accessed 22nd May 2021].

[5] Jacques Bertin, William Berg. (1983) Semiology of Graphics. University of Wisconsin Press.

[6] Ester, M., H. P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996

Appendix - Python Code

May 22, 2021

1 Beating the Trend

```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

import dash
import dash_table
import dash_core_components as dcc
import dash_html_components as html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output

#============================================================ General Functions
↪  ================================================
#Brings values to desired range (default being 0 and 1)
def normalizeList(L, minValue=0.0, maxValue=1.0):
    L = L - min(L)
    L = L / (max(L) - min(L))
    return L * (maxValue - minValue) + minValue

#Returns coordinates of a line that fits the data linearly and detrended attributes
def getLinearFit(years, att, useLog):

    #Use log scale if necessary
    att = att.astype(float)
    if useLog:
        att[att==0] = 0.0001
        att = np.log(att)
```

```
    #We get fit coefficients and create y values for a fitted line
    fitCoefficients = np.polyfit(years, att, 1)
    fitLeft = years[0] * fitCoefficients[0] + fitCoefficients[1]
    fitRight = years[-1] * fitCoefficients[0] + fitCoefficients[1]

    #If using an exponential fit, we convert our fit back to the exponential scale
    if useLog:
        fitLeft = np.exp(fitLeft)
        fitRight = np.exp(fitRight)

    #Create the final lists to return
    linearFitX = [years[0], years[-1]]
    linearFitY = [fitLeft, fitRight]
    detrendedAttributes = att - (fitCoefficients[0] * years + fitCoefficients[1])
    return linearFitX, linearFitY, detrendedAttributes

#============================================================Get Data
↪  ================================================
#Read data file from excel spreadsheet
dataFileName = 'Mobile Device Data for Assignment 2.xlsx'
dataFrame = pd.read_excel(dataFileName)
dataArray = dataFrame.to_numpy().T
attributeArray = dataArray[4:]
attributeNames = dataFrame.columns.values[4:]
releaseYear = dataArray[2].astype(float)
modelNames = dataFrame['Model'].astype(str)
useLogScale = [True, True, True, False, True, False, False, False, False, False, False,
↪  False]

#Create the 'Screen to Body Ratio' attribute
width = np.copy(attributeArray[6]).astype(float)
length = np.copy(attributeArray[7]).astype(float)
width[width==0] = min(width[width!=0])
length[length==0] = min(length[length!=0])
screenToBodyRatio = attributeArray[4] * attributeArray[5] / (width * length)
screenToBodyRatio = normalizeList(screenToBodyRatio)

attSum = attributeArray[0] + attributeArray[1] + attributeArray[2] + attributeArray[11]
attSum = normalizeList(attSum)

#Data to be used in scatter plots
```

```
scatterDatNames = np.array(['RAM', 'Storage', 'CPU', 'Pixel Density', 'Sum of
↪  Attributes'])
scatterData = np.stack((attributeArray[0], attributeArray[1], attributeArray[2],
↪  attributeArray[11], attSum), axis=1).T

#Get data about phone companies
deviceAndCompanySheet = pd.read_csv('USE THIS DATASET!!! Mobile Device Data Aligned with
↪  Company Name and ID.csv')
companyId = np.array(deviceAndCompanySheet['Company_ID']).astype(int)
companyNames = np.array(deviceAndCompanySheet['Company_real']).astype(str)
dropDownOptions, dropData, usedNames, count = [], [], [], 0
for i, name in enumerate(companyNames):
    if name in usedNames:
        j = usedNames.index(name)
        dropData[j].append(i)
    else:
        usedNames.append(name)
        dropDownOptions.append({'label': name, 'value': count})
        dropData.append([i])
        count += 1
dropData = [np.array(i).astype(int) for i in dropData]


#=========================================Create Dash
↪  App==============================================================================
app = dash.Dash(__name__)

#Create a card for our controls
controls = dbc.Card([
    dbc.Row([
    dbc.Col(
        dbc.FormGroup([
            html.H1('Selected Attribute'),
            dcc.Dropdown(
                id = 'attOptions',
                options = [{'label': attName, 'value': i} for i, attName in
                ↪  enumerate(scatterDatNames)],
                value = 0)]),
        width=6),

    dbc.Col(
        dbc.FormGroup([
```

```
            html.H1('Select Company'),
            dcc.Dropdown(
                id = 'companySelection',
                options = dropDownOptions,
                multi=True
                )]),
        width=6)]),

    dbc.FormGroup([
        html.H1('Select Year Range:'),
        dcc.RangeSlider(
            id='yearSlider',
            min=releaseYear[0],
            max=releaseYear[-1], step=0.1,
            value=[releaseYear[0], releaseYear[-1]],
            marks={1991: '1991',
                1995: '1995',
                2000: '2000',
                2005: '2005',
                2010: '2010',
                2012: '2012'}),
        html.H1('Select Visualization Method:'),
        dcc.RadioItems(options=[
            {'label': 'Earliest Adapters', 'value': 'discrete'},
            {'label': 'Beating the Trend', 'value': 'continuous'},
            {'label': 'Closest to Median', 'value': 'median'}],
            value='continuous')
    ])
])

app.layout = html.Div([
    html.Div(controls),
    html.Div([
        #html.H3('Devices Beating the Trend'),
        dcc.Graph(id="bar-chart", figure={'layout': {"height": 700}}),
        #html.H3('Company Scores'),
        dcc.Graph(id="scatter",  figure={'layout': {"height": 700}})]),
    html.Div(id='DebugText')
])

@app.callback(
    Output("bar-chart", "figure"),
```

```
    Output("scatter", "figure"),
    Output('DebugText', 'children'),
    [Input("attOptions", "value"),
    Input('yearSlider', 'value'),
    Input('companySelection', 'value')])
def update_bar_chart(i, yearRange, selectedCompanies):

    debugTex = 'Blah'

    #Filter out data into specified year range
    isInYearRange = ((releaseYear >= yearRange[0]) * (releaseYear <=
    ↪  yearRange[1])).astype(bool)
    years = releaseYear[isInYearRange]
    att = scatterData[i]
    att = att[isInYearRange]
    names = modelNames[isInYearRange].astype(str)

    #Calculate linear fit based on filtered data
    linearFitX, linearFitY, detrendedAttributes = getLinearFit(years, att,
    ↪  useLogScale[i])

    numOfUniqueCompanies = len(np.unique(companyId))
    companyScores = []
    companyPhoneCount = []
    compNames = []

    for k, score in enumerate(detrendedAttributes):
        compName = companyNames[k]
        if compName in compNames:
            j = compNames.index(compName)
            companyScores[j] += score
            companyPhoneCount[j] += 1
        else:
            compNames.append(compName)
            companyScores.append(score)
            companyPhoneCount.append(1)

    companyScores = np.array(companyScores)
    companyPhoneCount = np.array(companyPhoneCount)
    compNames = np.array(compNames)
    companyFinalScore = companyScores# / companyPhoneCount
```

```
sortedZip = sorted(zip(companyFinalScore, compNames), reverse=True)
namesSorted = [name for _, name in sortedZip]
scoreSorted = [score for score, _ in sortedZip]

figBar = go.Figure()
if (selectedCompanies == None) or (selectedCompanies == []):
    figBar.add_trace(go.Bar(x=namesSorted, y=scoreSorted))
else:
    isSelected = np.zeros(len(namesSorted))
    for comp in selectedCompanies:
        name = usedNames[comp]
        isSelected[namesSorted.index(name)] = 1
    debugTex = isSelected
    figBar.add_trace(go.Bar(x=namesSorted, y=scoreSorted,
        marker=dict(color=isSelected, colorscale='Blugrn')))
#figBar.update_layout(hovermode="x unified")
figBar.update_layout(
    title="Company Scores",
    font_size = 30,
    xaxis_title="Companies",
    yaxis_title="Sum of Residuals",
    transition_duration=500,
    showlegend=False,
    hovermode="x unified")
figBar.update_xaxes(showticklabels=False)


#Scatterplot behaviour
figScat = go.Figure()
hoverStrings = np.array(['{}<br>Score: {:.2f}'.format(name, detrendedAttributes[i])
↪  for i, name in enumerate(names)])
if (selectedCompanies == None) or (selectedCompanies == []):
    figScat.add_trace(go.Scatter(
        x=years,
        y=att, mode='markers',
        hovertemplate=hoverStrings,
        name=scatterDatNames[i],
        marker=dict(size=8, color=detrendedAttributes, colorscale='Turbo')))
else:
    boolArray = np.zeros(len(releaseYear))
    for comp in selectedCompanies:
        for c in dropData[comp]:
```

```
                boolArray[c] = 1
        boolArray = boolArray.astype(bool)
        boolArray = boolArray[isInYearRange]

        figScat.add_trace(go.Scatter(
            x=years[~boolArray],
            y=att[~boolArray],
            mode='markers',
            hoverinfo='skip',
            marker=dict(size=6, color='lightblue')))

        figScat.add_trace(go.Scatter(
            x=years[boolArray],
            y=att[boolArray],
            mode='markers',
            hovertemplate=hoverStrings[boolArray],
            name=scatterDatNames[i],
            marker=dict(size=8, color=detrendedAttributes[boolArray],
            ↪  colorscale='Bluered')))

    figScat.add_trace(go.Scatter(x=linearFitX, y=linearFitY, name='Fit'))
    figScat.update_layout(
        title="Devices Beating the Trend",
        xaxis_title="Year",
        font_size = 30,
        yaxis_title=scatterDatNames[i],
        transition_duration=500,
        showlegend=False)

    #Use log scale if necessary
    if useLogScale[i]:
        figScat.update_yaxes(type="log")

    return figScat, figBar, debugTex

if __name__ == "__main__":
    app.run_server(debug=True, port=8051)
```

2 Earliest Adapters

```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
from sklearn.cluster import *

import dash
import dash_table
import dash_core_components as dcc
import dash_html_components as html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output

def normalizeList(L, minValue=0.0, maxValue=1.0):
    L = L - min(L)
    L = L / (max(L) - min(L))
    return L * (maxValue - minValue) + minValue

#============================================================Get Data
↪  ================================================
#Read data file from excel spreadsheet
dataFileName = 'Mobile Device Data for Assignment 2.xlsx'
dataFrame = pd.read_excel(dataFileName)
dataArray = dataFrame.to_numpy().T
attributeArray = dataArray[4:]
attributeNames = dataFrame.columns.values[4:]
releaseYear = dataArray[2].astype(float)
modelNames = dataFrame['Model'].astype(str)
useLogScale = [True, True, True, False, True, False, False, False, False, False, False,
↪  False]

attSum = attributeArray[0] + attributeArray[1] + attributeArray[2] + attributeArray[11]
attSum = normalizeList(attSum)

#Data to be used in scatter plots
scatterDatNames = np.array(['RAM', 'Storage', 'CPU', 'Pixel Density', 'Sum of
↪  Attributes'])
scatterData = np.stack((attributeArray[0], attributeArray[1], attributeArray[2],
↪  attributeArray[11], attSum), axis=1).T
```

```
epsilon = np.array([0.02, 0.01, 0.007, 0.002, 0.002])

#Get data about phone companies
deviceAndCompanySheet = pd.read_csv('USE THIS DATASET!!! Mobile Device Data Aligned with
↪  Company Name and ID.csv')
companyId = np.array(deviceAndCompanySheet['Company_ID']).astype(int)
companyNames = np.array(deviceAndCompanySheet['Company_real']).astype(str)
dropDownOptions, dropData, usedNames, count = [], [], [], 0
for i, name in enumerate(companyNames):
    if name in usedNames:
        j = usedNames.index(name)
        dropData[j].append(i)
    else:
        usedNames.append(name)
        dropDownOptions.append({'label': name, 'value': count})
        dropData.append([i])
        count += 1
dropData = [np.array(i).astype(int) for i in dropData]

#=========================================Create Dash
↪  App=============================================================================
app = dash.Dash(__name__)

#Create a card for our controls
controls = dbc.Card([
    dbc.Row([
    dbc.Col(
        dbc.FormGroup([
            html.H1('Selected Attribute'),
            dcc.Dropdown(
                id = 'attOptions',
                options = [{'label': attName, 'value': i} for i, attName in
                ↪  enumerate(scatterDatNames)],
                value = 0)]),
        width=6),

    dbc.Col(
        dbc.FormGroup([
            html.H1('Select Company'),
            dcc.Dropdown(
                id = 'companySelection',
                options = dropDownOptions,
```

```
                multi=True
                )]),
        width=6)]),

    dbc.FormGroup([
        html.H1('Select Year Range:'),
        dcc.RangeSlider(
            id='yearSlider',
            min=releaseYear[0],
            max=releaseYear[-1], step=0.1,
            value=[releaseYear[0], releaseYear[-1]],
            marks={1991: '1991',
                1995: '1995',
                2000: '2000',
                2005: '2005',
                2010: '2010',
                2012: '2012'}),
        html.H1('Select Visualization Method:'),
        dcc.RadioItems(options=[
            {'label': 'Earliest Adapters', 'value': 'discrete'},
            {'label': 'Beating the Trend', 'value': 'continuous'},
            {'label': 'Closest to Median', 'value': 'median'}],
            value='discrete')
    ])
])

app.layout = html.Div([
    html.Div(controls),
    html.Div([
        dcc.Graph(id="scatter",  figure={'layout': {"height": 700}})]),
        dcc.Graph(id="bar-chart", figure={'layout': {"height": 700}}),
    html.Div(id='DebugText')
])

@app.callback(
    Output("scatter", "figure"),
    Output("bar-chart", "figure"),
    Output('DebugText', 'children'),
    [Input("attOptions", "value"),
    Input('yearSlider', 'value'),
    Input('companySelection', 'value')])
def update_bar_chart(i, yearRange, selectedCompanies):
```

```
debugTex = ''

#Filter out data into specified year range
isInYearRange = ((releaseYear >= yearRange[0]) * (releaseYear <=
↪  yearRange[1])).astype(bool)
years = releaseYear[isInYearRange]
att = scatterData[i]
att = att[isInYearRange]
names = modelNames[isInYearRange].astype(str)

#Do one dimensional clustering
att = att.astype(float)
if useLogScale[i]:
    att[att==0] = np.min(att[att!=0])
    attLog = normalizeList(np.log(att))
    debugTex = np.min(attLog)
    clusterID = DBSCAN(eps=epsilon[i]).fit(attLog.reshape(-1, 1))
else:
    clusterID = DBSCAN(eps=epsilon[i]).fit(att.reshape(-1, 1))
labels = clusterID.labels_

firstInClusters, firstInClusterIDs = [], []
arrayIndex = np.arange(len(att))
for idx in np.unique(labels):
    isInCluster = (labels == idx)
    firstInClusterID = np.min(arrayIndex[isInCluster])
    if firstInClusters == []:
        firstInClusters.append([releaseYear[firstInClusterID],
        ↪  att[firstInClusterID]])
        firstInClusterIDs.append(firstInClusterID)
    if firstInClusters[-1][1] < att[firstInClusterID]:
        firstInClusters.append([releaseYear[firstInClusterID],
        ↪  att[firstInClusterID]])
        firstInClusterIDs.append(firstInClusterID)
firstInClusters = np.array(firstInClusters)

np.random.seed(seed=1)
uniqueLabels = list(np.unique(labels))
rand = np.random.rand(len(uniqueLabels))
colors = np.array([rand[uniqueLabels.index(labe)] for labe in labels])
colors[labels==-1] = 0
```

```
#Create the scatter plot
figScat = go.Figure()
figScat.add_trace(go.Scatter(
    x=years,
    y=att, mode='markers',
    hovertemplate=names,
    name=scatterDatNames[i],
    marker=dict(size=8, color=colors, colorscale='Turbo')))

figScat.add_trace(go.Scatter(
    x = years[firstInClusterIDs],
    y = att[firstInClusterIDs],
    hovertemplate=names[firstInClusterIDs],
    mode='markers',
    marker=dict(size=20, color=colors[firstInClusterIDs], colorscale='Turbo')))

figScat.update_layout(
    title="Earliest Adapters",
    font_size = 30,
    xaxis_title="Year",
    yaxis_title=scatterDatNames[i],
    transition_duration=500,
    showlegend=False)

compNames = []
companyScores = []
for idx in firstInClusterIDs:
    compName = companyNames[idx]
    if compName in compNames:
        j = compNames.index(compName)
        companyScores[j] += 1
    else:
        compNames.append(compName)
        companyScores.append(1)

figBar = go.Figure()
if (selectedCompanies == None) or (selectedCompanies == []):
    figBar.add_trace(go.Bar(x=compNames, y=companyScores))

#Add title and specify other properties of the bar graph
figBar.update_layout(
    title="Company Scores",
```

```
        font_size = 30,
        #xaxis_title="Companies",
        yaxis_title="Score",
        transition_duration=500,
        showlegend=False,
        hovermode="x unified",
        xaxis = go.XAxis(
            title = 'Companies',
            showticklabels=False),
        )

    if useLogScale[i]:
        figScat.update_yaxes(type="log")

    return figScat, figBar, debugTex#, figBar, debugTex


if __name__ == "__main__":
    app.run_server(debug=True, port=8052)
```

3 Closest to Median

```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

import dash
import dash_table
import dash_core_components as dcc
import dash_html_components as html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output

#============================================================ General Functions
↪  ================================================
#Brings values to desired range (default being 0 and 1)
def normalizeList(L, minValue=0.0, maxValue=1.0):
    L = L - min(L)
    L = L / (max(L) - min(L))
```

```
    return L * (maxValue - minValue) + minValue

#Given the scores of each model, we calculate the score of each company
def getCompanyScores(scores, companyNames):
    companyScores, compNames = [], []
    for k, score in enumerate(scores):
        compName = companyNames[k]
        if compName in compNames:
            j = compNames.index(compName)
            companyScores[j] += score
        else:
            compNames.append(compName)
            companyScores.append(score)

    #Convert from list to numpy arrays
    companyScores = np.array(companyScores)
    compNames = np.array(compNames)

    #Sort arrays based on score
    sortedZip = sorted(zip(companyScores, compNames), reverse=True)
    namesSorted = [name for _, name in sortedZip]
    scoreSorted = [score for score, _ in sortedZip]
    return namesSorted, scoreSorted

#============================================================Get Data
↪  ================================================
#Read data file from excel spreadsheet
dataFileName = 'Mobile Device Data for Assignment 2.xlsx'
dataFrame = pd.read_excel(dataFileName)
dataArray = dataFrame.to_numpy().T
attributeArray = dataArray[4:]
attributeNames = dataFrame.columns.values[4:]
releaseYear = dataArray[2].astype(float)
modelNames = dataFrame['Model'].astype(str)


#Select attributes to be used in scatter plots
indexOfDataToUse = np.array([3, 4, 5, 6, 7, 8, 9, 10])
scatterDatNames = attributeNames[indexOfDataToUse]
scatterData = attributeArray[indexOfDataToUse]

#Get data about phone companies
```

```
deviceAndCompanySheet = pd.read_csv('USE THIS DATASET!!! Mobile Device Data Aligned with
↪  Company Name and ID.csv')
companyId = np.array(deviceAndCompanySheet['Company_ID']).astype(int)
companyNames = np.array(deviceAndCompanySheet['Company_real']).astype(str)
dropDownOptions, dropData, usedNames, count = [], [], [], 0
for i, name in enumerate(companyNames):
    if name in usedNames:
        j = usedNames.index(name)
        dropData[j].append(i)
    else:
        usedNames.append(name)
        dropDownOptions.append({'label': name, 'value': count})
        dropData.append([i])
        count += 1
dropData = [np.array(i).astype(int) for i in dropData]


#=========================================Create Dash
↪  App=============================================================================
app = dash.Dash(__name__)

#Create a card for our controls
controls = dbc.Card([
    dbc.Row([

    #Create dropdown for attribute selection
    dbc.Col(
        dbc.FormGroup([
            html.H1('Select Attribute'),
            dcc.Dropdown(
                id = 'attOptions',
                options = [{'label': attName, 'value': i} for i, attName in
                ↪  enumerate(scatterDatNames)],
                value = 0)]),
        width=6),

    #Create dropdown for company selection
    dbc.Col(
        dbc.FormGroup([
            html.H1('Select Company'),
            dcc.Dropdown(
                id = 'companySelection',
```

```
                options = dropDownOptions,
                multi=True
                )]),
        width=6)]),

    #Create year selection slider
    dbc.FormGroup([
        html.H1('Select Year Range:'),
        dcc.RangeSlider(
            id='yearSlider',
            min=releaseYear[0],
            max=releaseYear[-1], step=0.1,
            value=[releaseYear[0], releaseYear[-1]],
            marks={1991: '1991',
                1995: '1995',
                2000: '2000',
                2005: '2005',
                2010: '2010',
                2012: '2012'}),

        #Create radio buttons for selecting visualization methods
        html.H1('Select Visualization Method:'),
        dcc.RadioItems(options=[
            {'label': 'Earliest Adapters', 'value': 'discrete'},
            {'label': 'Beating the Trend', 'value': 'continuous'},
            {'label': 'Closest to Median', 'value': 'median'}],
            value='median')
    ])
])

#Add controls and plots to the main app layout
app.layout = html.Div([
    html.Div(controls),
    html.Div([
        dcc.Graph(id="bar-chart", figure={'layout': {"height": 700}}),
        dcc.Graph(id="scatter",  figure={'layout': {"height": 700}})]),
    html.Div(id='DebugText')
])

#Function for updating plots based on user input
#This function gets called by the dash app library and is not called anywhere in this
↪  python code file
```

```
@app.callback(
    Output("bar-chart", "figure"),
    Output("scatter", "figure"),
    Output('DebugText', 'children'),
    [Input("attOptions", "value"),
    Input('yearSlider', 'value'),
    Input('companySelection', 'value')])
def update_bar_chart(i, yearRange, selectedCcompanies):
    debugTex = 'Blah'

    #Filter out data into specified year range and prepare data for plot
    isInYearRange = ((releaseYear >= yearRange[0]) * (releaseYear <=
    ↪  yearRange[1])).astype(bool)
    years = releaseYear[isInYearRange]
    att = scatterData[i]
    att = att[isInYearRange]
    names = modelNames[isInYearRange].astype(str)

    #Calculate scores based on distance to median
    median = np.median(att)
    distToMedian = np.abs(median - att)
    distToMedian = normalizeList(distToMedian).astype(float)
    scores = 2 / (1 + np.exp(4 * distToMedian))

    #Add main scatter data
    figScat = go.Figure()
    hoverStrings = np.array(['{}<br>Score: {:.2f}'.format(name, scores[i]) for i, name in
    ↪  enumerate(names)])
    if (selectedCompanies == None) or (selectedCompanies == []):
        figScat.add_trace(go.Scatter(
            x=years,
            y=att, mode='markers',
            hovertemplate=hoverStrings,
            name=scatterDatNames[i],
            marker=dict(size=8, color=scores, colorscale='Turbo')))

    #If any company is selected, only highligh data points corresponding to selected
    ↪  companies
    else:
        #Get selected Companies
        selectedComps = np.zeros(len(releaseYear))
        for comp in selectedCompanies:
```

```
        for c in dropData[comp]:
            selectedComps[c] = 1
    selectedComps = selectedComps.astype(bool)
    selectedComps = selectedComps[isInYearRange]

    #Add the non-highlighted points to plot
    figScat.add_trace(go.Scatter(
        x=years[~selectedComps],
        y=att[~selectedComps],
        mode='markers',
        hoverinfo='skip',
        marker=dict(size=6, color='lightblue')))

    #Add highlighted points to plot
    figScat.add_trace(go.Scatter(
        x=years[selectedComps],
        y=att[selectedComps],
        mode='markers',
        hovertemplate=hoverStrings[selectedComps],
        name=scatterDatNames[i],
        marker=dict(size=8, color=scores[selectedComps], colorscale='Bluered')))

#Add line representing the median
figScat.add_trace(go.Scatter(
    x = [np.min(years), np.max(years)],
    y = [median, median],
    name = 'Median'))

#Specify plot title and other plot attributes
figScat.update_layout(
    title="Closest to Median",
    font_size = 30,
    xaxis_title="Year",
    yaxis_title=scatterDatNames[i],
    transition_duration=500,
    showlegend=False)

#Create company scores bar graph
figBar = go.Figure()
namesSorted, scoreSorted = getCompanyScores(scores, companyNames)
if (selectedCompanies == None) or (selectedCompanies == []):
    figBar.add_trace(go.Bar(x=namesSorted, y=scoreSorted))
```

```
    #If any companies are selected, highlight the selected companies
    else:
        isSelected = np.zeros(len(namesSorted))
        for comp in selectedCompanies:
            name = usedNames[comp]
            isSelected[namesSorted.index(name)] = 1
        figBar.add_trace(go.Bar(
            x=namesSorted,
            y=scoreSorted,
            marker=dict(
                color=isSelected,
                colorscale='Blugrn')))

    #Add title and specify other properties of the bar graph
    figBar.update_layout(
        title="Company Scores",
        font_size = 30,
        xaxis_title="Companies",
        yaxis_title="Sum of Residuals",
        transition_duration=500,
        showlegend=False,
        hovermode="x unified")
    figBar.update_xaxes(showticklabels=False)

    #Return figures to dash app
    return figScat, figBar, debugTex

if __name__ == "__main__":
    app.run_server(debug=True, port=8053)
```

4 Scatter Density Plot

```
import pandas as pd
from pandas import Series, DataFrame
import seaborn as sns
import matplotlib.pyplot as plt

normed_product_data = pd.read_csv("Mobile Device Data for Assignment 2 - Normalized
↪  Product Data.csv")
```

```
#====================Scatter Plot=======================
count=1
plt.subplots(figsize=(80,80))
for i in ['RAM Capacity (Mb)', 'Storage (Mb)', 'CPU Clock (MHz)',
       'Display Diagonal (in)', 'Display Width(px)', 'Display Length(px)',
       'Width (mm)', 'Length (mm)', 'Depth (mm)', 'Volume (cubic cm)',
       'Mass (grams)', 'Pixel Density (per inch)']:
  plt.subplot(4,4,count)
  sns.scatterplot(normed_product_data["Release Year"], normed_product_data[i], color =
  ↪  'red')
  plt.xlabel('Release Year',fontsize = 40)
  plt.ylabel(i,fontsize = 40)
  plt.xticks(fontsize = 40)
  plt.yticks(fontsize = 40)
  count+=1
plt.show()


#====================Density Plot=======================
count=1
plt.subplots(figsize=(20, 20))
for i in ['RAM Capacity (Mb)', 'Storage (Mb)', 'CPU Clock (MHz)',
       'Display Diagonal (in)', 'Display Width(px)', 'Display Length(px)',
       'Width (mm)', 'Length (mm)', 'Depth (mm)', 'Volume (cubic cm)',
       'Mass (grams)', 'Pixel Density (per inch)']:
  plt.subplot(4,4,count)
  sns.distplot(normed_product_data[i], rug=True)
  count+=1
plt.show()
```