**PROBABILITY AND STATISTICS SEMESTER PROJECT Fall 2023**

**Laptop Price Predictor**



|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 21F-9167 | Saad Ashraf | CS 5B |

*Next Pages:*

**1. Problem Statement**

The increasing variety of laptops in the market with diverse specifications makes it challenging for consumers to determine the fair price of a laptop based on its features. Our project addresses this problem by developing a laptop price prediction model using machine learning techniques.

**2. Objective**

The main objective of this project is to create a reliable and accurate prediction model that can estimate the price of a laptop based on its essential parameters. By leveraging linear regression, we aim to provide users with a tool that helps them make informed decisions when purchasing a laptop.

**3. Data Description**

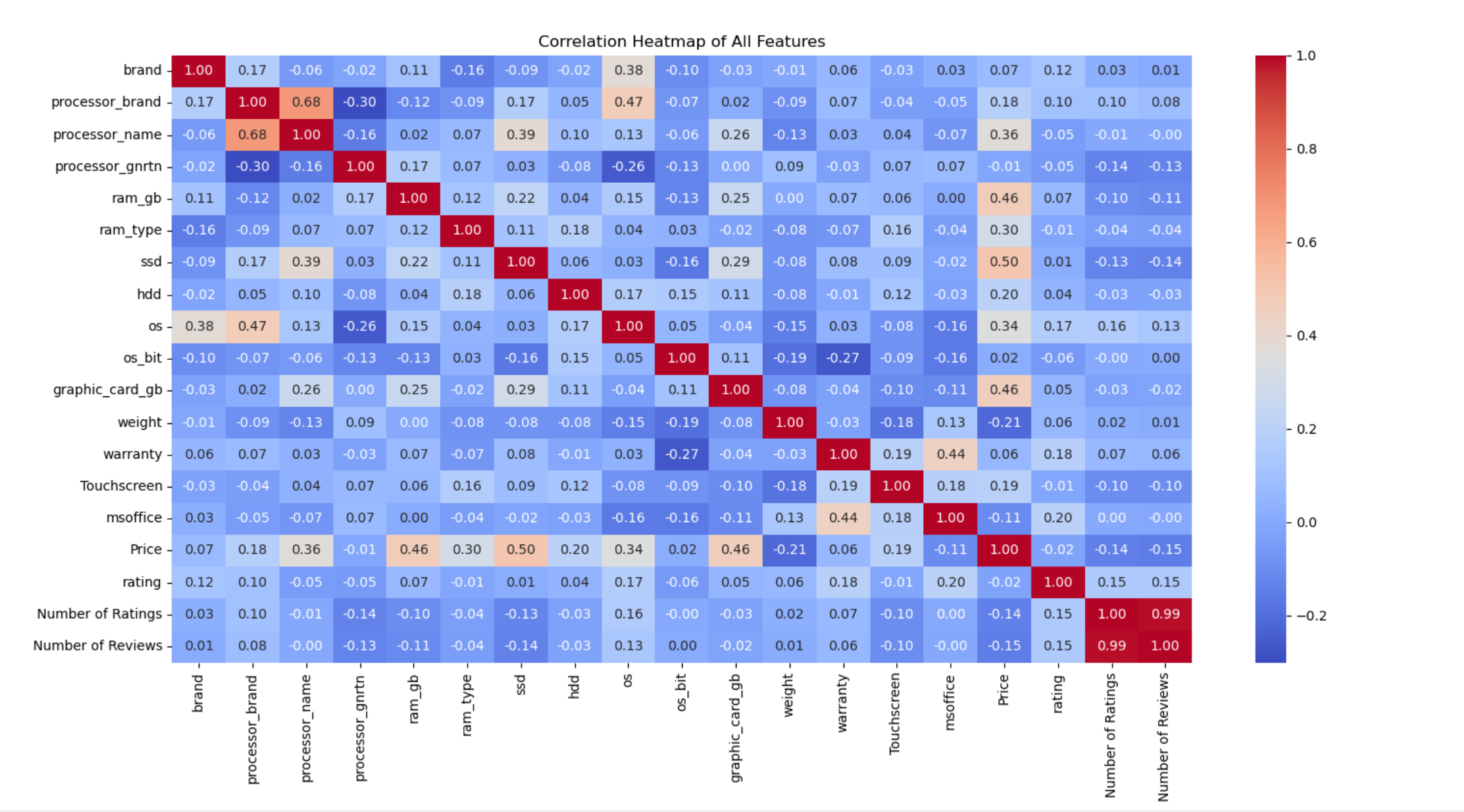
We utilized a dataset from Kaggle, accessible through the following link: https://www.kaggle.com/datasets/anubhavgoyal10/laptop-prices-dataset

The dataset includes information on various laptop features such as brand, processor details, RAM specifications, storage types, operating system, graphics card, weight, warranty, touchscreen availability, and the presence of Microsoft Office.

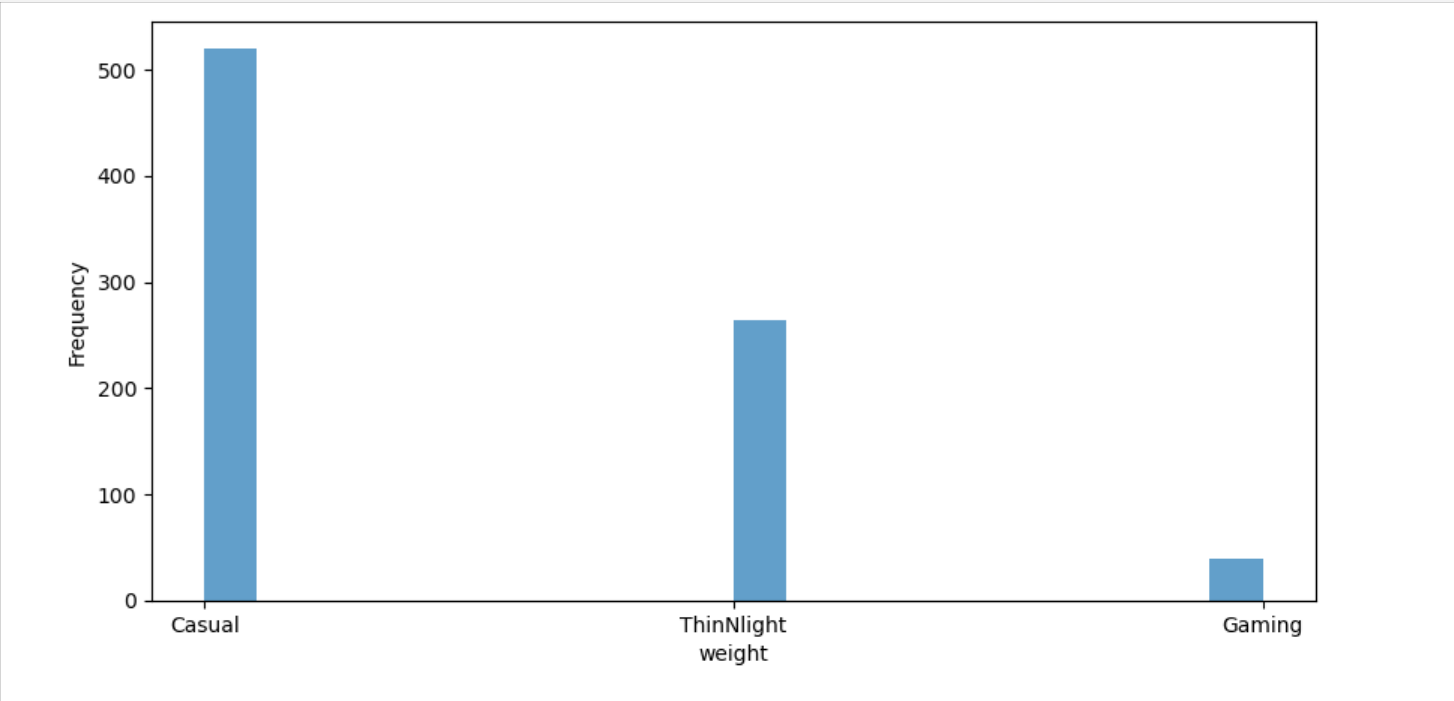
**4. Results**

After preprocessing the dataset and training the linear regression model, we achieved a prediction model capable of estimating laptop prices based on the specified parameters. The model's accuracy was validated using appropriate evaluation metrics, and the results demonstrated its effectiveness in providing reliable price predictions. Heatmap was used to find attributes that effect price the most. It was *weight*, *graphic\_card\_gb, ssd, ram\_gb.*

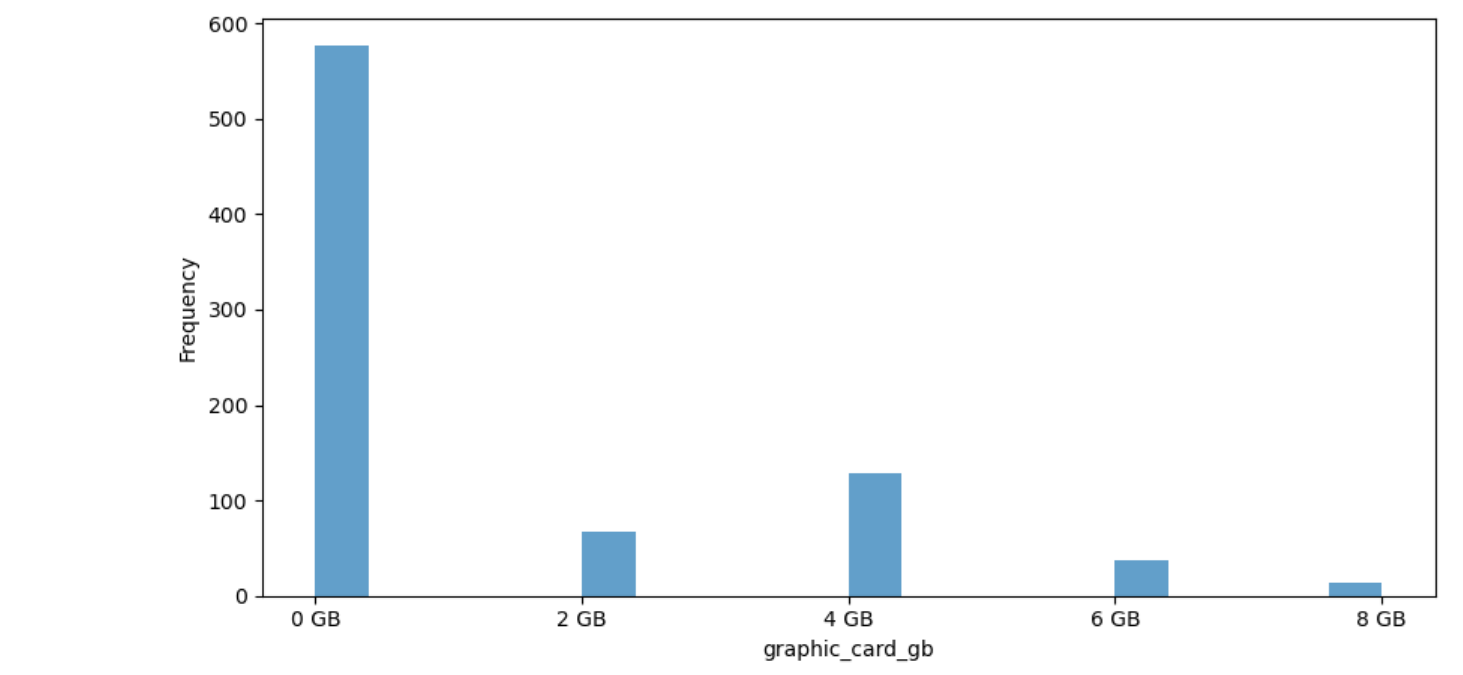
*Heatmap:*



*Histogram (weight):*



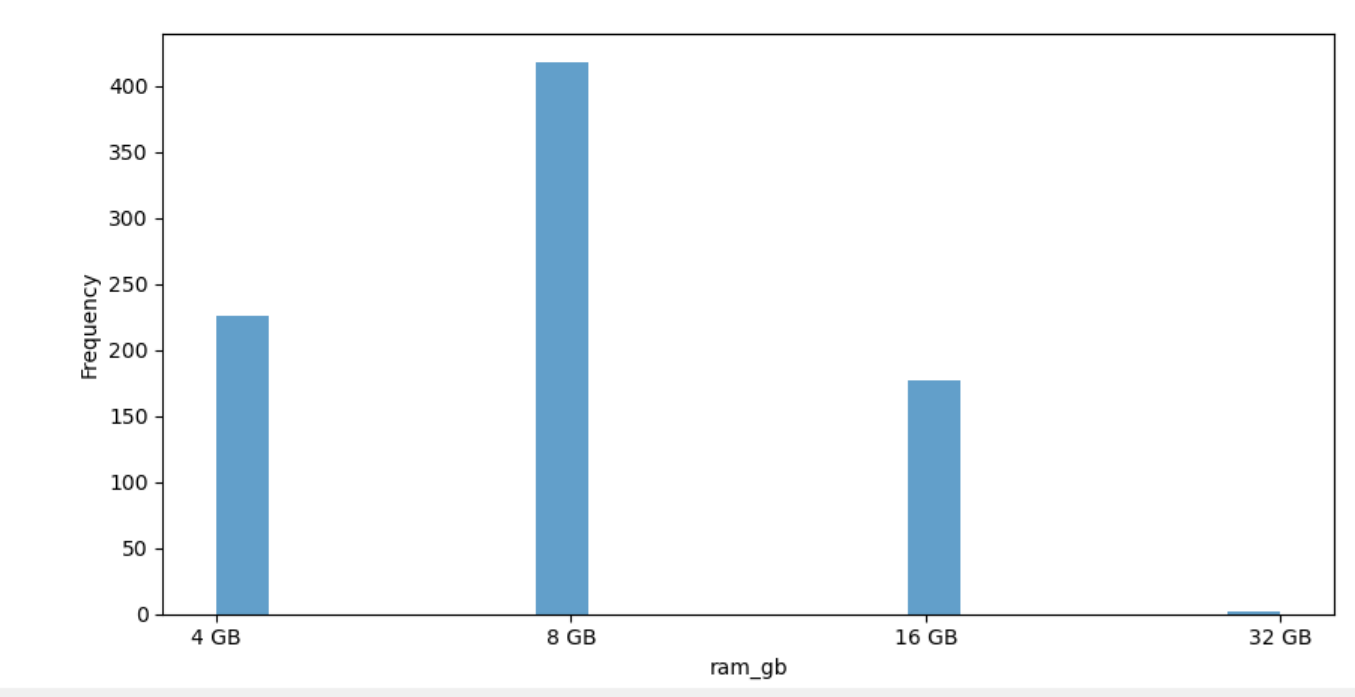
*Histogram (graphic\_card\_gb):*



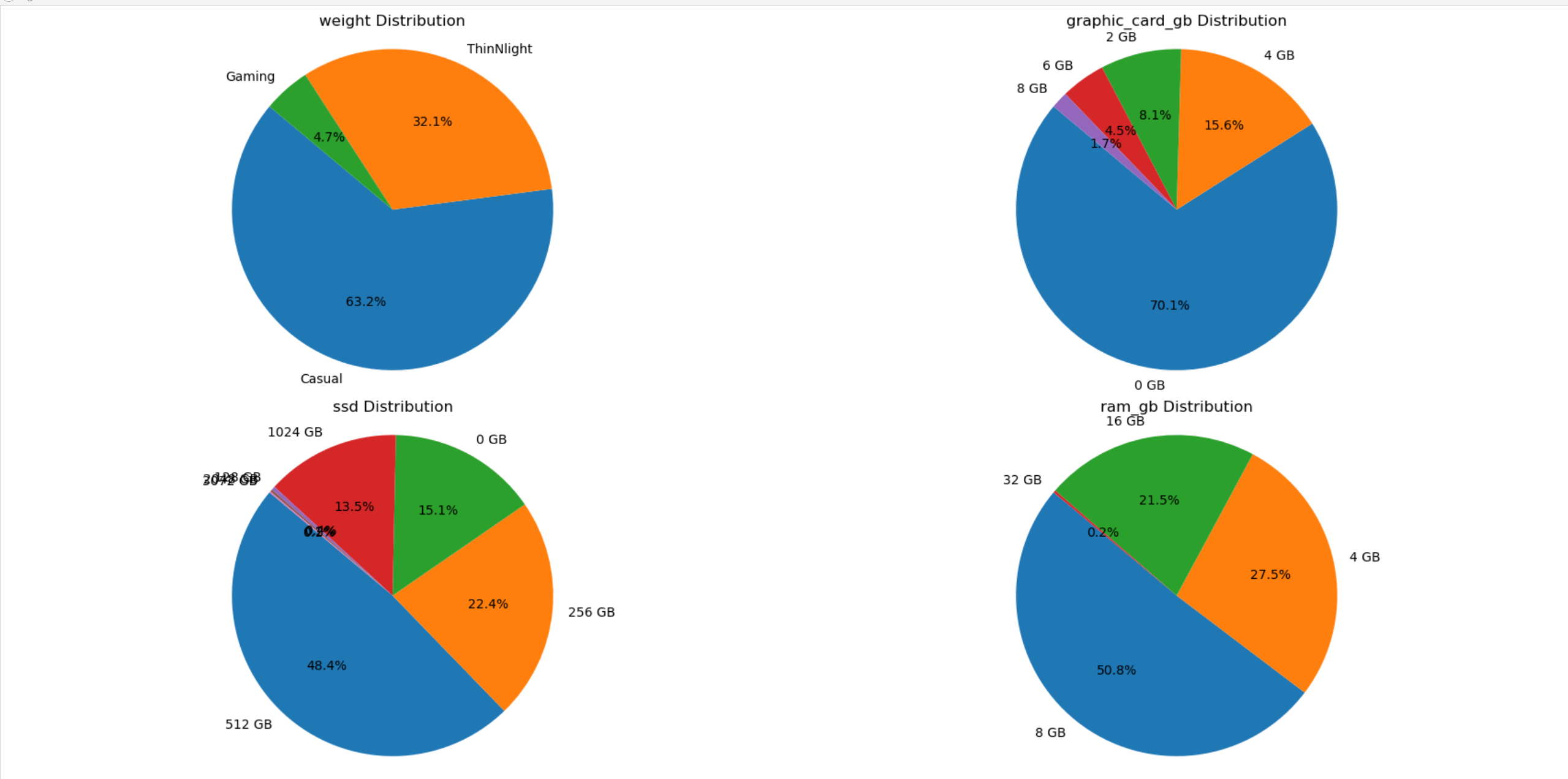
*Histogram (ssd):*



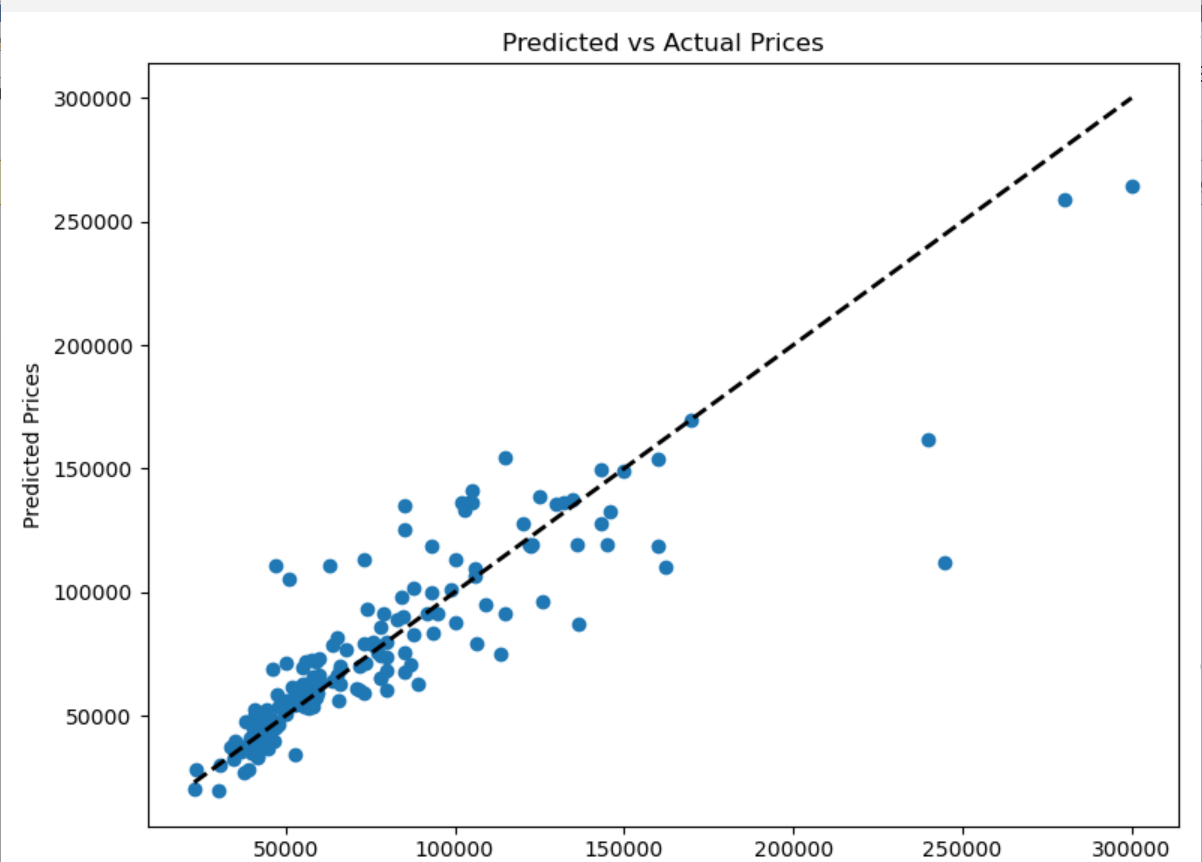
*Histogram (ram\_gb)*

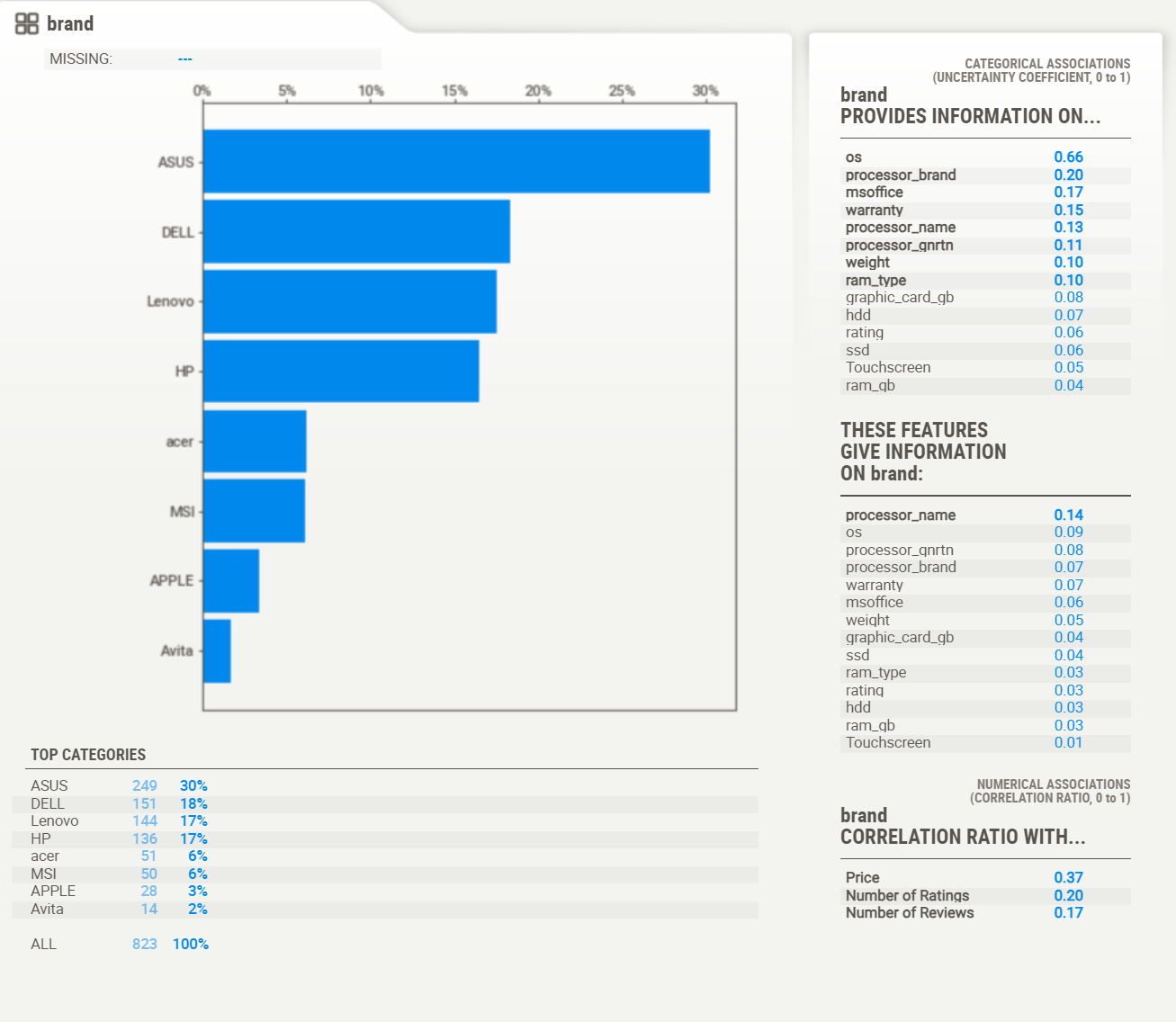


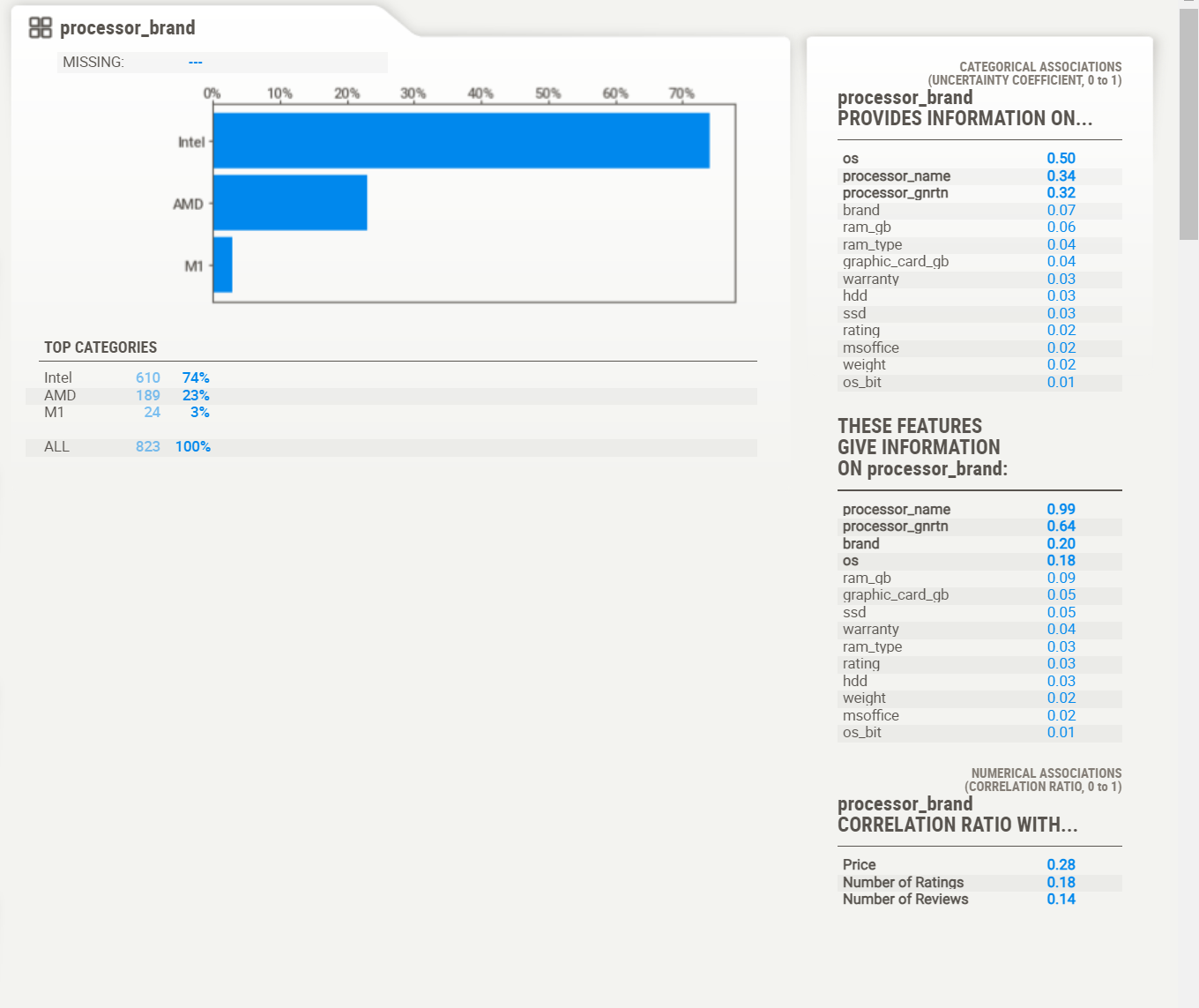
*Pie Chart (weight distribution, graphic\_card\_gb distribution, ssd distribution, ram\_gb distribution):*

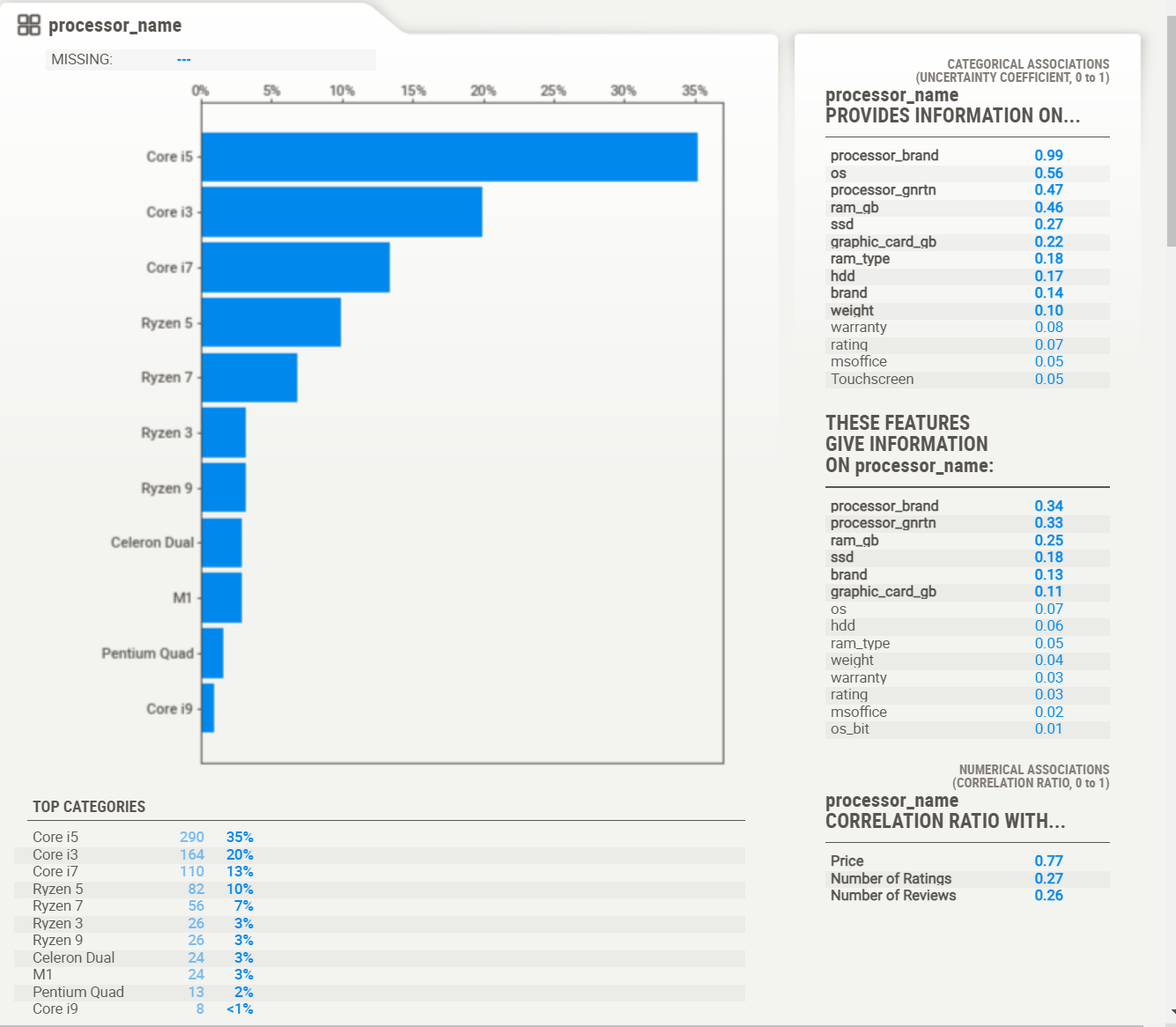


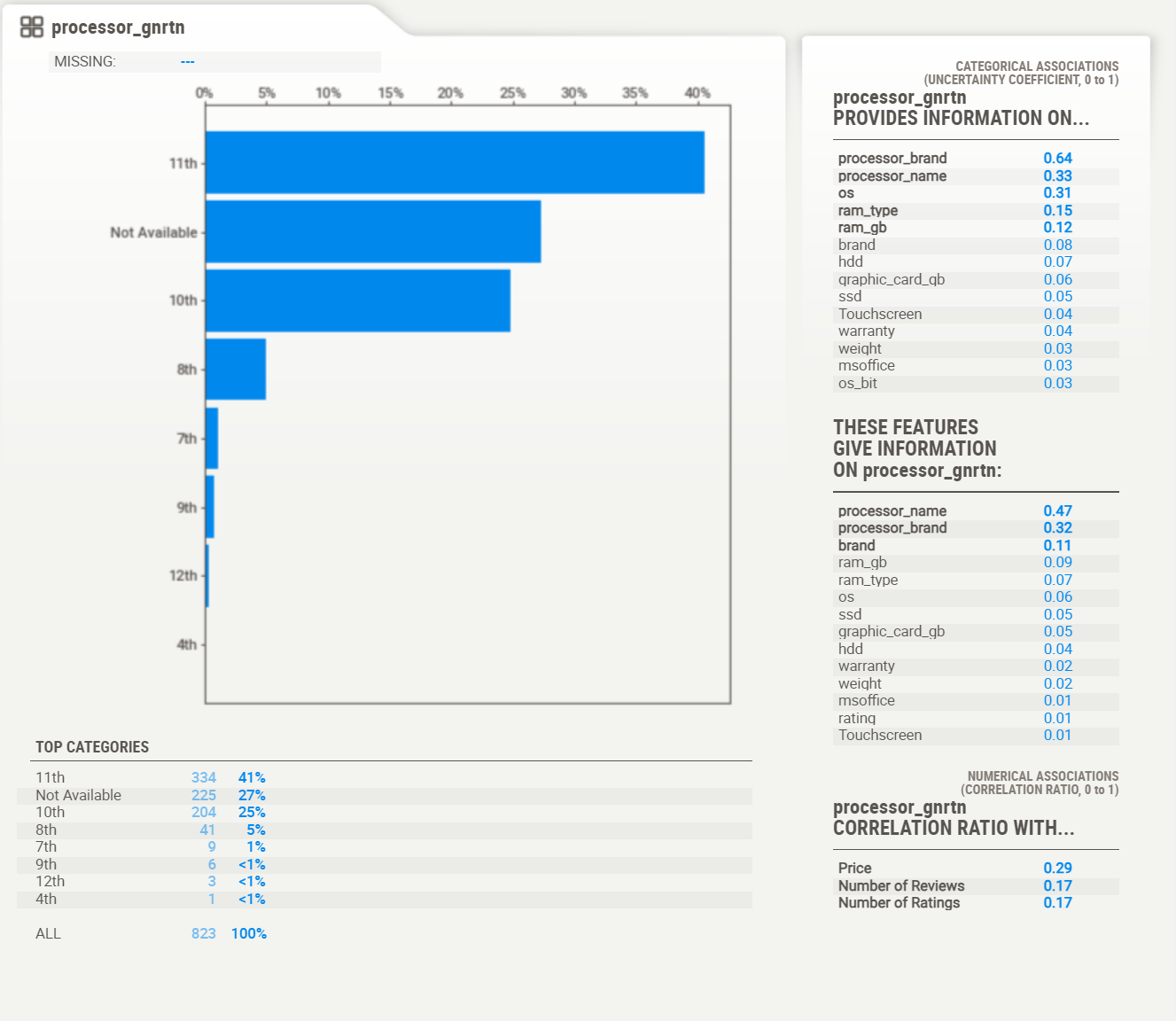
*Scatter Plot:*

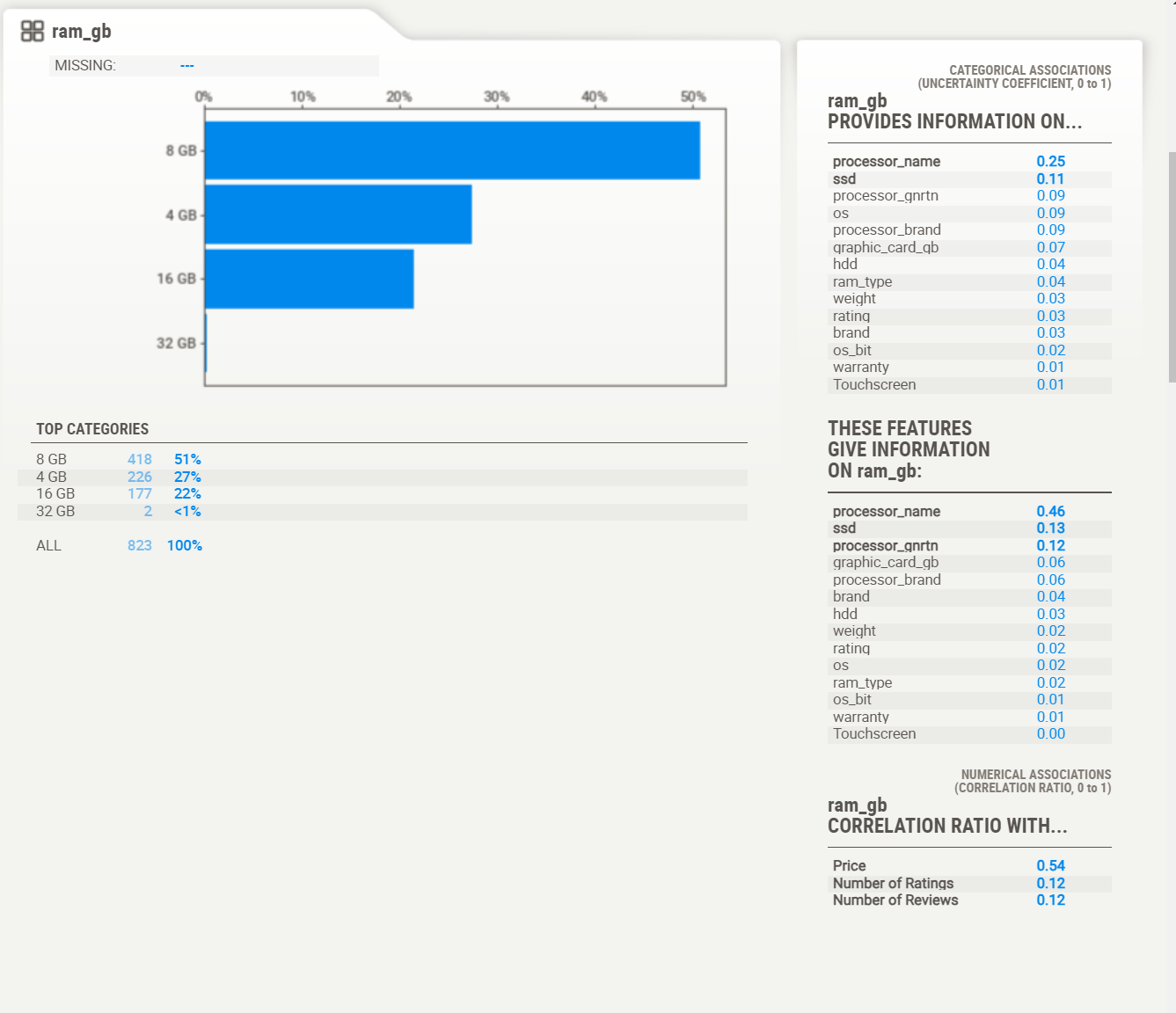
**

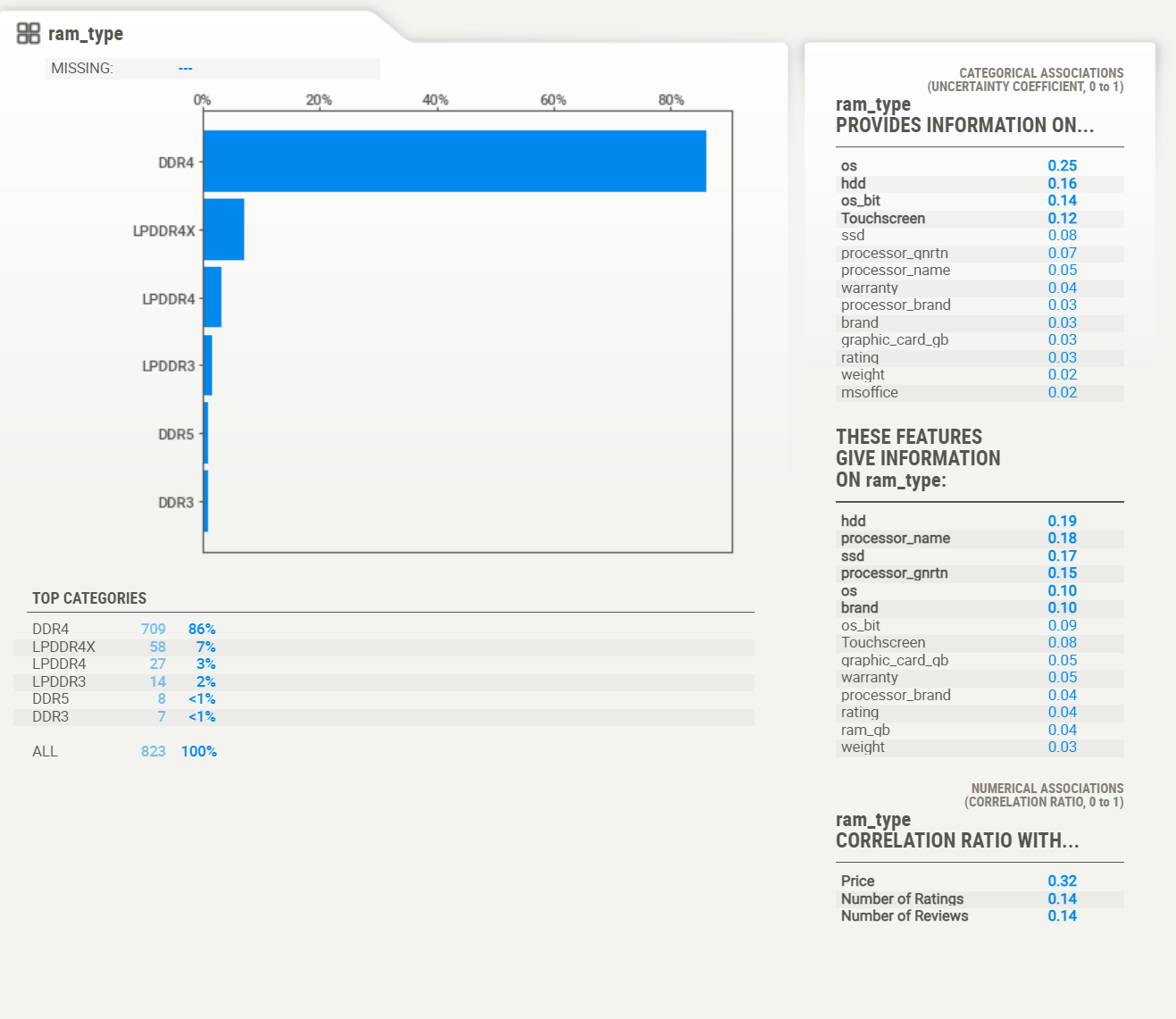
**

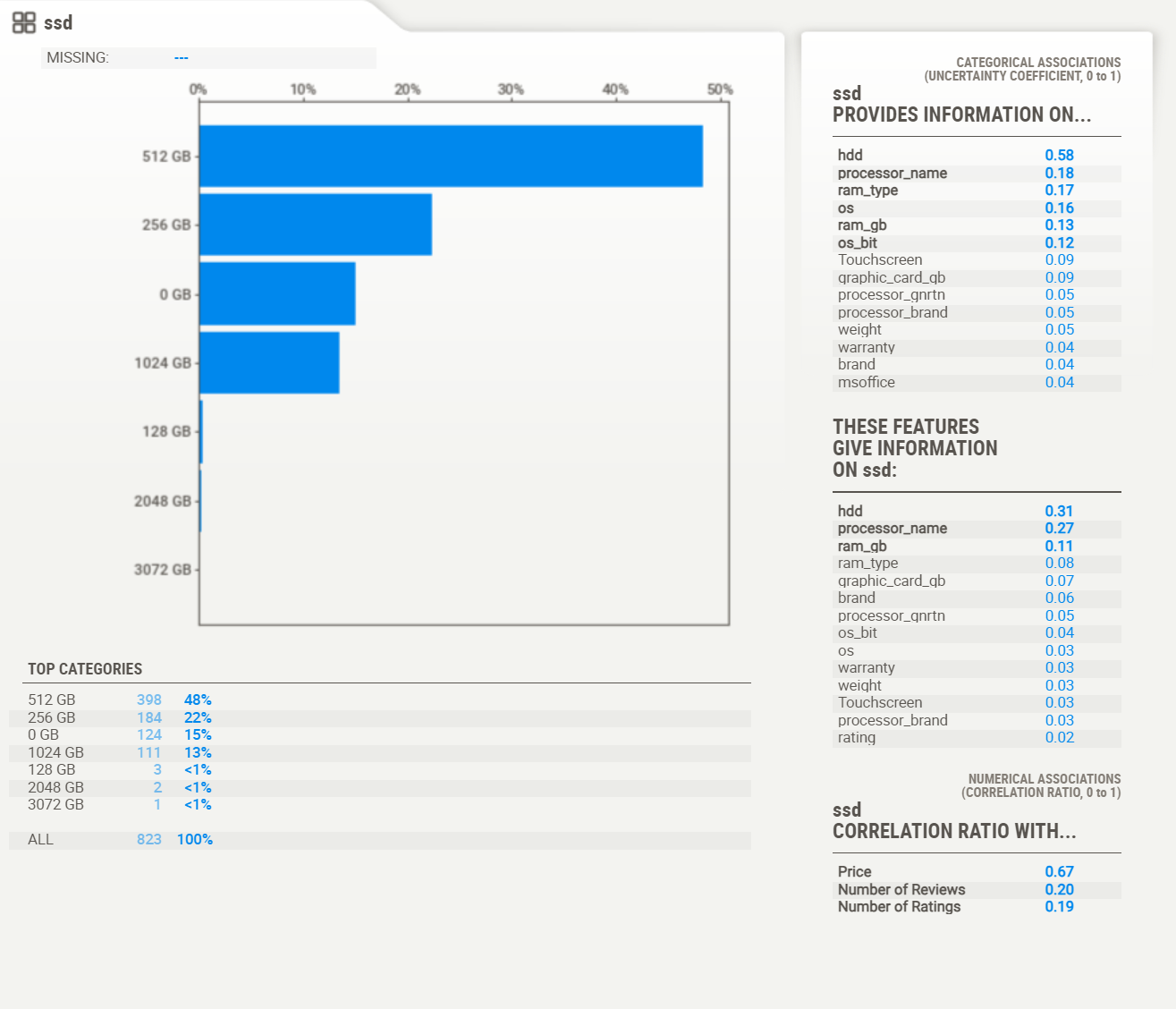
**

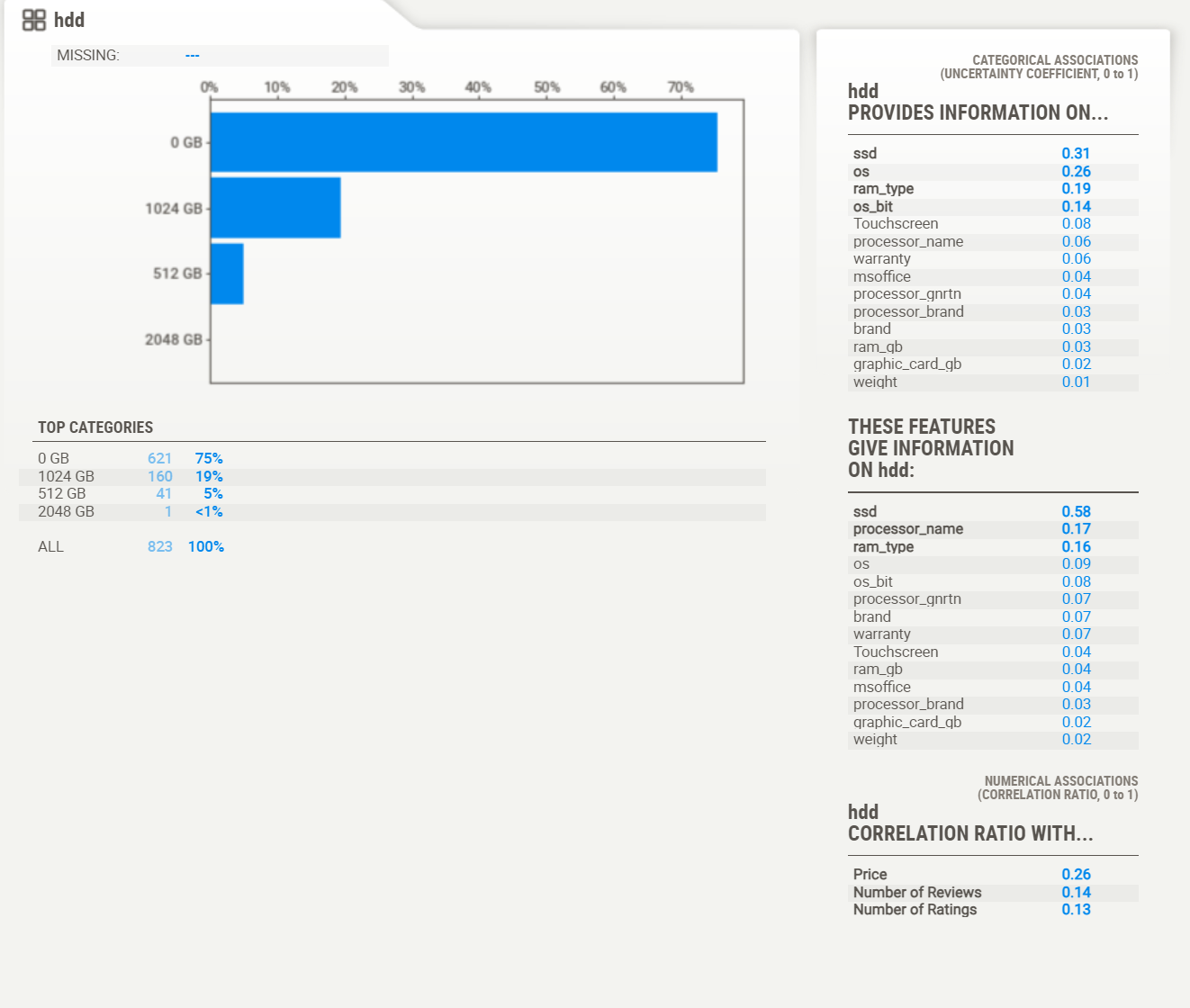
**

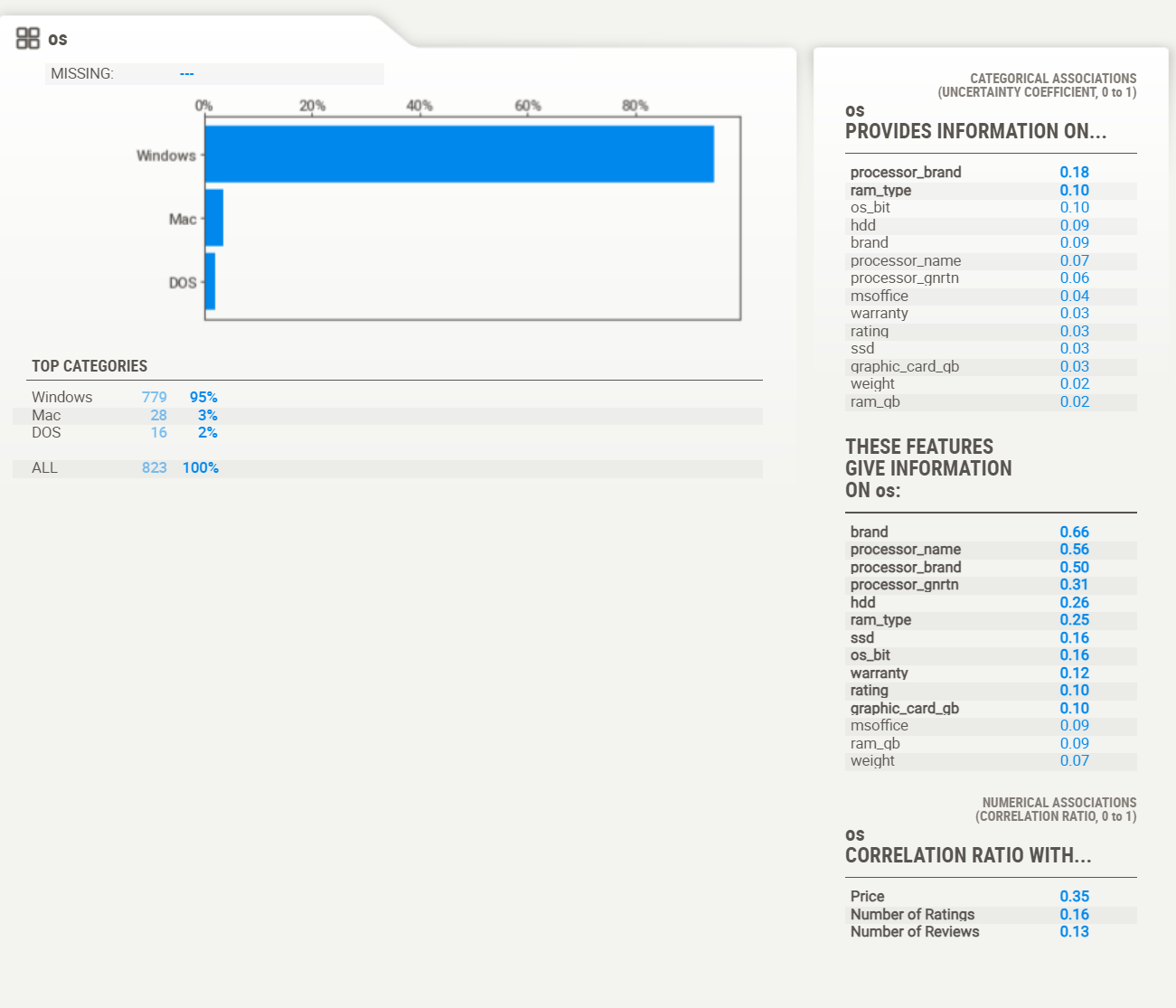
**

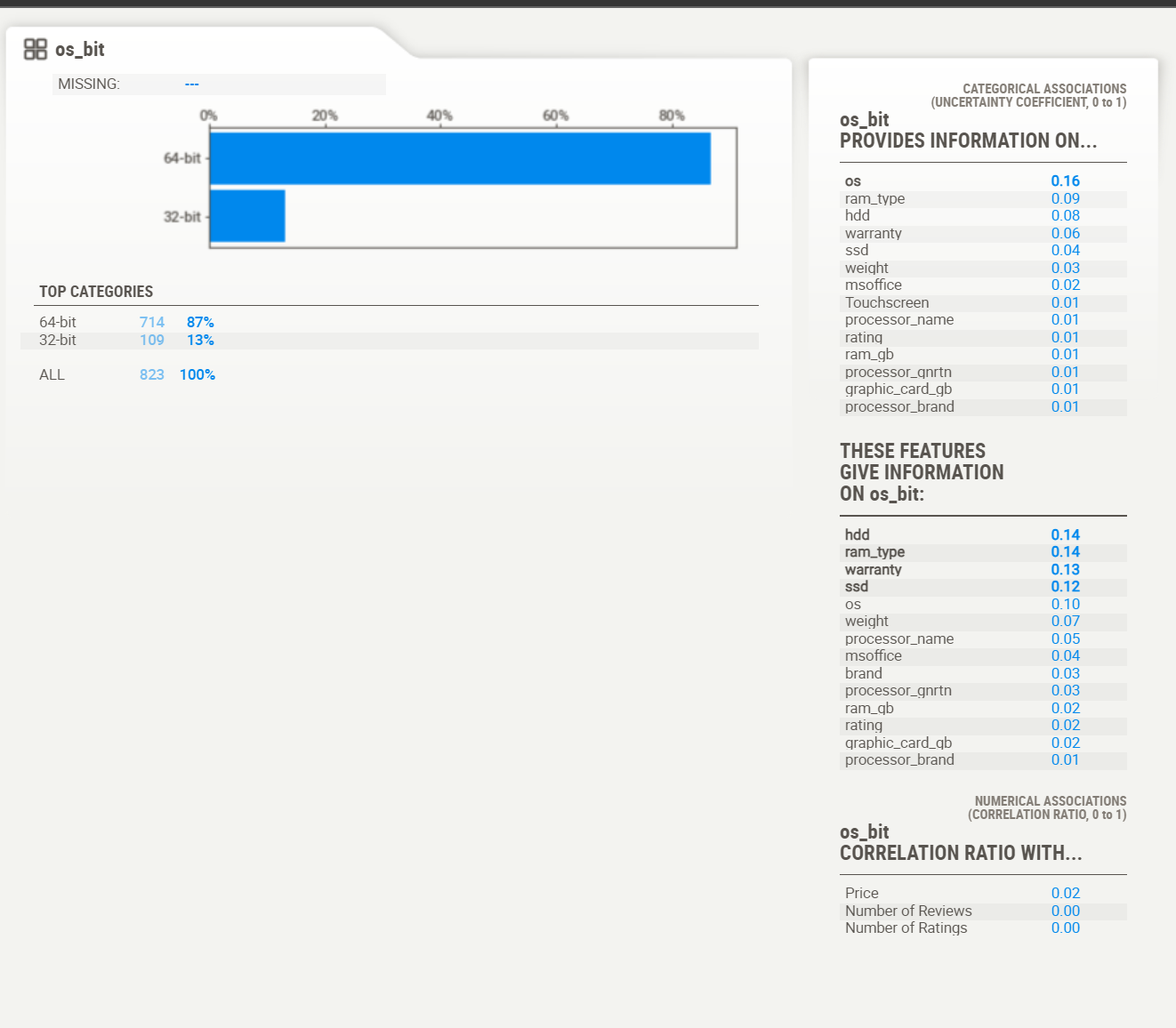
**

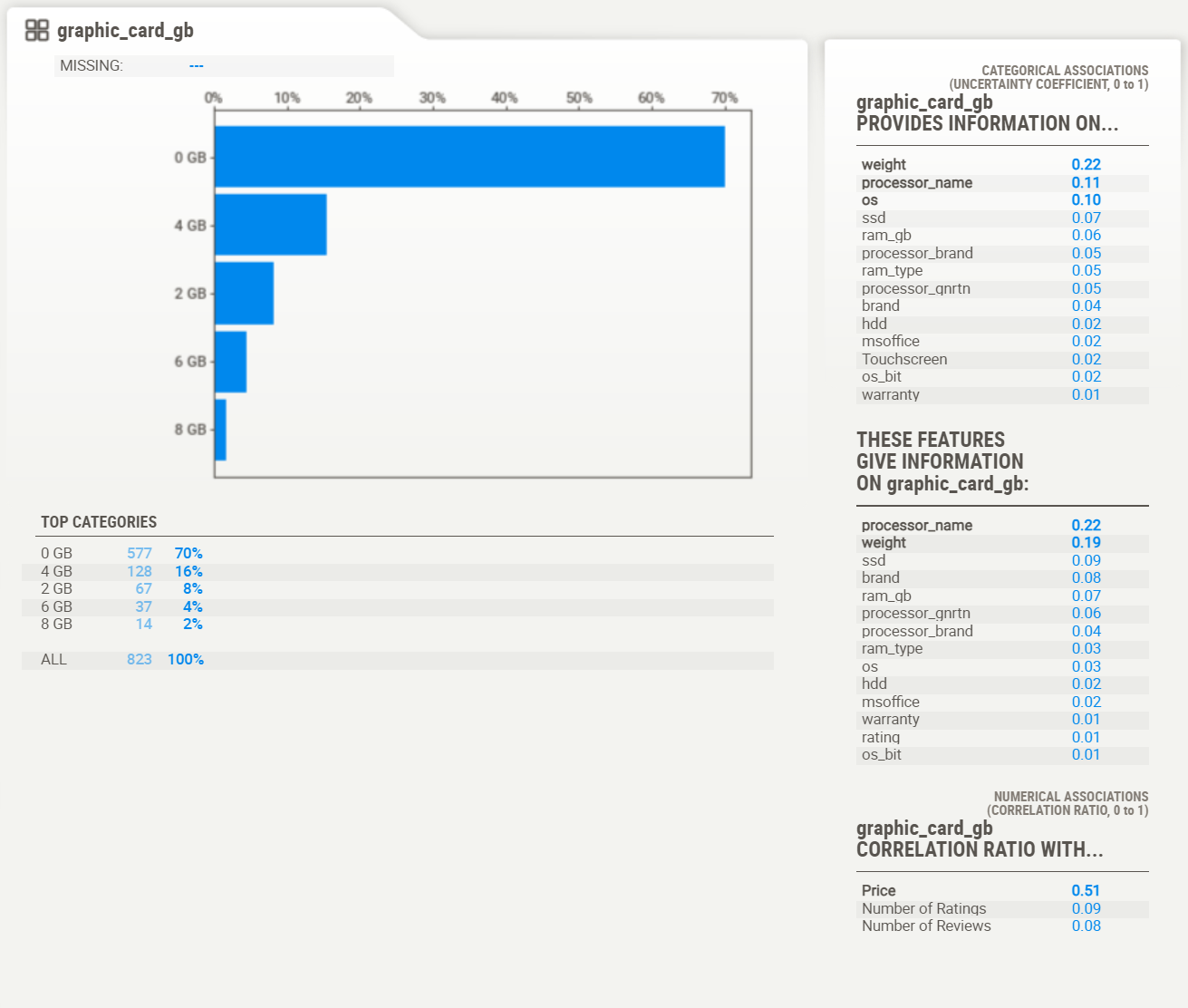
**

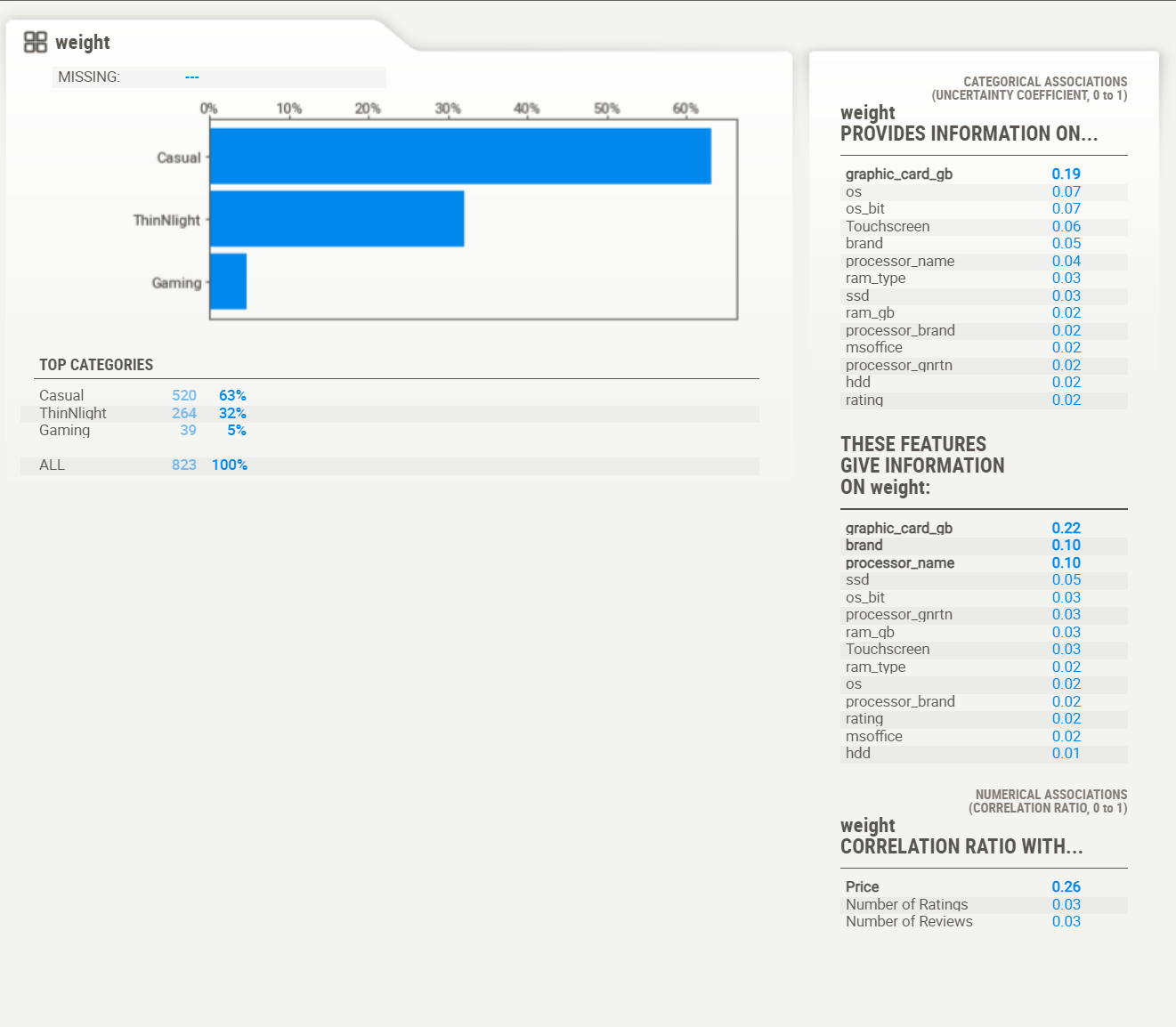
**

**

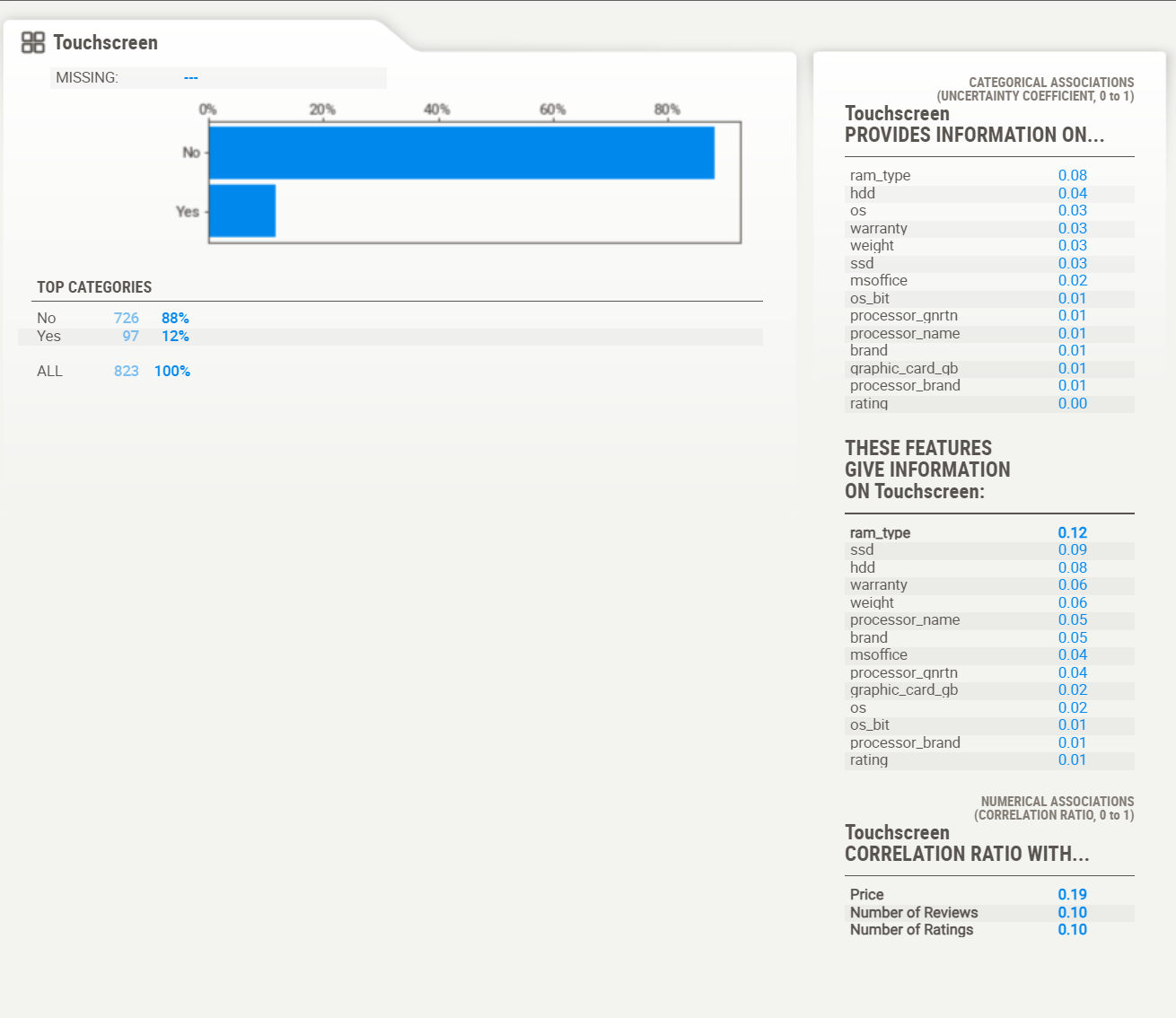
**

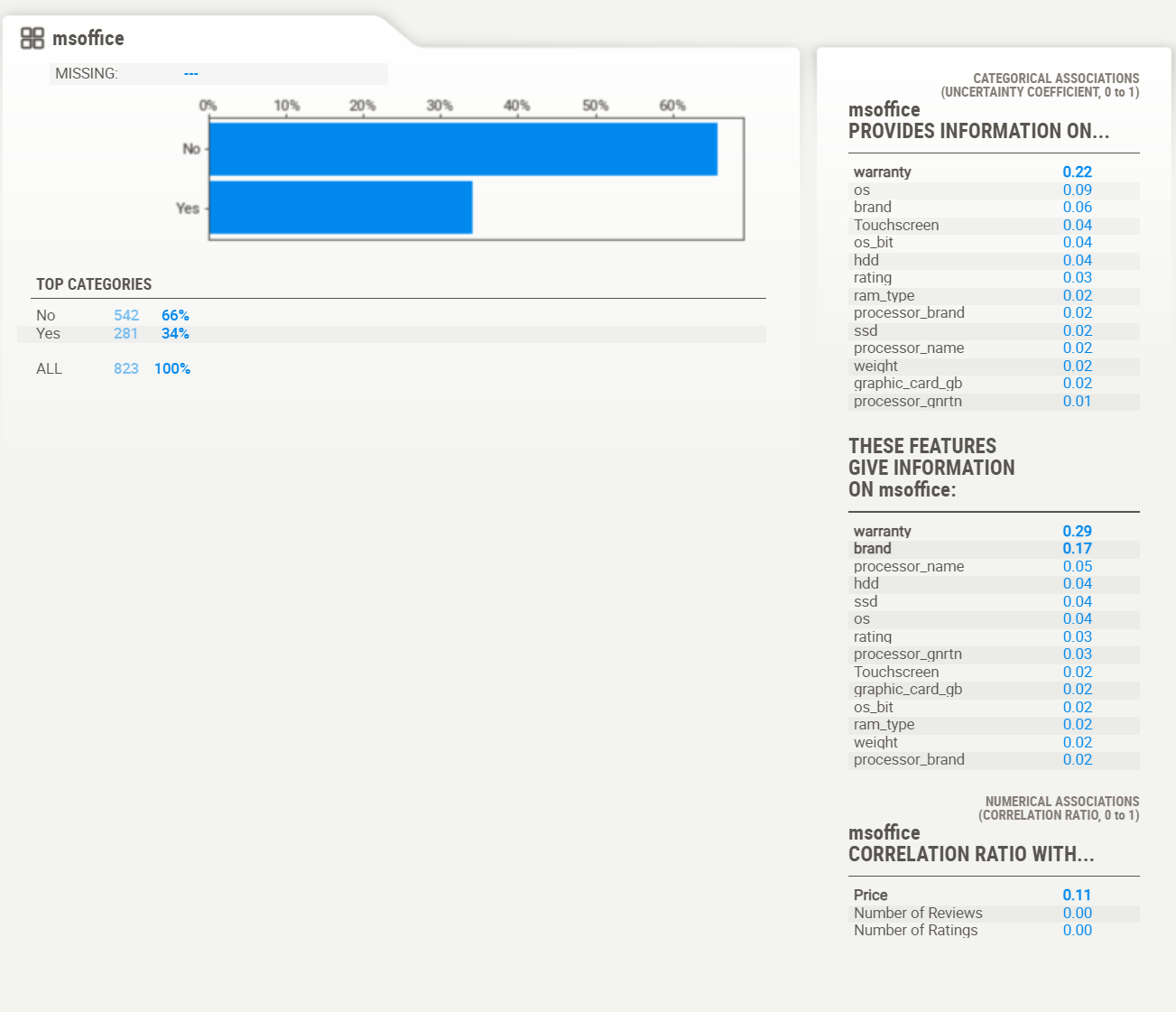
**

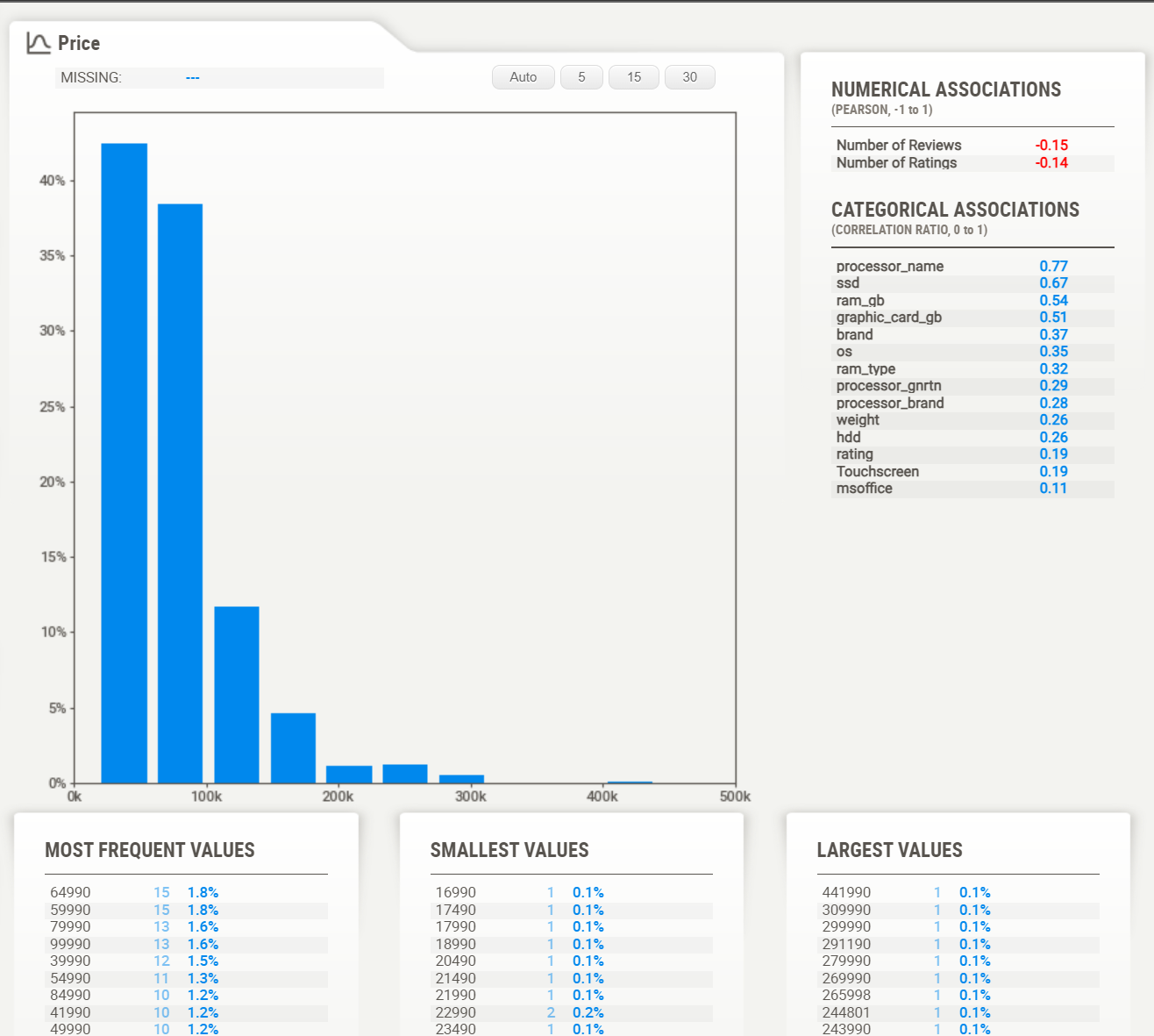
**

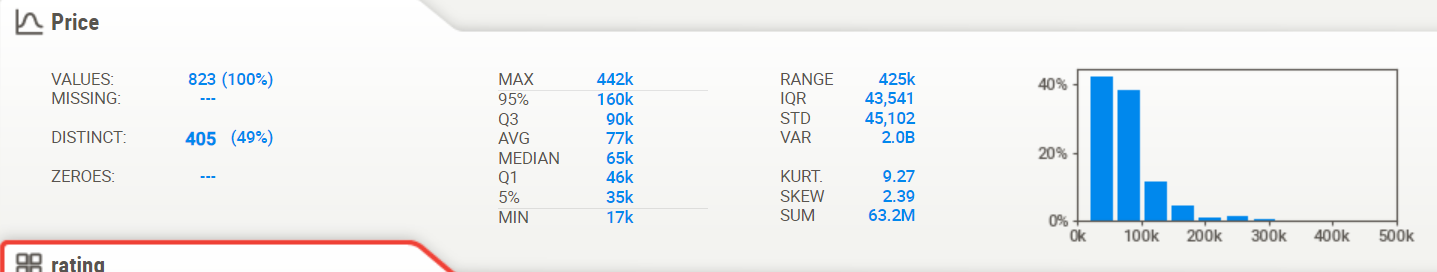
**

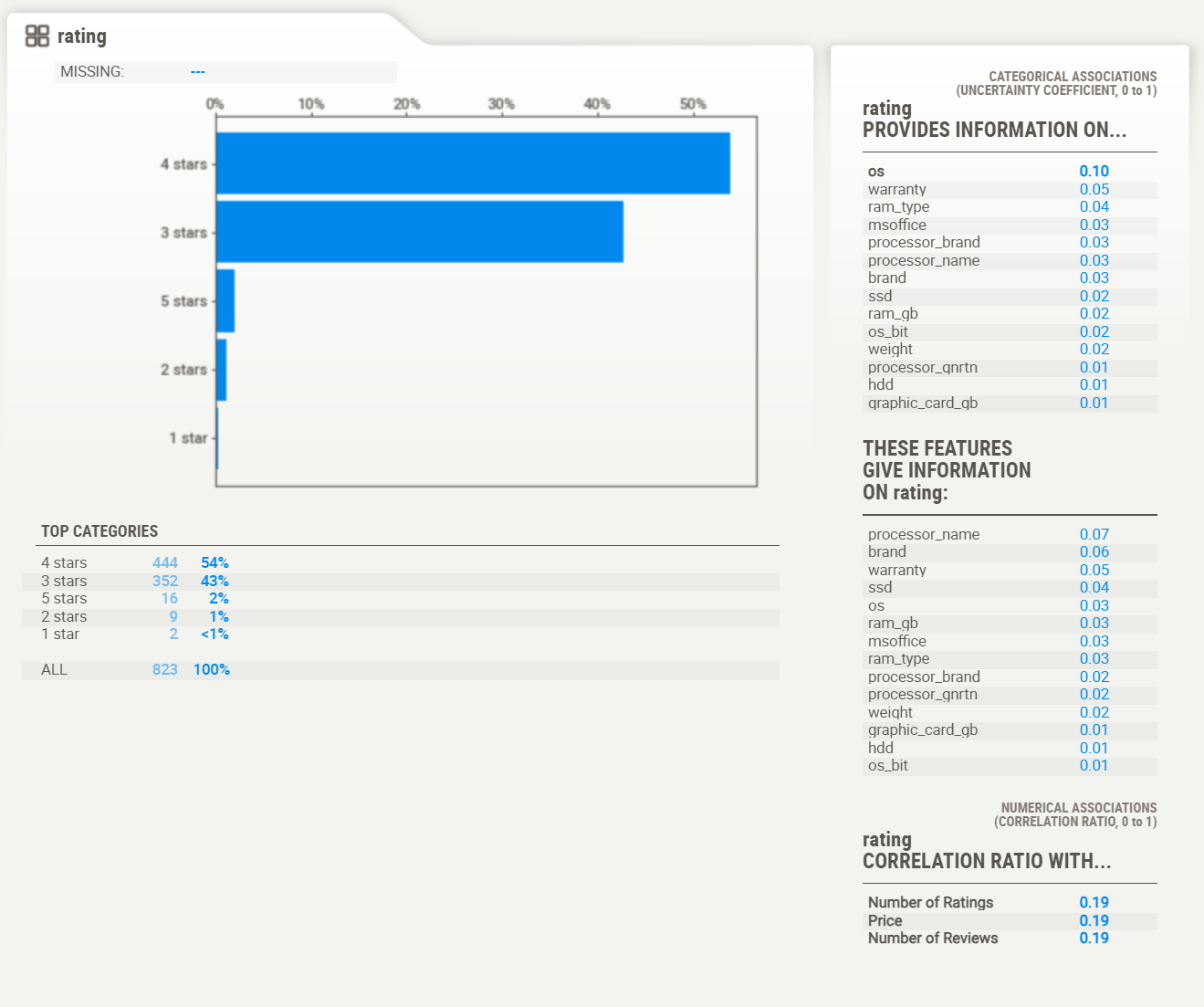
**

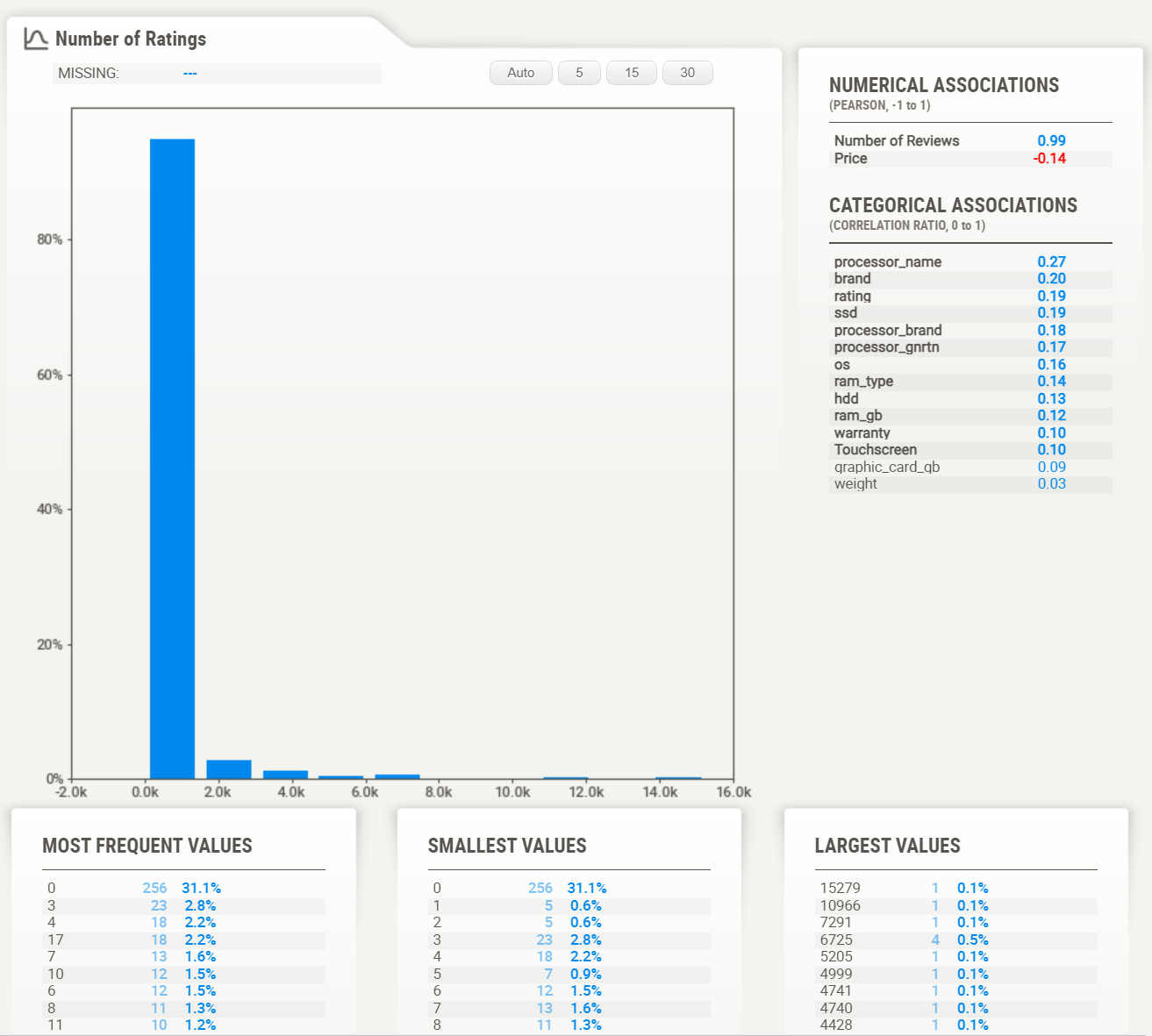
**

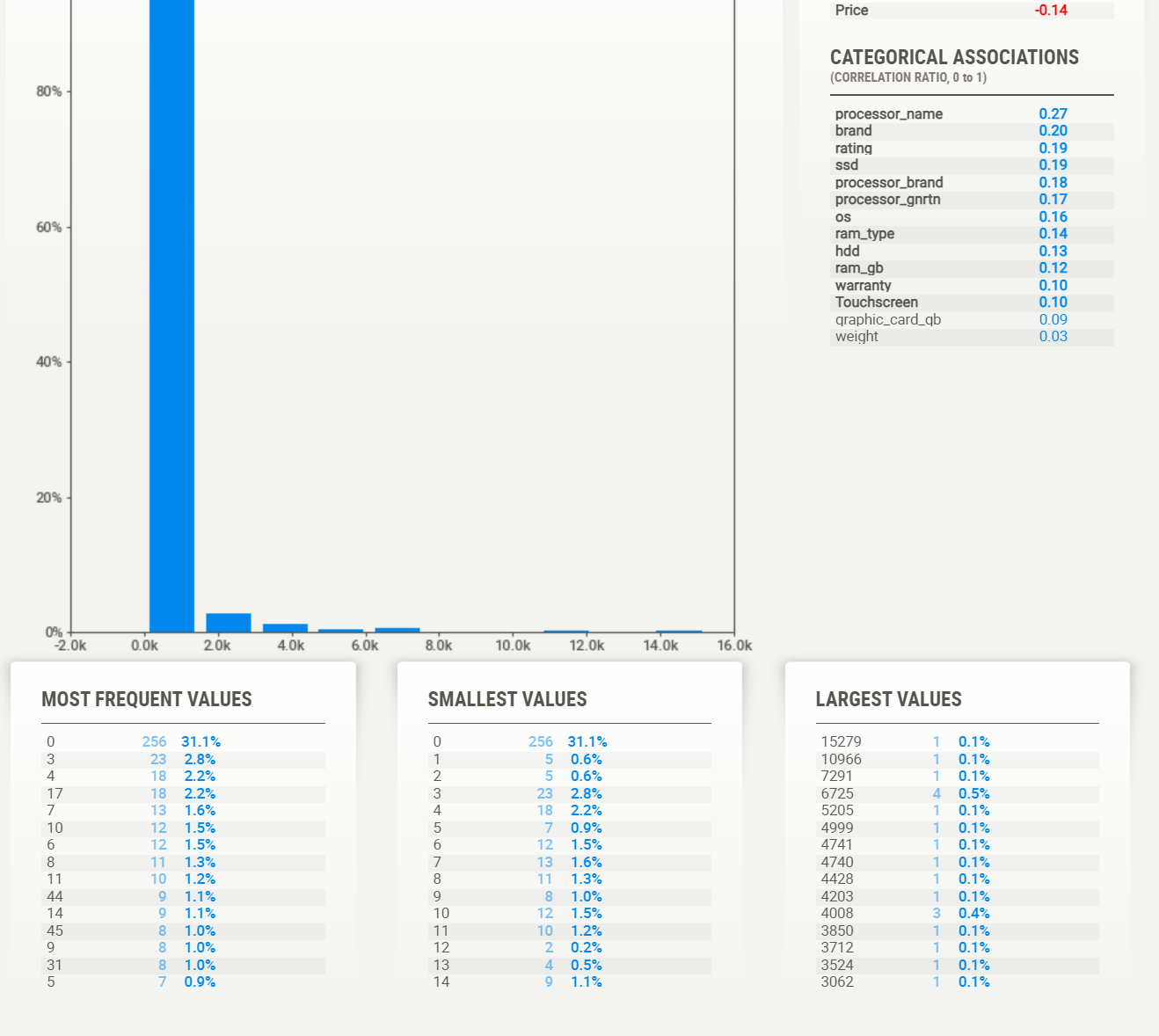
**

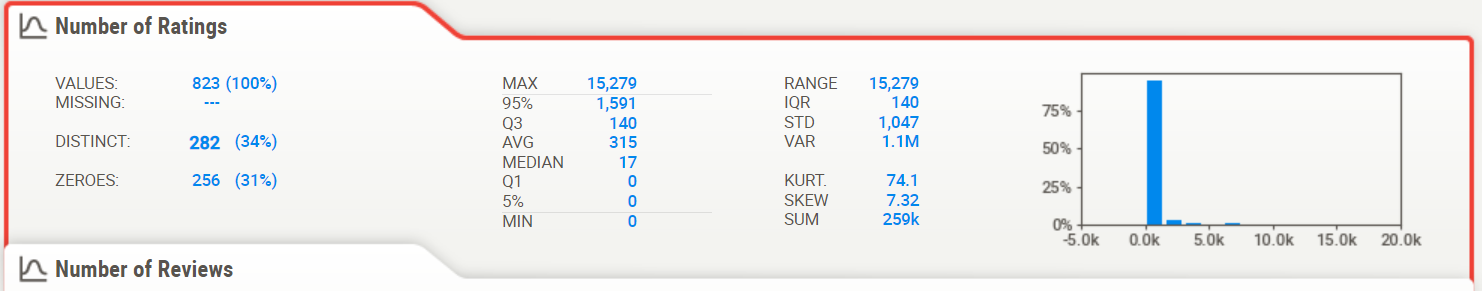
**

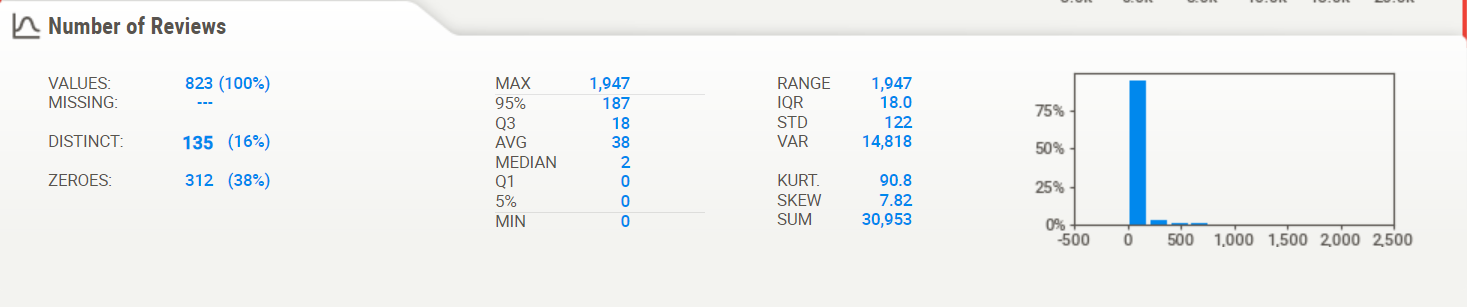
**

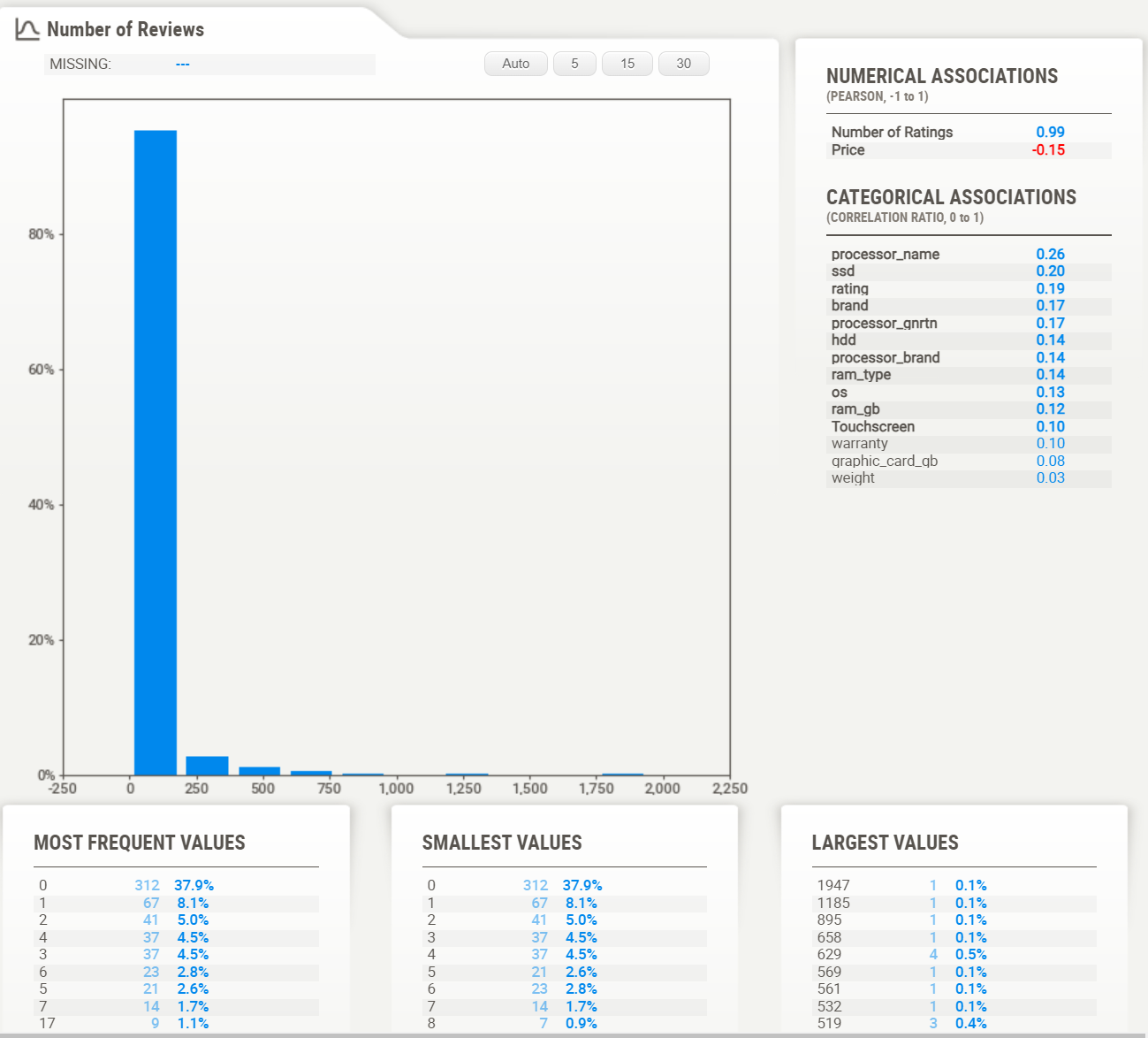
**

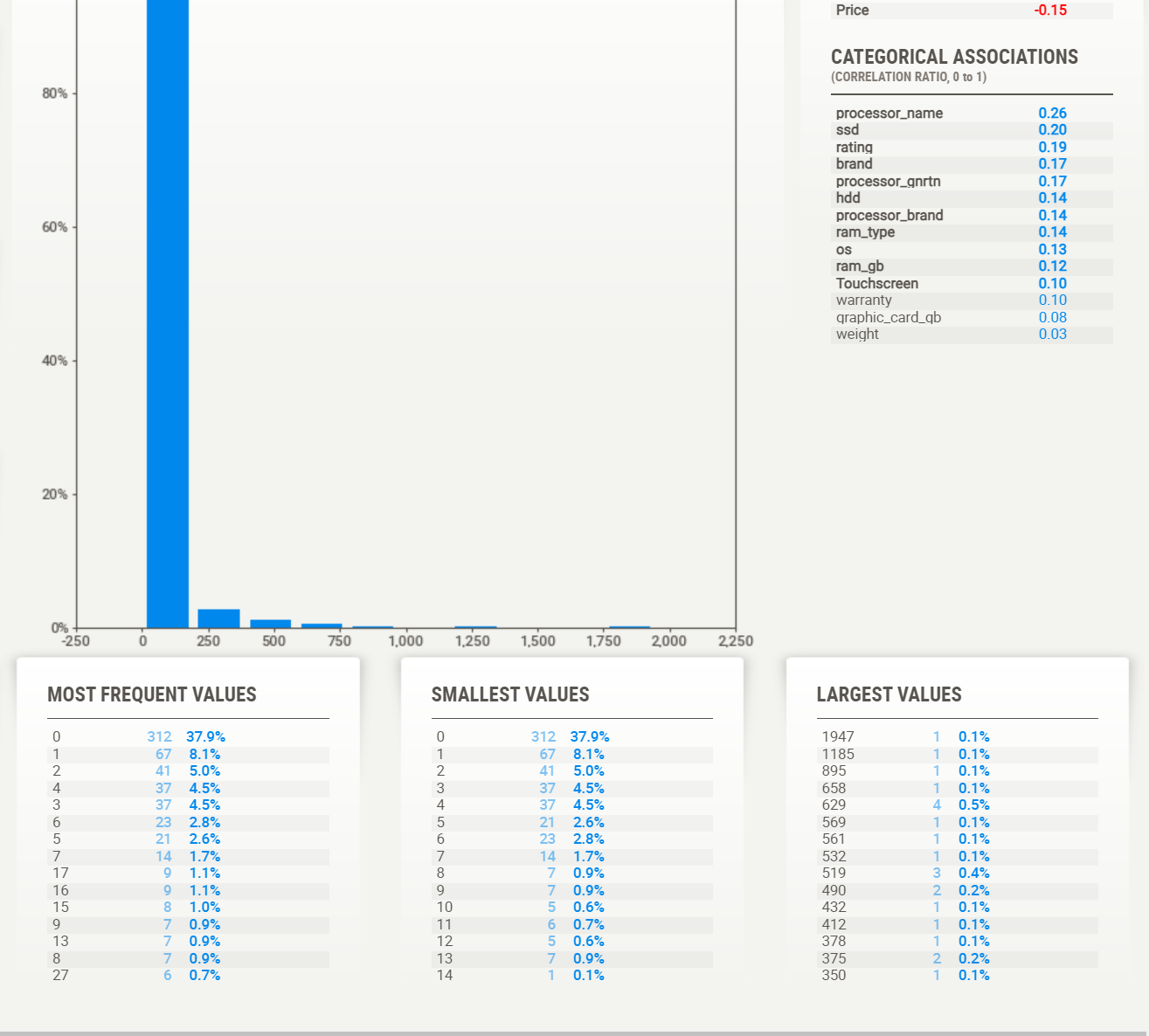
**

**

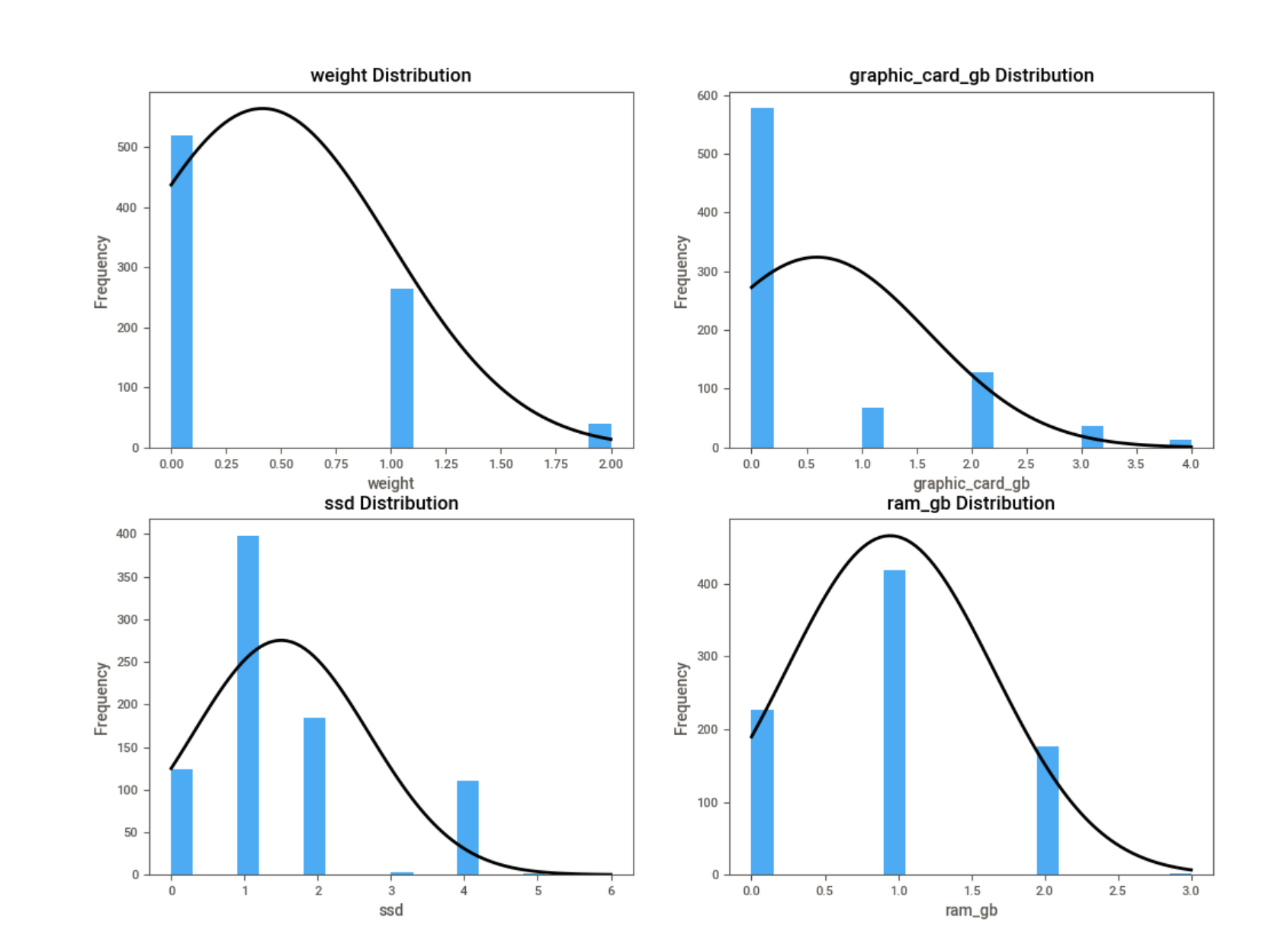
**

**

**

**

*Normal Distribution:*

**

**5. Codes**

import tkinter as tk

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import norm

from tkinter import ttk, filedialog, messagebox

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

import numpy as np

import sweetviz as sv

class LaptopInfoGUI:

def \_\_init\_\_(self, root):

self.file\_path = None

self.root = root

self.root.title("Laptop Pricing Analysis")

self.root.configure(bg="#9370DB")

self.dataset\_uploaded = False

#self.laptop\_info = {} #Initialize self.laptop\_info as an empty dictionary

self.root.style = ttk.Style()

self.root.style.configure("Navbar.TButton", background="orange", foreground="black")

self.selected\_options = {

"Laptop Brand": [],

"Processor Brand": [],

"Processor Name": [],

"Laptop Generation": [],

"RAM": [],

"RAM Type": [],

"SSD Size": [],

"HDD Size": [],

"OS": [],

"OS-bit": [],

"Graphic Card Size": [],

"Laptop Weight": [],

"Warranty": [],

"Touch Screen": [],

"MS-Office": []

}

self.create\_widgets()

def create\_widgets(self):

header\_label = ttk.Label(self.root, text="Laptop Price Predictor", font=("Arial Rounded MT Bold", 24, "bold"))

header\_label.pack(side="top", pady=10)

navbar\_frame = ttk.Frame(self.root)

navbar\_frame.pack(side="top", fill="x")

upload\_button = ttk.Button(navbar\_frame, text="Upload Dataset", command=self.upload\_dataset, style="Navbar.TButton")

upload\_button.pack(side="left", padx=10, pady=10)

notebook = ttk.Notebook(self.root)

notebook.pack(side="top", fill="both", expand=True, padx=10, pady=10)

questions = [

"brand", "processor\_brand", "processor\_name", "processor\_gnrtn", "ram\_gb",

"ram\_type", "ssd", "hdd", "os", "os\_bit", "graphic\_card\_gb", "weight",

"warranty", "Touchscreen", "msoffice"

]

self.entries = {}

for i, question in enumerate(questions):

frame = ttk.Frame(notebook, padding="10")

notebook.add(frame, text=question)

if question == "brand":

brands = ['ASUS' ,'Lenovo', 'Acer', 'Avita', 'HP', 'DELL' ,'MSI', 'APPLE']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top",anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Brand")

self.entries[question] = brand\_combobox

elif question == "processor\_brand":

brands = ['Intel', 'AMD' ,'M1']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Processor Brand")

self.entries[question] = brand\_combobox

elif question == "processor\_name":

brands = ['Core i3' ,'Core i5', 'Celeron Dual', 'Ryzen 5', 'Core i7', 'Core i9', 'M1','Pentium Quad', 'Ryzen 3', 'Ryzen 7', 'Ryzen 9']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Processor Name")

self.entries[question] = brand\_combobox

elif question == "processor\_gnrtn":

brands = ['4th','7th', '8th', '9th','10th', '11th' ,'12th']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Generation")

self.entries[question] = brand\_combobox

elif question == "ram\_gb":

brands = ['4 GB' ,'8 GB' ,'16 GB', '32 GB']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select RAM")

self.entries[question] = brand\_combobox

elif question == "ram\_type":

brands = ['DDR4', 'LPDDR4', 'LPDDR4X', 'DDR5', 'DDR3', 'LPDDR3']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select RAM Type")

self.entries[question] = brand\_combobox

elif question == "ssd":

brands = ['0 GB', '128 GB', '256 GB','512 GB' , '1024 GB' ,'2048 GB' ,'3072 GB']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select SSD Size")

self.entries[question] = brand\_combobox

elif question == "hdd":

brands = ['0 GB', '512 GB','1024 GB', '2048 GB']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select HDD Size")

self.entries[question] = brand\_combobox

elif question == "os":

brands = ['Windows', 'DOS' ,'Mac']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select OS")

self.entries[question] = brand\_combobox

elif question == "os\_bit":

brands = ['32-bit','64-bit']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select OS-bit")

self.entries[question] = brand\_combobox

elif question == "graphic\_card\_gb":

brands =['0 GB', '2 GB', '4 GB', '6 GB', '8 GB']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Graphic Card Size")

self.entries[question] = brand\_combobox

elif question == "weight":

brands =['Casual' ,'ThinNlight', 'Gaming']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Weight")

self.entries[question] = brand\_combobox

elif question == "warranty":

brands =['No warranty', '1 year' ,'2 years', '3 years']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Warranty Choice")

self.entries[question] = brand\_combobox

elif question == "Touchscreen":

brands =['Yes','No']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select Screen Choice")

self.entries[question] = brand\_combobox

elif question == "msoffice":

brands =['Yes','No']

brand\_combobox = ttk.Combobox(frame, values=brands)

brand\_combobox.pack(side="top", anchor="ne", padx=10, pady=5)

brand\_combobox.set("Select MS-Office Choice")

self.entries[question] = brand\_combobox

else:

entry = ttk.Entry(frame)

entry.pack(side="top", anchor="ne", padx=10, pady=5)

self.entries[question] = entry

save\_button = ttk.Button(self.root, text="Save", command=self.save\_info)

save\_button.place(x=1577, y=180, width=100, height=30)

plot\_button = ttk.Button(self.root, text="View Plots", command=self.plot\_display)

plot\_button.place(x=1371, y=180, width=100, height=30)

submit\_button = ttk.Button(self.root, text="Submit", command=self.submit\_info, state=tk.DISABLED)

submit\_button.place(x=1473, y=180, width=100, height=30)

self.submit\_button = submit\_button

footer\_label = ttk.Label(self.root, text="Powered by Fastians®", font=("TkDefaultFont", 11), foreground="black", background="#9370DB")

footer\_label.pack(side="bottom", pady=10)

def plot\_display(self):

df = pd.read\_csv(self.file\_path)

fd= pd.read\_csv(self.file\_path)

report = sv.analyze(fd)

report.show\_html("./report.html")

selected\_columns = ["weight", "graphic\_card\_gb", "ssd", "ram\_gb"]

plt.figure(figsize=(12, 10))

for i, column in enumerate(selected\_columns, 1):

plt.subplot(2, 2, i)

plt.hist(fd[column], bins=20, alpha=0.7)

plt.xlabel(column)

plt.ylabel('Frequency')

plt.tight\_layout()

plt.show()

plt.figure(figsize=(12, 10))

scatter\_plot\_frame = ttk.Frame(self.root)

scatter\_plot\_frame.pack(side="top", padx=10, pady=10)

for i, column in enumerate(selected\_columns, 1):

plt.subplot(2, 2, i)

counts = fd[column].value\_counts()

plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=140)

plt.axis('equal')

plt.title(f'{column} Distribution')

plt.tight\_layout()

plt.show()

for col in fd.columns:

if fd[col].dtype == 'object':

fd[col] = pd.factorize(fd[col])[0]

correlation\_matrix = fd.corr()

plt.figure(figsize=(12, 10))

heatmap = sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', annot\_kws={"size": 10})

plt.title('Correlation Heatmap of All Features')

plt.show()

new\_window = tk.Toplevel(self.root)

new\_window.title("Histograms with Normal Distributions")

new\_window.geometry("800x600") # Set the window size as needed

# Create a figure to hold the plots

figure = plt.Figure(figsize=(12, 10))

for i, column in enumerate(selected\_columns, 1):

hist\_plot = figure.add\_subplot(2, 2, i)

hist\_plot.hist(fd[column], bins=20, alpha=0.7)

hist\_plot.set\_xlabel(column)

hist\_plot.set\_ylabel('Frequency')

hist\_plot.set\_title(f'{column} Distribution')

# Fit and plot normal distribution curve

xmin, xmax = fd[column].min(), fd[column].max()

x = np.linspace(xmin, xmax, 100)

p = norm.pdf(x, fd[column].mean(), fd[column].std())

hist\_plot.plot(x, p \* len(fd[column]), 'k', linewidth=2)

canvas = FigureCanvasTkAgg(figure, master=new\_window)

canvas.draw()

canvas.get\_tk\_widget().pack()

# Display the window

new\_window.mainloop()

def save\_info(self):

for question, entry in self.entries.items():

selected\_option = entry.get()

if selected\_option and selected\_option != f"Select {question}":

if question not in self.selected\_options:

self.selected\_options[question] = [selected\_option]

elif selected\_option not in self.selected\_options[question]:

self.selected\_options[question].append(selected\_option)

print(f"{question}: {self.selected\_options[question]}")

else:

print(f"{question}: {self.selected\_options[question]} (Already saved)")

def upload\_dataset(self):

self.file\_path = filedialog.askopenfilename(filetypes=[("CSV files", "\*.csv")])

if self.file\_path:

try:

df = pd.read\_csv(self.file\_path)

self.dataset\_uploaded = True

self.submit\_button["state"] = tk.NORMAL

data\_text = tk.Text(self.root, wrap="none", height=2500, width=1000)

data\_text.pack(side="top", padx=10, pady=10)

data\_text.insert(tk.END, df.to\_string(index=False))

print("Dataset Uploaded Successfully!")

except Exception as e:

messagebox.showerror("Error", f"Error reading CSV file: {str(e)}")

def display\_result(self, laptop\_info):

result\_text = "Collected Laptop Information:\n\n"

for key, value in laptop\_info.items():

result\_text += f"{key}: {value}\n"

messagebox.showinfo("Laptop Information", result\_text)

def submit\_info(self):

df = pd.read\_csv(self.file\_path)

fd= pd.read\_csv(self.file\_path)

selected\_columns = [

"brand", "processor\_brand", "processor\_name", "processor\_gnrtn", "ram\_gb",

"ram\_type", "ssd", "hdd", "os", "os\_bit", "graphic\_card\_gb", "weight",

"warranty", "Touchscreen", "msoffice", "Price"

]

laptop\_info = {}

for question, entry in self.entries.items():

laptop\_info[question] = entry.get()

new\_data\_df = pd.DataFrame([laptop\_info])

df\_filtered = df[selected\_columns]

df\_filtered = pd.get\_dummies(df\_filtered)

target = df\_filtered["Price"]

features = df\_filtered.drop("Price", axis=1)

new\_data\_df = pd.get\_dummies(new\_data\_df)

new\_data\_df = new\_data\_df.reindex(columns=features.columns, fill\_value=0)

model = LinearRegression()

X = features

y = target

model.fit(X, y)

predicted\_price = model.predict(new\_data\_df)[0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

y\_pred = model.predict(X\_test)

accuracy = r2\_score(y\_test, y\_pred) \* 100

result\_text = "Collected Laptop Information:\n\n"

for key, value in laptop\_info.items():

result\_text += f"{key}: {value}\n"

result\_text += f"\nPredicted Price: ${predicted\_price:.2f}\nAccuracy (R-squared): {accuracy:.2f}%\n"

messagebox.showinfo("Laptop Information with Predictions", result\_text)

self.display\_result(laptop\_info)

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, y\_pred)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

#plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Predicted vs Actual Prices')

plt.tight\_layout()

root1 = tk.Tk()

app = LaptopInfoGUI(root1)

root1.mainloop()

**6. Conclusion**

In conclusion, our project successfully addressed the problem of predicting laptop prices based on essential parameters. The developed linear regression model exhibited robust performance, providing accurate predictions for a diverse range of laptops. This tool can be valuable for both consumers and sellers in estimating fair prices and making informed decisions.

Future work may involve exploring more advanced machine learning techniques, incorporating additional features, and expanding the dataset for even more accurate predictions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*