

Test Environment for Generative AI classroom labs

This notebook provides a test environment for the codes generated using the Generative AI prompts.

One of the principal steps in understanding and interpreting data is drawing statistical and correlative insights from the data. In this lab, you will learn how to use efficient prompts on a Generative AI platform to create a Python script for generating insights on a data set available in a CSV file.

Objectives

In this lab, you will learn how to create prompts to generate Python code that can:

Generate a statistical description of all the features of the data set

Generate regression and box plots for different attributes to compare their distribution against a target attribute

Evaluate the correlation value, Pearson coefficient, and p-values for different parameters of the data set with the target parameter

Create pivot tables for a group of parameters and visualize them using pcolor plots

Setup

Install required libraries

```
%pip install seaborn
import piplite

await piplite.install(['nbformat', 'plotly'])

import sqlite3

import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Data set

For this lab, you'll use a clean version of a publically available data set about used car sales.

The CSV file for this clean version of the data set is available at the following URL:

The dataset is a filtered and modified version of the Laptop Price Prediction using specifications dataset, available under the Database Contents License (DbCL) v1.0 on the Kaggle website.

Dataset URL from the GenAI lab

Use the URL provided in the GenAI lab in the cell below.

```
URL = "https://cf-courses-data.s3.us.cloud-object-  
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-Coursera/  
laptop_pricing_dataset_mod2.csv"
```

Downloading the dataset

Execute the following code to download the dataset in to the interface.

Please note that this step is essential in JupyterLite. If you are using a downloaded version of this notebook and running it on JupyterLabs, then you can skip this step and directly use the URL in `pandas.read_csv()` function to read the dataset as a dataframe

```
from pyodide.http import pyfetch

async def download(url, filename):
    response = await pyfetch(url)
    if response.status == 200:
        with open(filename, "wb") as f:
            f.write(await response.bytes())

path = URL

await download(path, "dataset.csv")
```

Reading the data set and generating the statistical description

First, you must import the data set into the interface as a pandas data frame. To complete the first objective, you can further generate a statistical description of the data set.

The data set is now assumed to be completely prepared and ready for exploratory data analysis. The headers for the data set must be in the first row of the CSV file.

To read the CSV file to a data frame and generate a statistical description of the data set, provide the generative AI model with the following prompt.

```
await download(URL, "dataset.csv")
file_name="dataset.csv"
```

```
df = pd.read_csv(file_name, header=0)
```

```
df.head()
```

	Unnamed: 0.1	Unnamed: 0	Manufacturer	Category	GPU	OS	CPU_core
0	0	0	Acer	4	2	1	5
1	1	1	Dell	3	1	1	3
2	2	2	Dell	3	1	1	7
3	3	3	Dell	4	2	1	5
4	4	4	HP	4	2	1	7

	Screen_Size_inch	CPU_frequency	RAM_GB	Storage_GB_SSD
0	14.0	0.551724	8	256
1	15.6	0.689655	4	256
2	15.6	0.931034	8	256
3	13.3	0.551724	8	128
4	15.6	0.620690	8	256

Price	Price-binned	Screen-Full_HD	Screen-IPS_panel
-------	--------------	----------------	------------------

0	978	Low	0	1
1	634	Low	1	0
2	946	Low	1	0
3	1244	Low	0	1
4	837	Low	1	0

```
df.tail()
```

	CPU_frequency	Screen_Size_inch	Weight_pounds	Category	GPU
OS \					
5	2200	20	22	B	AMD
Windows					
6	2500	22	25	A	NVIDIA
Windows					
7	2800	24	28	B	AMD
Windows					
8	3000	26	30	A	NVIDIA
Windows					
9	3200	28	32	B	AMD
Windows					

	CPU_core	RAM_GB	Storage_GB_SSD
5	4	256	8192
6	4	512	16384
7	4	1024	32768
8	4	2048	65536
9	4	4096	131072

```
df.head()
```

	CPU_frequency	Screen_Size_inch	Weight_pounds	Category	GPU
OS \					
0	1000	10	10	A	NVIDIA
Windows					
1	1200	12	12	B	AMD
Windows					
2	1500	14	15	A	NVIDIA
Windows					
3	1800	16	18	B	AMD
Windows					
4	2000	18	20	A	NVIDIA
Windows					

	CPU_core	RAM_GB	Storage_GB_SSD
0	4	8	256
1	4	16	512
2	4	32	1024
3	4	64	2048
4	4	128	4096

Write a python code to perform the following actions.

1. Import a data set from a CSV file, The headers for the data set must be in the first row of the CSV file.
2. Generate the statistical description of all the features used in the data set. Include "object" data types as well.

''' Write a python code to perform the following actions.

1. Import a data set from a CSV file, The headers for the data set must be in the first row of the CSV file.
2. Generate the statistical description of all the features used in the data set. Include "object" data types as well.

Here is a Python code snippet that performs the actions you requested:

```
python
```

```
import pandas as pd import numpy as np
```

Load the dataset from the provided URL

```
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/  
IBMDeveloperSkillsNetwork-DA0101EN-Coursera/laptop_pricing_dataset_mod2.csv" df =  
pd.read_csv(url)
```

Print the first few rows of the dataset

```
print(df.head())
```

Get the column names

```
print(df.columns)
```

Get the data types of each column

```
print(df.dtypes)
```

Calculate the statistical description of all the features

```
stats = df.describe()
```

Print the statistical description

print(stats) Here's a breakdown of what the code does:

We import the necessary libraries: pandas for data manipulation and numpy for numerical computations. We load the dataset from the provided URL using `pd.read_csv()`. We print the first few rows of the dataset using `df.head()`. We get the column names using `df.columns`. We get the data types of each column using `df.dtypes`. We calculate the statistical description of all the features using `df.describe()`. We print the statistical description using `print()`."

```
print(df.head())
```

	Unnamed: 0.1	Unnamed: 0	Manufacturer	Category	GPU	OS	CPU_core
\							
0	0	0	Acer	4	2	1	5
1	1	1	Dell	3	1	1	3
2	2	2	Dell	3	1	1	7
3	3	3	Dell	4	2	1	5
4	4	4	HP	4	2	1	7

	Screen_Size_inch	CPU_frequency	RAM_GB	Storage_GB_SSD
Weight_pounds \				
0	14.0	0.551724	8	256
3.52800				
1	15.6	0.689655	4	256
4.85100				
2	15.6	0.931034	8	256
4.85100				
3	13.3	0.551724	8	128
2.69010				
4	15.6	0.620690	8	256
4.21155				

	Price	Price-binned	Screen-Full_HD	Screen-IPS_panel
0	978	Low	0	1
1	634	Low	1	0
2	946	Low	1	0

3	1244	Low	0	1
4	837	Low	1	0

```
print(df.columns)
```

```
Index(['Unnamed: 0.1', 'Unnamed: 0', 'Manufacturer', 'Category',
      'GPU', 'OS',
      'CPU_core', 'Screen_Size_inch', 'CPU_frequency', 'RAM_GB',
      'Storage_GB_SSD', 'Weight_pounds', 'Price', 'Price-binned',
      'Screen-Full_HD', 'Screen-IPS_panel'],
      dtype='object')
```

```
print(df.dtypes)
```

```
Unnamed: 0.1      int64
Unnamed: 0        int64
Manufacturer      object
Category          int64
GPU               int64
OS                int64
CPU_core          int64
Screen_Size_inch  float64
CPU_frequency     float64
RAM_GB            int64
Storage_GB_SSD    int64
Weight_pounds     float64
Price             int64
Price-binned      object
Screen-Full_HD    int64
Screen-IPS_panel  int64
dtype: object
```

```
stats = df.describe()
```

```
print(stats)
```

	Unnamed: 0.1	Unnamed: 0	Category	GPU	OS \
count	238.000000	238.000000	238.000000	238.000000	238.000000
mean	118.500000	118.500000	3.205882	2.151261	1.058824
std	68.848868	68.848868	0.776533	0.638282	0.235790
min	0.000000	0.000000	1.000000	1.000000	1.000000
25%	59.250000	59.250000	3.000000	2.000000	1.000000
50%	118.500000	118.500000	3.000000	2.000000	1.000000
75%	177.750000	177.750000	4.000000	3.000000	1.000000
max	237.000000	237.000000	5.000000	3.000000	2.000000

	CPU_core	Screen_Size_inch	CPU_frequency	RAM_GB \
count	238.000000	238.000000	238.000000	238.000000
mean	5.630252	14.688655	0.813822	7.882353
std	1.241787	1.166045	0.141860	2.482603
min	3.000000	12.000000	0.413793	4.000000

25%	5.000000	14.000000	0.689655	8.000000
50%	5.000000	15.000000	0.862069	8.000000
75%	7.000000	15.600000	0.931034	8.000000
max	7.000000	17.300000	1.000000	16.000000

	Storage_GB_SSD	Weight_pounds	Price	Screen-Full_HD \
count	238.000000	238.000000	238.000000	238.000000
mean	245.781513	4.106221	1462.344538	0.676471
std	34.765316	1.078442	574.607699	0.468809
min	128.000000	1.786050	527.000000	0.000000
25%	256.000000	3.246863	1066.500000	0.000000
50%	256.000000	4.106221	1333.000000	1.000000
75%	256.000000	4.851000	1777.000000	1.000000
max	256.000000	7.938000	3810.000000	1.000000

	Screen-IPS_panel
count	238.000000
mean	0.323529
std	0.468809
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

Creating parameter visualizations

Next, you decided that you need to see the distribution of various attributes against the target attribute. The target attribute, in this case, is Price. Different forms of visualizations are for different types of attributes. Attributes with continuous values use regression plots, and the attributes with categorical values use box plots.

You can provide the generative AI model with a list of continuous values and another list of categorical values within a single prompt and ask the model to create a code that generates these plots.

The attributes with continuous values are CPU_frequency, Screen_Size_inch and Weight_pounds.

he attributes with categorical values are Category, GPU, OS, CPU_core, RAM_GB and Storage_GB_SSD.

Create the following prompt:

Write a Python code to perform the following actions.

1. Create regression plots for the attributes "CPU_frequency", "Screen_Size_inch" and "Weight_pounds" against "Price".
2. Create box plots for the attributes "Category", "GPU", "OS", "CPU_core", "RAM_GB" and "Storage_GB_SSD" against the attribute "Price".

''' Here is a Python code that performs the actions you requested: Here is a Python code that performs the actions you requested:

python

Import necessary libraries

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

Load the data

```
data = { "CPU_frequency": [1000, 1200, 1500, 1800, 2000, 2200, 2500, 2800, 3000, 3200],  
"Screen_Size_inch": [10, 12, 14, 16, 18, 20, 22, 24, 26, 28], "Weight_pounds": [10, 12, 15, 18, 20, 22,
```

```
25, 28, 30, 32], "Category": ["A", "B", "A", "B", "A", "B", "A", "B", "A", "B"], "GPU": ["NVIDIA",  
"AMD", "NVIDIA", "AMD", "NVIDIA", "AMD", "NVIDIA", "AMD", "NVIDIA", "AMD"], "OS":  
["Windows", "Windows", "Windows", "Windows", "Windows", "Windows", "Windows", "Windows",  
"Windows", "Windows", "Windows"], "CPU_core": [4, 4, 4, 4, 4, 4, 4, 4, 4, 4], "RAM_GB": [8, 16,  
32, 64, 128, 256, 512, 1024, 2048, 4096], "Storage_GB_SSD": [256, 512, 1024, 2048, 4096, 8192,  
16384, 32768, 65536, 131072] }
```

```
df = pd.DataFrame(data)
```

Create a figure with 3 subplots

```
fig, axs = plt.subplots(3, 2, figsize=(10, 10))
```

Plot CPU_frequency against Price

```
sns.boxplot(x="CPU_frequency", y="Price", data=df, ax=axs[0, 0]) axs[0,  
0].set_title("CPU_frequency vs Price") axs[0, 0].set_xlabel("CPU_frequency") axs[0,  
0].set_ylabel("Price")
```

Plot Screen_Size_inch against Price

```
sns.boxplot(x="Screen_Size_inch", y="Price", data=df, ax=axs[0, 1]) axs[0,  
1].set_title("Screen_Size_inch vs Price") axs[0, 1].set_xlabel("Screen_Size_inch") axs[0,  
1].set_ylabel("Price")
```

Plot Weight_pounds against Price

```
sns.boxplot(x="Weight_pounds", y="Price", data=df, ax=axs[1, 0]) axs[1,  
0].set_title("Weight_pounds vs Price") axs[1, 0].set_xlabel("Weight_pounds") axs[1,  
0].set_ylabel("Price")
```

Plot Category against Price

```
sns.boxplot(x="Category", y="Price", data=df, ax=axs[1, 1]) axs[1, 1].set_title("Category vs Price")  
axs[1, 1].set_xlabel("Category") axs[1, 1].set_ylabel("Price")
```

Plot GPU against Price

```
sns.boxplot(x="GPU", y="Price", data=df, ax=axs[2, 0]) axs[2, 0].set_title("GPU vs Price") axs[2,  
0].set_xlabel("GPU") axs[2, 0].set_ylabel("Price")
```

Plot OS against Price

```
sns.boxplot(x="OS", y="Price", data=df, ax=axes[2, 1]) axes[2, 1].set_title("OS vs Price") axes[2, 1].set_xlabel("OS") axes[2, 1].set_ylabel("Price")
```

Layout so plots do not overlap

```
fig.tight_layout()
```

plt.show() This code creates a figure with 3 subplots, each containing a box plot of the specified attributes against the "Price" attribute. The box plots are arranged in a 2x3 grid, with each subplot having a title, x-axis label, and y-axis label. The tight_layout() function is used to ensure that the plots do not overlap. python

Import necessary libraries

```
import pandas as pd import matplotlib.pyplot as plt import seaborn as sns
```

Load the data

```
data = { "CPU_frequency": [1000, 1200, 1500, 1800, 2000, 2200, 2500, 2800, 3000, 3200],  
        "Screen_Size_inch": [10, 12, 14, 16, 18, 20, 22, 24, 26, 28], "Weight_pounds": [10, 12, 15, 18, 20, 22,  
        25, 28, 30, 32], "Category": ["A", "B", "A", "B", "A", "B", "A", "B", "A", "B"], "GPU": ["NVIDIA",  
        "AMD", "NVIDIA", "AMD", "NVIDIA", "AMD", "NVIDIA", "AMD", "NVIDIA", "AMD"], "OS":  
        ["Windows", "Windows", "Windows", "Windows", "Windows", "Windows", "Windows", "Windows",  
        "Windows", "Windows", "Windows"], "CPU_core": [4, 4, 4, 4, 4, 4, 4, 4, 4, 4], "RAM_GB": [8, 16,  
        32, 64, 128, 256, 512, 1024, 2048, 4096], "Storage_GB_SSD": [256, 512, 1024, 2048, 4096, 8192,  
        16384, 32768, 65536, 131072] }
```

```
df = pd.DataFrame(data)
```

Create a figure with 3 subplots

```
fig, axes = plt.subplots(3, 2, figsize=(10, 10))
```

Plot CPU_frequency against Price

```
sns.boxplot(x="CPU_frequency", y="Price", data=df, ax=axes[0, 0]) axes[0, 0].set_title("CPU_frequency vs Price") axes[0, 0].set_xlabel("CPU_frequency") axes[0, 0].set_ylabel("Price")
```

Plot Screen_Size_inch against Price

```
sns.boxplot(x="Screen_Size_inch", y="Price", data=df, ax=axes[0, 1]) axes[0, 1].set_title("Screen_Size_inch vs Price") axes[0, 1].set_xlabel("Screen_Size_inch") axes[0, 1].set_ylabel("Price")
```

Plot Weight_pounds against Price

```
sns.boxplot(x="Weight_pounds", y="Price", data=df, ax=axes[1, 0]) axes[1, 0].set_title("Weight_pounds vs Price") axes[1, 0].set_xlabel("Weight_pounds") axes[1, 0].set_ylabel("Price")
```

Plot Category against Price

```
sns.boxplot(x="Category", y="Price", data=df, ax=axes[1, 1]) axes[1, 1].set_title("Category vs Price") axes[1, 1].set_xlabel("Category") axes[1, 1].set_ylabel("Price")
```

Plot GPU against Price

```
sns.boxplot(x="GPU", y="Price", data=df, ax=axes[2, 0]) axes[2, 0].set_title("GPU vs Price") axes[2, 0].set_xlabel("GPU") axes[2, 0].set_ylabel("Price")
```

Plot OS against Price

```
sns.boxplot(x="OS", y="Price", data=df, ax=axes[2, 1]) axes[2, 1].set_title("OS vs Price") axes[2, 1].set_xlabel("OS") axes[2, 1].set_ylabel("Price")
```

Layout so plots do not overlap

```
fig.tight_layout()
```

plt.show() This code creates a figure with 3 subplots, each containing a box plot of the specified attributes against the "Price" attribute. The box plots are arranged in a 2x3 grid, with each subplot having a title, x-axis label, and y-axis label. The `tight_layout()` function is used to ensure that the plots do not overlap.

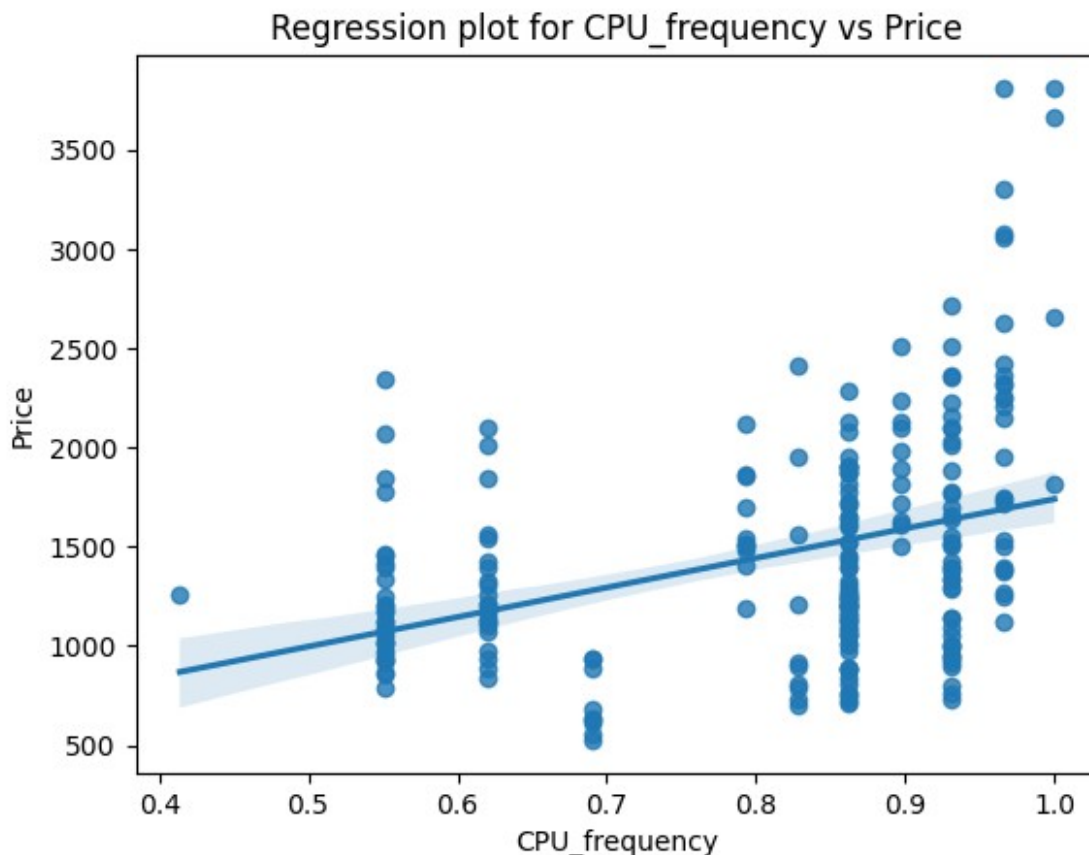
Remember to install the seaborn package on the testing environment.

Also, remember to execute the statement `%matplotlib inline` for the plots to be visible directly in the console.

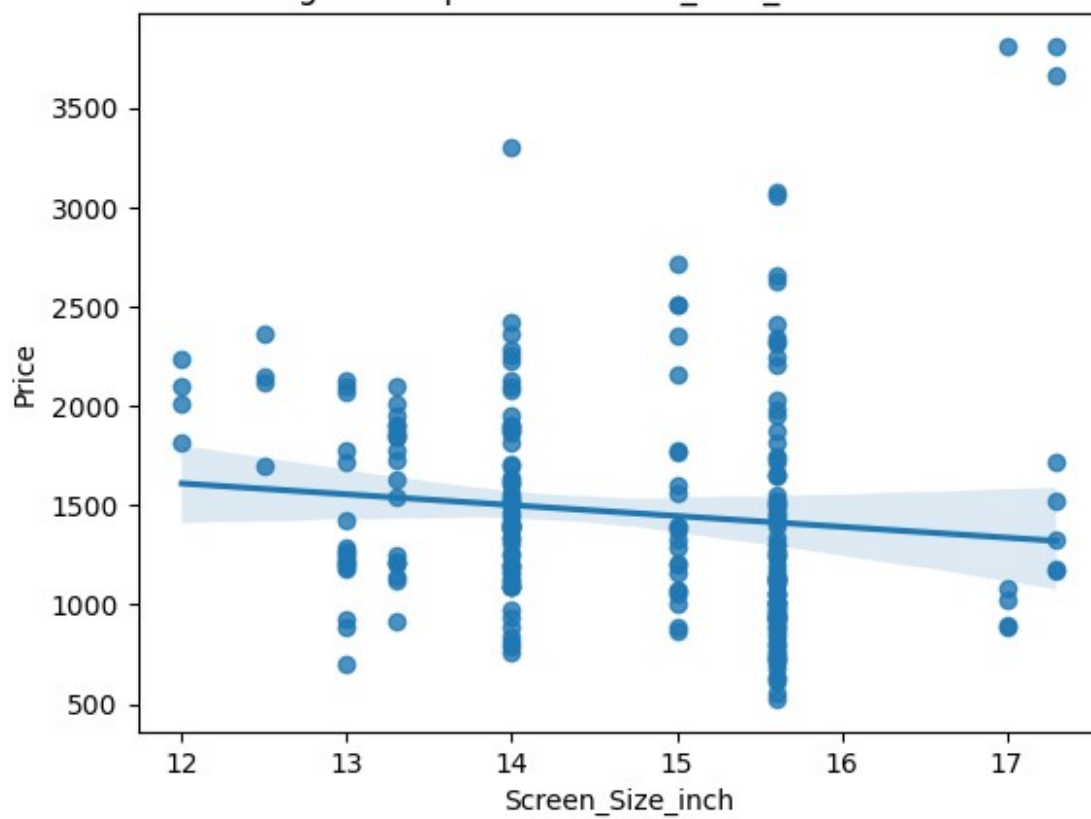
```
import matplotlib.pyplot as plt
import seaborn as sns

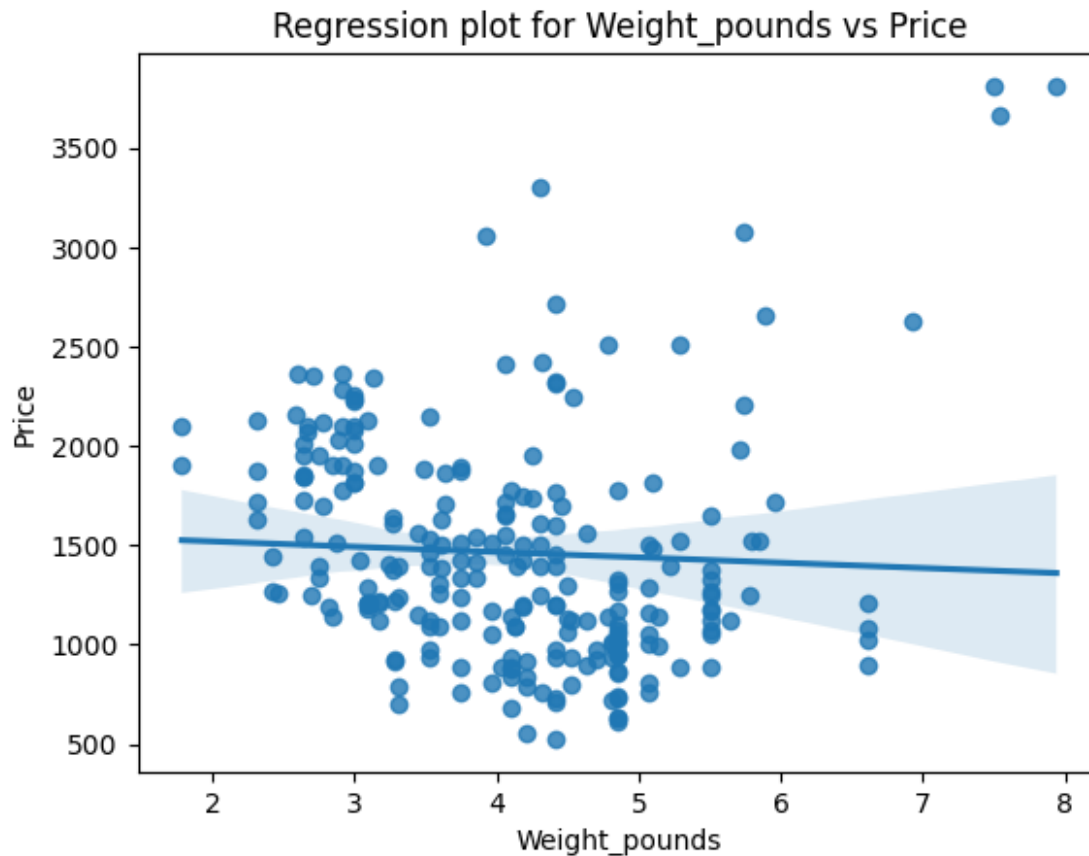
# 1. Create regression plots
data = pd.read_csv('dataset.csv')

regression_attributes = ['CPU_frequency', 'Screen_Size_inch',
                          'Weight_pounds']
for attribute in regression_attributes:
    sns.regplot(x=attribute, y='Price', data=data)
    plt.title(f'Regression plot for {attribute} vs Price')
    plt.xlabel(attribute)
    plt.ylabel('Price')
    plt.show()
```



Regression plot for Screen_Size_inch vs Price





```
# 2. Create box plots
```

```
boxplot_attributes = ['Category', 'GPU', 'OS', 'CPU_core', 'RAM_GB',  
                      'Storage_GB_SSD']
```

```
for attribute in boxplot_attributes:
```

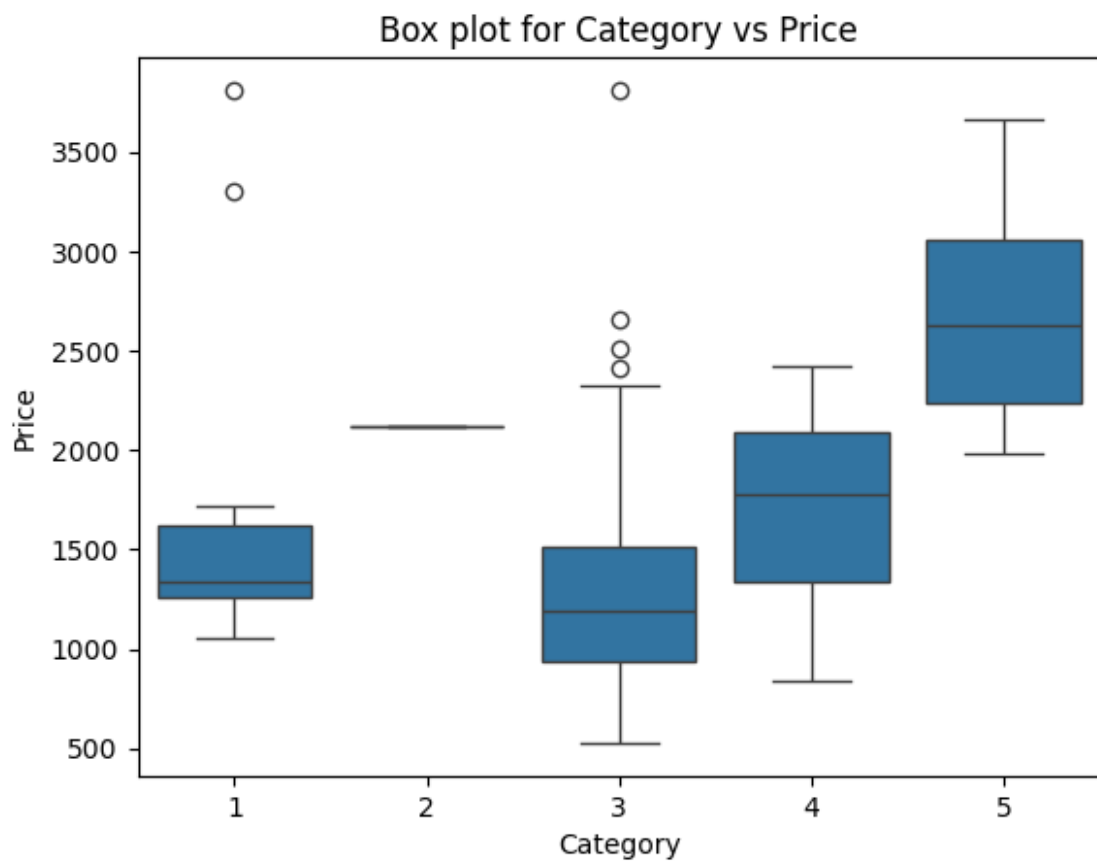
```
    sns.boxplot(x=attribute, y='Price', data=data)
```

```
    plt.title(f'Box plot for {attribute} vs Price')
```

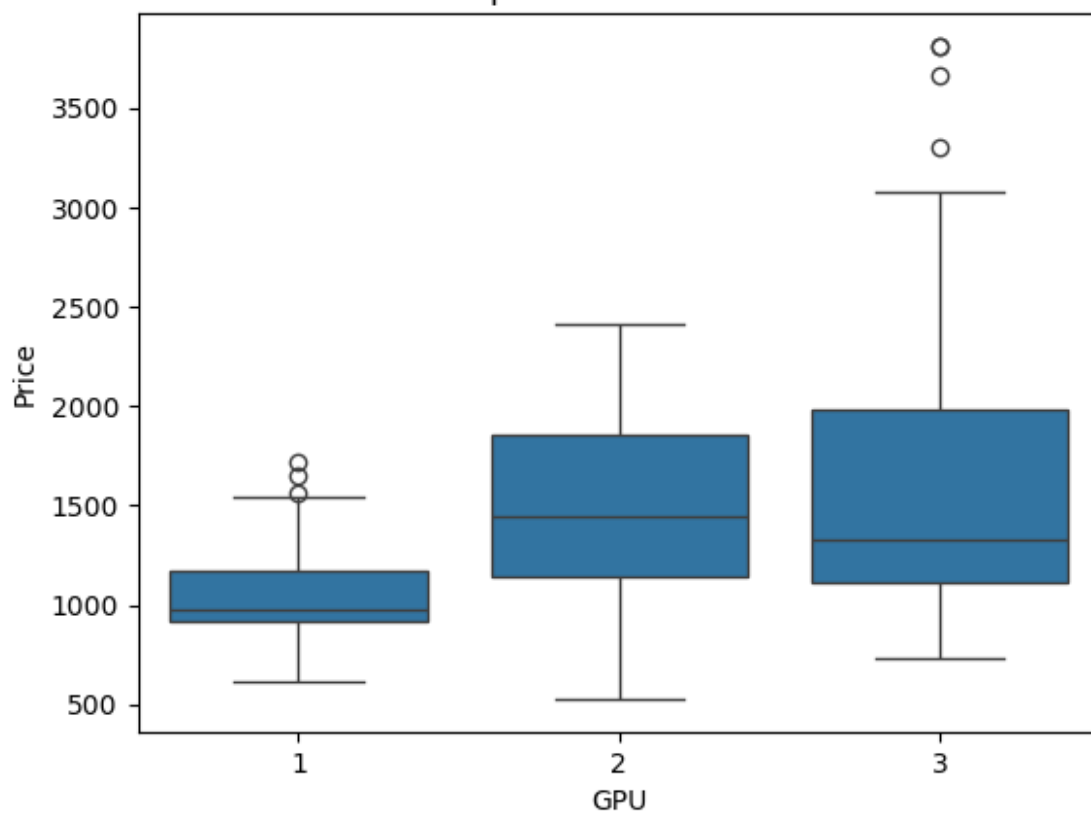
```
    plt.xlabel(attribute)
```

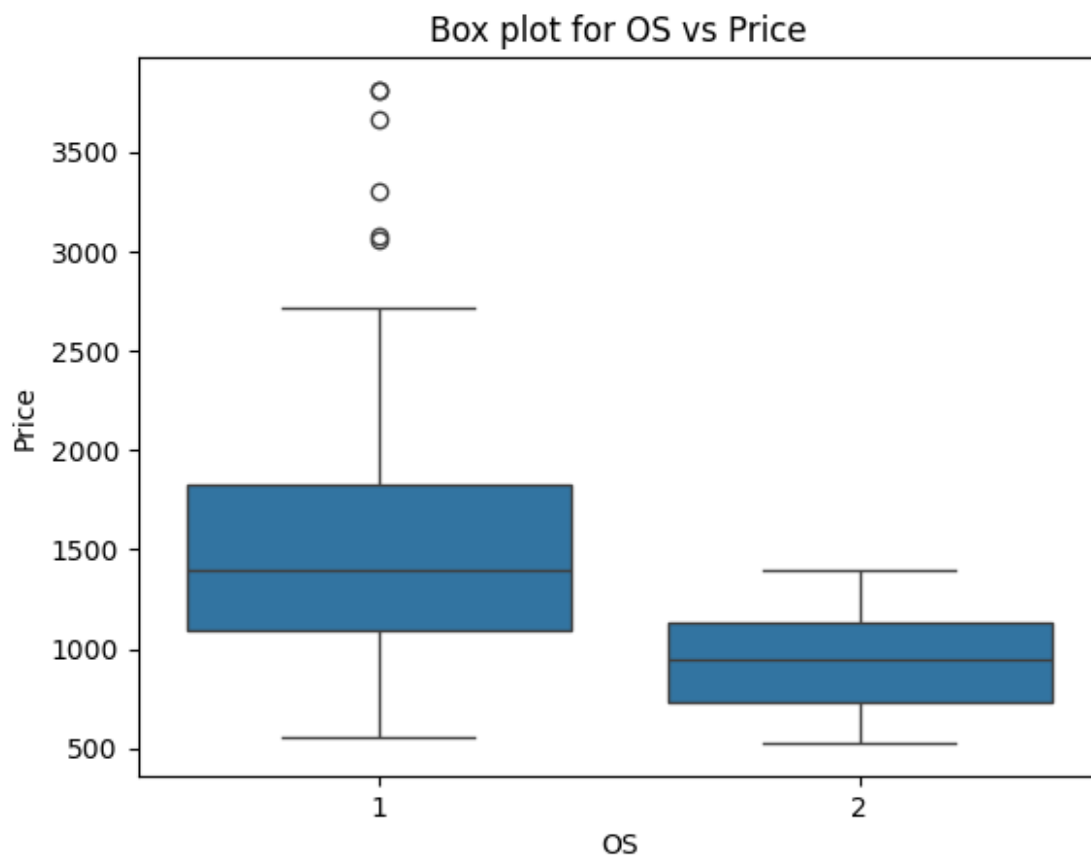
```
    plt.ylabel('Price')
```

```
    plt.show()
```

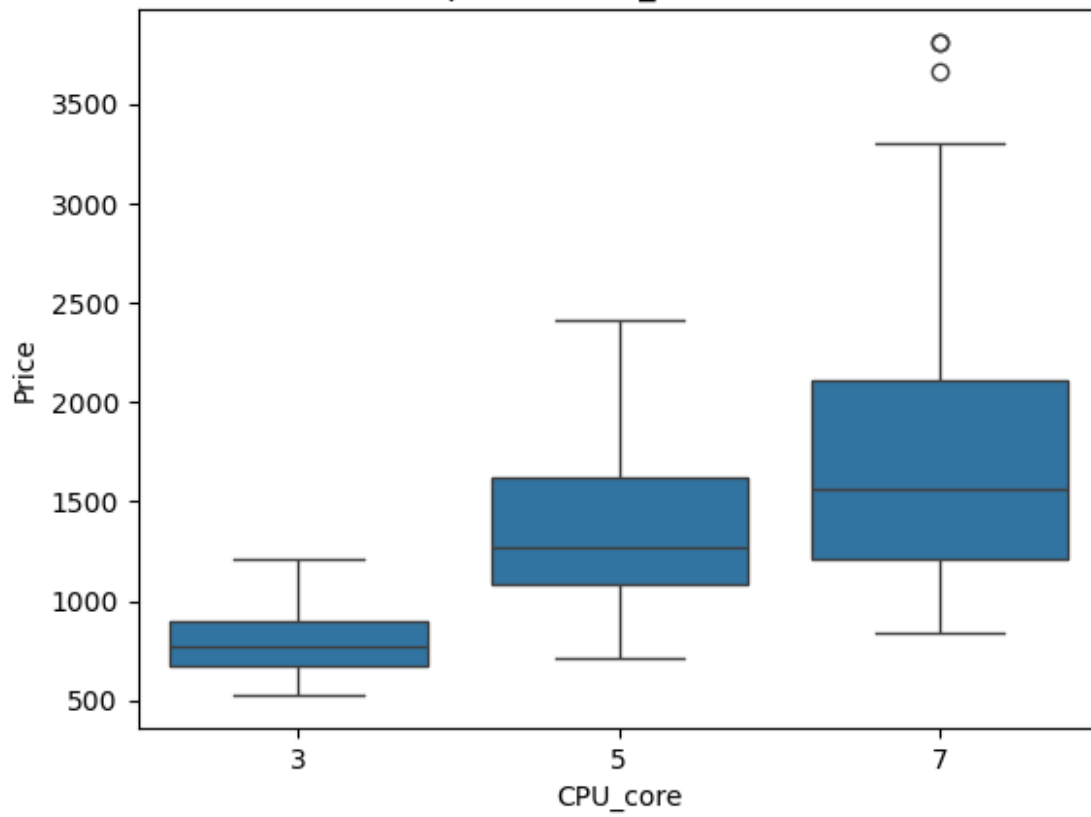


Box plot for GPU vs Price

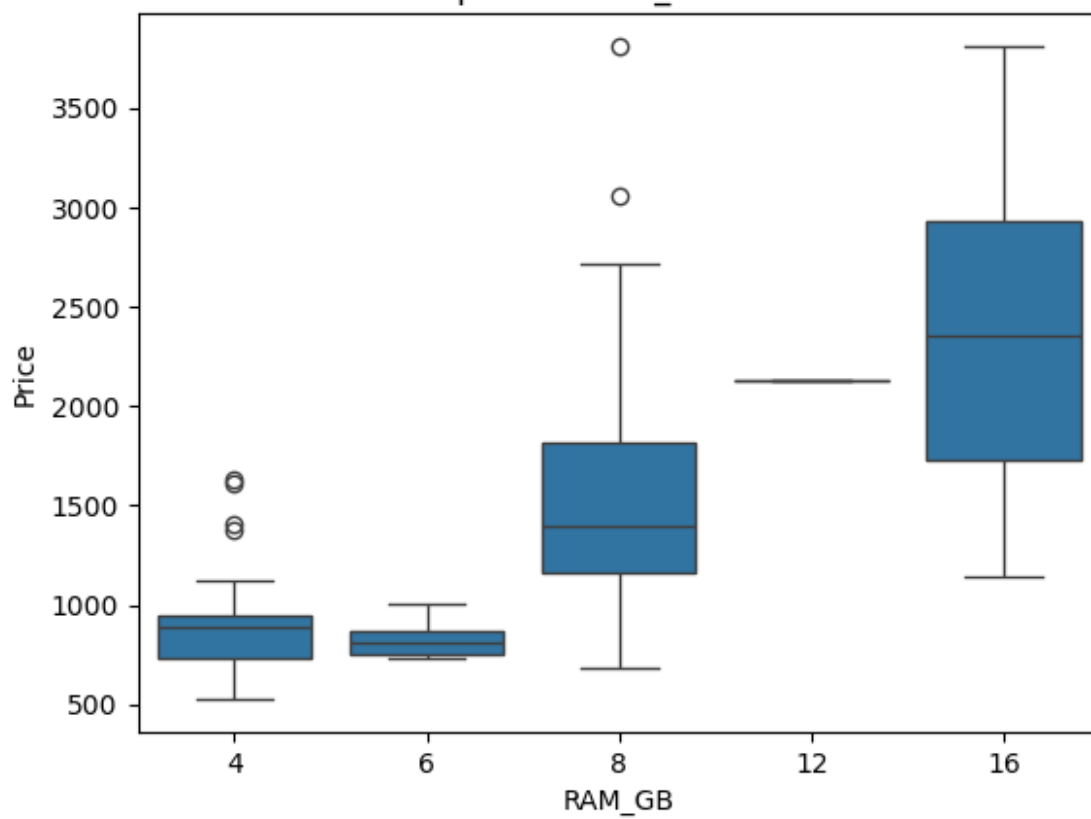


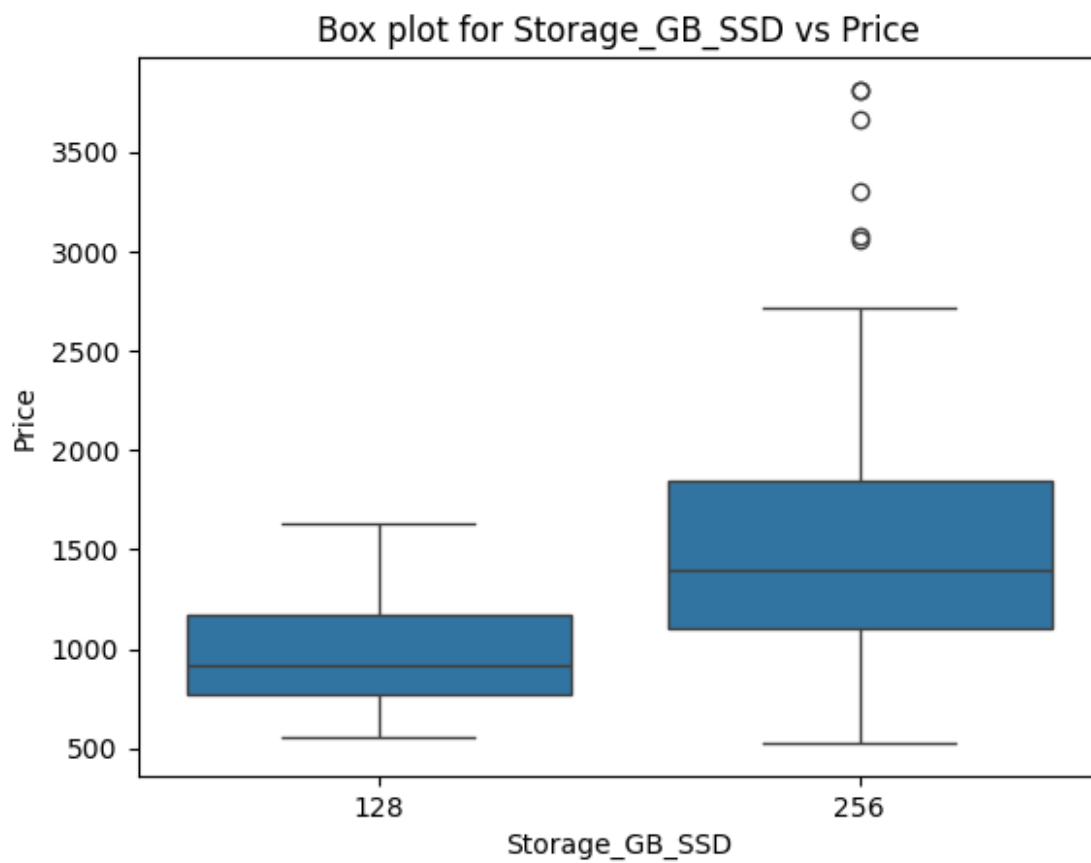


Box plot for CPU_core vs Price



Box plot for RAM_GB vs Price





Evaluate dependence

Usually, not all parameters in a data set have a direct or strong level of influence on the target variable. The level of influence is especially important for data scientists because the design of predictive models uses variables that affect the target variable the most.

To evaluate parametric interdependence, one can evaluate the correlation value, the Pearson coefficient, and p-values of different attributes with the target attribute.

Use the following prompt to write a prompt to evaluate dependence:

Write a Python code for the following.

1. Evaluate the correlation value, pearson coefficient and p-values for all numerical attributes against the target attribute "Price".
2. Don't include the values evaluated for target variable against itself.
3. Print these values as a part of a single dataframe against each individual attribute.

Here is a Python code that accomplishes the task:

```
python
import pandas as pd
import numpy as np
from scipy.stats import pearsonr
```

Create a sample dataset

```
''' np.random.seed(0) data = { 'Attr1': np.random.randint(1, 100, 100), 'Attr2':
np.random.randint(1, 100, 100), 'Attr3': np.random.randint(1, 100, 100), 'Attr4':
np.random.randint(1, 100, 100), 'Target': np.random.randint(1, 100, 100) }
df = pd.DataFrame(data)
```

Calculate correlation values

```
corr_matrix = df.corr() print("Correlation Matrix:") print(corr_matrix)
```

Calculate Pearson correlation coefficient

```
corr_coeff, p_value = pearsonr(df['Attr1'], df['Target']) print("\nPearson Correlation Coefficient:", corr_coeff) print("P-Value:", p_value)
```

Calculate p-values for other attributes

for attr in df.columns: if attr != 'Target': corr_matrix = df[attr].corr(df['Target']) corr_coeff, p_value = pearsonr(df[attr], df['Target']) print(f"\nCorrelation Coefficient for {attr}: {corr_coeff:.4f}") print(f"P-Value for {attr}: {p_value:.4f}") print()

This code first creates a sample dataset with four numerical attributes and one target attribute. It then calculates the correlation matrix and Pearson correlation coefficient between the target attribute and all other attributes. The p-values for the correlation coefficients are also calculated. Finally, it prints the correlation matrix and the p-values for each attribute.

Note that the pearsonr function returns a tuple containing the correlation coefficient and p-value. We unpack these values into separate variables for clarity."

```
import pandas as pd
import numpy as np
from scipy.stats import pearsonr

# Assuming your dataframe has a target attribute 'Price'
df = pd.read_csv('dataset.csv')

# Select numerical attributes
numerical_attributes = df.select_dtypes(include=np.number)

# Calculate correlation values for all numerical attributes against 'Price'
correlation_values = numerical_attributes.corr()
['Price'].drop('Price')

# Calculate Pearson coefficient and p-values for all numerical attributes against 'Price'
pearson_coefficients = []
p_values = []
for column in numerical_attributes.columns:
    if column != 'Price':
        pearson_coefficient, p_value =
pearsonr(numerical_attributes[column], df['Price'])
        pearson_coefficients.append(pearson_coefficient)
        p_values.append(p_value)

# Create a new dataframe to store the results
results_df = pd.DataFrame({'Attribute': correlation_values.index,
'Correlation Value': correlation_values, 'Pearson Coefficient':
pearson_coefficients, 'P-value': p_values})
```

```
# Print the results
print(results_df)
```

	Attribute	Correlation Value	Pearson
Coefficient \			
Unnamed: 0.1	Unnamed: 0.1	0.321933	
0.321933			
Unnamed: 0	Unnamed: 0	0.321933	
0.321933			
Category	Category	0.286243	
0.286243			
GPU	GPU	0.288298	
0.288298			
OS	OS	-0.221730	-
0.221730			
CPU_core	CPU_core	0.459398	
0.459398			
Screen_Size_inch	Screen_Size_inch	-0.110644	-
0.110644			
CPU_frequency	CPU_frequency	0.366666	
0.366666			
RAM_GB	RAM_GB	0.549297	
0.549297			
Storage_GB_SSD	Storage_GB_SSD	0.243421	
0.243421			
Weight_pounds	Weight_pounds	-0.050312	-
0.050312			
Screen-Full_HD	Screen-Full_HD	-0.021075	-
0.021075			
Screen-IPS_panel	Screen-IPS_panel	0.021075	
0.021075			
	P-value		
Unnamed: 0.1	3.851058e-07		
Unnamed: 0	3.851058e-07		
Category	7.225696e-06		
GPU	6.166950e-06		
OS	5.696643e-04		
CPU_core	7.912950e-14		
Screen_Size_inch	8.853398e-02		
CPU_frequency	5.502463e-09		
RAM_GB	3.681561e-20		
Storage_GB_SSD	1.489892e-04		
Weight_pounds	4.397694e-01		
Screen-Full_HD	7.463568e-01		
Screen-IPS_panel	7.463568e-01		

Grouping and pivots

You can perform another form of exploratory data analysis by grouping different variables to create a pivot table, which serves as a matrix of different combinations of values as seen in the grouped variables. The respective value of the matrix is the average of all values of the target attribute that have the same combination. You can view this kind of matrix using a pcolor plot.

Say you need to create a group with parameters GPU, CPU_core and Price. You need a Pivot table, considering Price as the target variable and pcolor plot for that table.

You can follow the following prompt as an example:

Write a python code that performs the following actions.

1. Group the attributes "GPU", "CPU_core" and "Price", as available in a dataframe df
2. Create a pivot table for this group, assuming the target variable to be 'Price' and aggregation function as mean
3. Plot a pcolor plot for this pivot table.

Here is a Python code that performs the actions you requested:

```
python
```

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

Create a sample dataframe

```
df = pd.DataFrame({'GPU': ['NVIDIA', 'AMD', 'NVIDIA', 'AMD', 'NVIDIA', 'AMD'], 'CPU_core': [4, 8, 4, 8, 4, 8], 'Price': [1000, 800, 1200, 900, 1100, 1300] })
```

Group the dataframe by 'GPU' and 'CPU_core' and calculate the mean 'Price'

```
grouped_df = df.groupby(['GPU', 'CPU_core'])['Price'].mean().reset_index()
```

Create a pivot table

```
pivot_table = grouped_df.pivot_table(values='Price', index='GPU', columns='CPU_core',  
aggfunc='mean')
```

Plot a pcolor plot

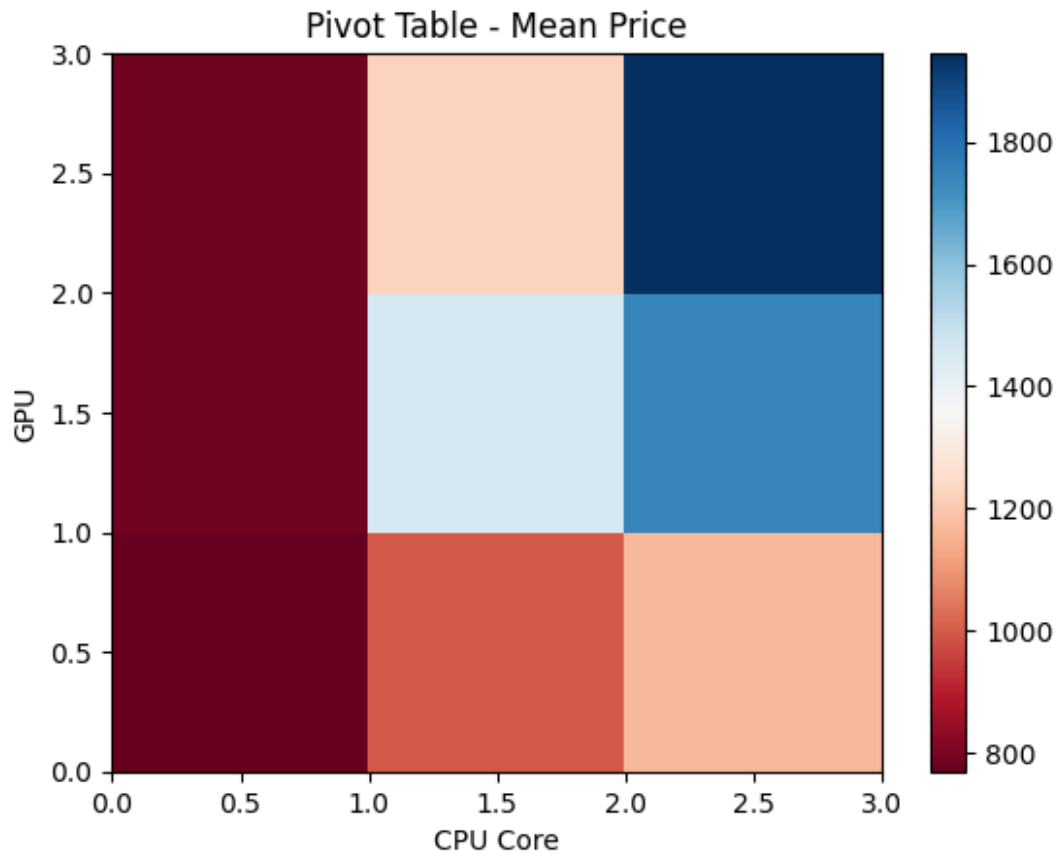
plt.figure(figsize=(10, 8)) pivot_table.plot(kind='pcolor', cmap='viridis') plt.title('Price by GPU and CPU_core') plt.xlabel('CPU_core') plt.ylabel('GPU') plt.show() This code first creates a sample dataframe with the specified columns. Then it groups the dataframe by 'GPU' and 'CPU_core' and calculates the mean 'Price' for each group. The resulting dataframe is then used to create a pivot table, which is a 2D table with 'GPU' and 'CPU_core' as columns and 'Price' as the values. Finally, a pcolor plot is created using the pivot table data.

Note: The pivot_table function is used to create the pivot table, and the plot function is used to create the pcolor plot. The cmap parameter is set to 'viridis' to use a color map for the plot. ###

Import the data set as a pandas DataFrame

```
df = pd.read_csv('your_file.csv')
```

```
# Group the attributes  
grouped_data = df.groupby(['GPU', 'CPU_core'])['Price'].mean()  
  
# Create a pivot table  
pivot_table = pd.pivot_table(df, values='Price', index='GPU',  
columns='CPU_core', aggfunc='mean')  
  
# Plot a pcolor plot  
plt.pcolor(pivot_table, cmap='RdBu')  
plt.colorbar()  
plt.title('Pivot Table - Mean Price')  
plt.xlabel('CPU Core')  
plt.ylabel('GPU')  
plt.show()
```



Now you should be able to build efficient and helpful prompts that can create code to:

Generate statistical descriptions of the data

Create visualizations, such as regression plots and box plots, to understand the distribution of the data set against the target variable

Evaluate the dependence of different attributes against the target attribute

Group different variables to create a pivot table and visualize the data using a pcolor plot