In [802…

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import plotly.express as px
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
import warnings
```

In [803…

```python
# Loading the csv file into the pandas dataframe

chip = pd.read_csv('chip_dataset.csv')
chip
```

Out[803…

	Unnamed: 0	Product	Type	Release Date	Process Size (nm)	TDP (W)	Die Size (mm^2)	Transistors (million)	Freq (MHz)	Foundry	Vendor	FP16 GFLOPS	FP32 GFLOPS	FP64 GFLOPS
0	0	AMD Athlon 64 3500+	CPU	2007-02-20	65.0	45.0	77.0	122.0	2200.0	Unknown	AMD	NaN	NaN	NaN
1	1	AMD Athlon 200GE	CPU	2018-09-06	14.0	35.0	192.0	4800.0	3200.0	Unknown	AMD	NaN	NaN	NaN
2	2	Intel Core i5-1145G7	CPU	2020-09-02	10.0	28.0	NaN	NaN	2600.0	Intel	Intel	NaN	NaN	NaN
3	3	Intel Xeon E5-2603 v2	CPU	2013-09-01	22.0	80.0	160.0	1400.0	1800.0	Intel	Intel	NaN	NaN	NaN
4	4	AMD Phenom II X4 980 BE	CPU	2011-05-03	45.0	125.0	258.0	758.0	3700.0	Unknown	AMD	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4849	4849	NVIDIA Quadro 3000M	GPU	2011-02-22	40.0	75.0	332.0	1950.0	450.0	TSMC	NVIDIA	NaN	432.0	36.0
4850	4850	Intel GMA 950	GPU	2005-06-01	90.0	7.0	NaN	NaN	250.0	Intel	Intel	NaN	NaN	NaN
4851	4851	NVIDIA GeForce GT 320M	GPU	2010-03-03	40.0	23.0	100.0	486.0	500.0	TSMC	NVIDIA	NaN	52.8	NaN
4852	4852	NVIDIA GeForce FX 5200	GPU	2003-03-06	150.0	NaN	65.0	29.0	250.0	TSMC	NVIDIA	NaN	NaN	NaN
4853	4853	NVIDIA GeForce 9300 SE	GPU	2008-06-01	65.0	NaN	86.0	210.0	540.0	TSMC	NVIDIA	NaN	20.8	NaN

4854 rows × 14 columns

In [804…

```python
# checking for nulls

chip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4854 entries, 0 to 4853
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             4854 non-null   int64  
 1   Product                4854 non-null   object 
 2   Type                   4854 non-null   object 
 3   Release Date           4854 non-null   object 
 4   Process Size (nm)      4845 non-null   float64
 5   TDP (W)                4228 non-null   float64
 6   Die Size (mm^2)        4139 non-null   float64
 7   Transistors (million)  4143 non-null   float64
 8   Freq (MHz)             4854 non-null   float64
 9   Foundry                4854 non-null   object 
 10  Vendor                 4854 non-null   object 
 11  FP16 GFLOPS            536 non-null    float64
 12  FP32 GFLOPS            1948 non-null   float64
 13  FP64 GFLOPS            1306 non-null   float64
dtypes: float64(8), int64(1), object(5)
memory usage: 531.0+ KB
```

In [805…

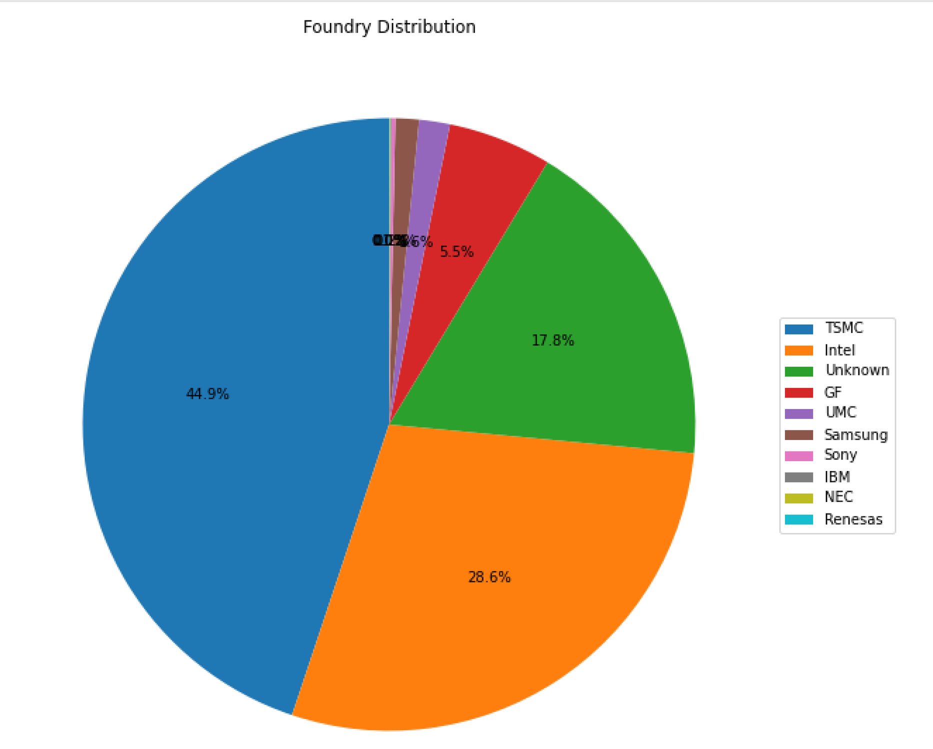
```python
# Dropping columns with too many null values

chip = chip.drop(['FP16 GFLOPS', 'FP32 GFLOPS', 'FP64 GFLOPS'], axis=1)
```

In [806…

```python
# Replace null values with column-wise average

chip = chip.fillna(chip.mean())
chip = chip.round(decimals=0).astype(object)
warnings.filterwarnings("ignore")
```

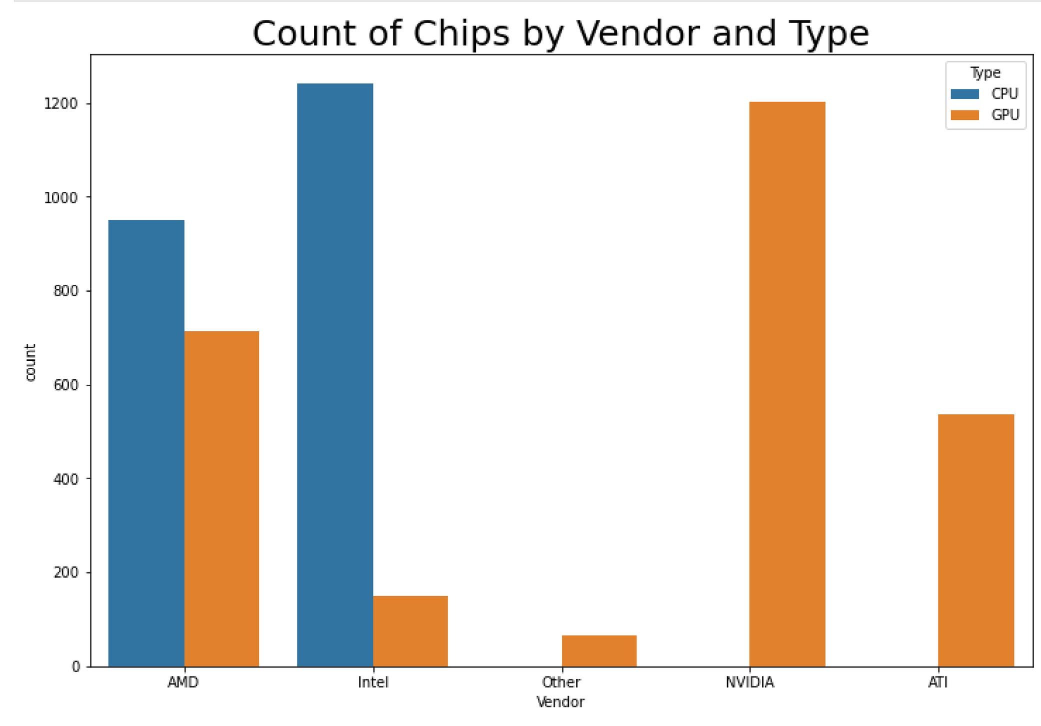
In [807…

```python
# Creating a pie chart

values = chip['Foundry'].value_counts()
names = values.index

plt.figure(figsize=(20, 10))
patches, _, _ = plt.pie(values, labels=None, autopct='%1.1f%%', startangle=90)

plt.title('Foundry Distribution')

plt.legend(patches, names, loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```



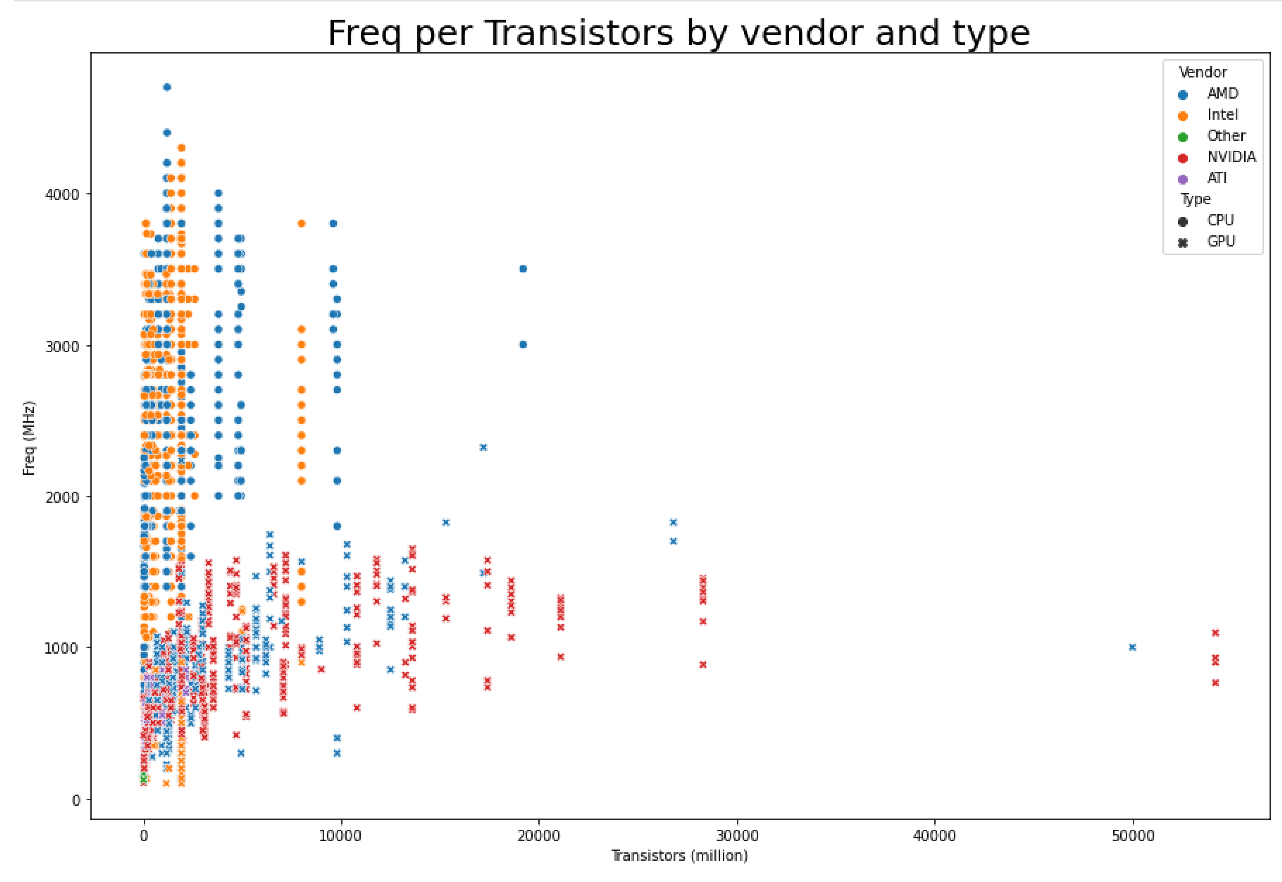
In [808…

```python
# Creating a bar chart

plt.figure(figsize = (12,8))
ax = sns.countplot(data=chip, x="Vendor", hue="Type")
ax.set_title("Count of Chips by Vendor and Type", fontsize= 25)
plt.show()
```



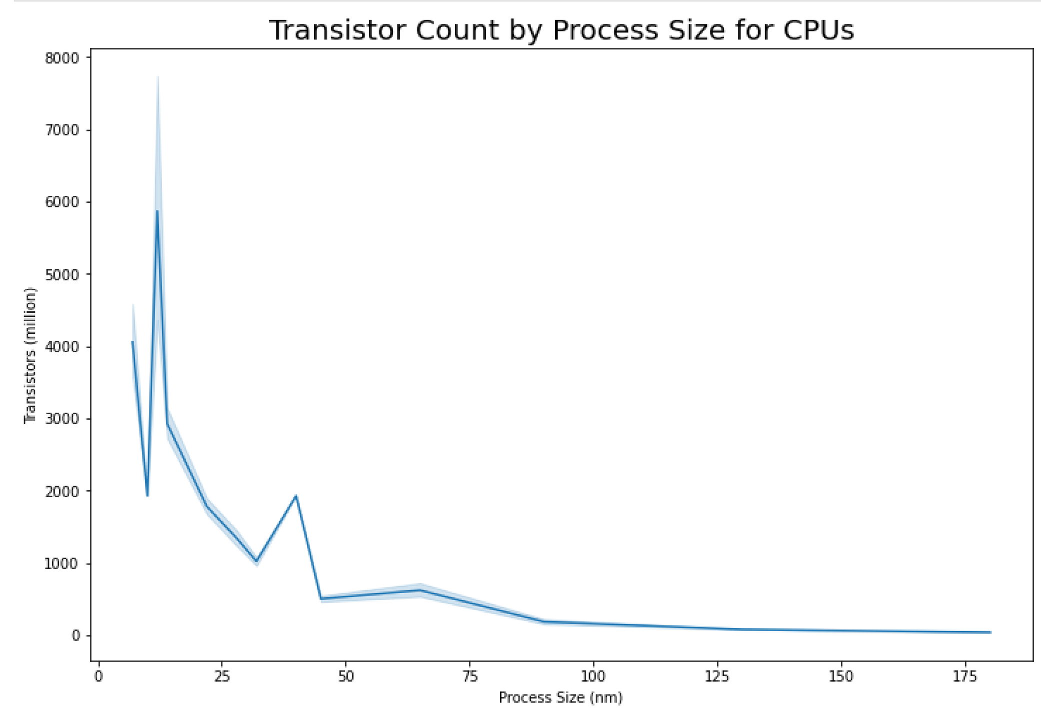
In [809…

```python
# Creating a scatterplot chart

plt.figure(figsize=(15, 10))
ax = sns.scatterplot(data=chip, x="Transistors (million)", y="Freq (MHz)", hue="Vendor", style="Type")
ax.set_title("Freq per Transistors by vendor and type", fontsize= 25)
plt.show()
```



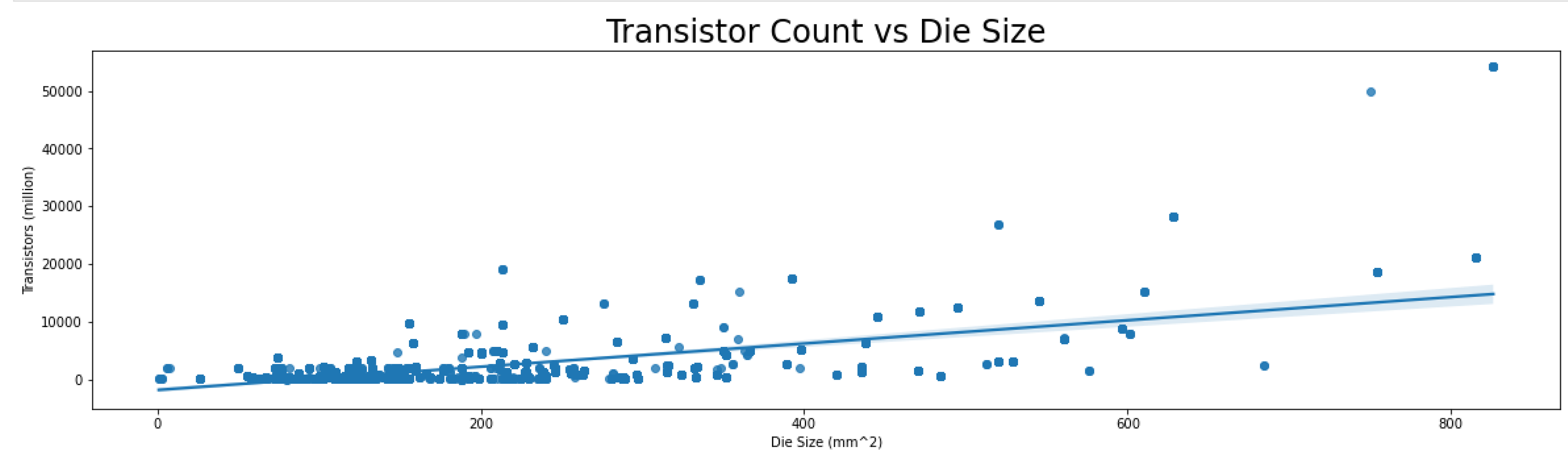
In [810…

```python
# Creating a line graph

plt.figure(figsize=(12, 8))
my_chip = chip.query("Type == 'CPU'")
sns.lineplot(data=my_chip, x="Process Size (nm)", y="Transistors (million)")
plt.title("Transistor Count by Process Size for CPUs", fontsize= 20)
plt.show()
```



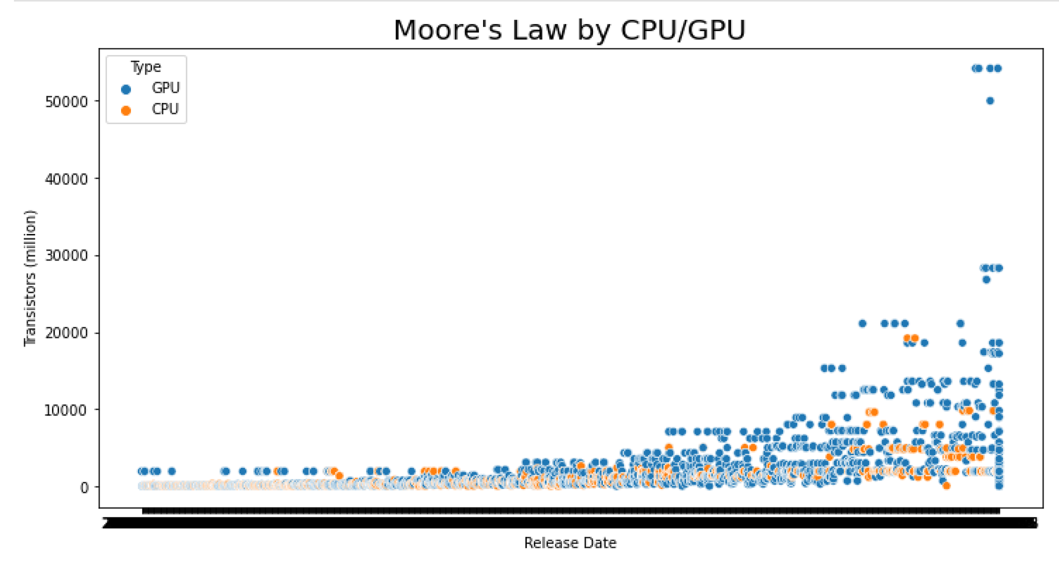
In [811…

```python
# Creating a scatterplot chart

chip['Die Size (mm^2)'] = chip['Die Size (mm^2)'].astype(float)
chip['Transistors (million)'] = chip['Transistors (million)'].astype(float)

plt.figure(figsize=(20, 5))
sns.regplot(x='Die Size (mm^2)', y='Transistors (million)', data=chip)

plt.xlabel('Die Size (mm^2)')
plt.ylabel('Transistors (million)')
plt.title('Transistor Count vs Die Size', fontsize=24)

plt.show()
```



In [812…

```python
# Let's chack Moore's Law

plt.figure(figsize=(12, 6))
sns.scatterplot(data=chip.sort_values(by='Release Date'), x='Release Date', y='Transistors (million)', hue='Type')
plt.title("Moore's Law by CPU/GPU", fontsize= 20)
plt.show()
```



In [813…

```python
# Splitting the dataset into features and target

X = chip[['Die Size (mm^2)', 'Transistors (million)']]
y = chip["Vendor"]
```

In [814…

```python
# Splitting the dataset into training and test sets in the ratio 70/30

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 10)
```

In [815…

```python
# Feature Scaling

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [816…

```python
# Training a Random Forests classifier

rf = RandomForestClassifier(n_estimators=10, criterion='gini',random_state=1)
rf.fit(X_train,y_train)
```

Out[816…

```
RandomForestClassifier(n_estimators=10, random_state=1)
```

In [817…

```python
# Performing 5-Fold Cross Validation

k = 5
kf = KFold(n_splits=k)

#for other scoring parameters see here https://scikit-learn.org/stable/modules/model_evaluation.html
result = cross_val_score(rf, X_train, y_train.ravel(), cv = kf, scoring='accuracy')

print(f' Avg accuracy:{result.mean()}')
```

```
 Avg accuracy:0.9537810794420862
```

In [818…

```python
# Making predictions using your test data (X_test)- Random Forests

y_pred_rf = rf.predict(X_test)
```

In [819…

```python
# Random Forests - Generating Classification Report

print(classification_report(y_test, y_pred_rf, target_names = ['AMD', 'NVIDIA', 'ATI', 'intel', 'Other Vnedor']))
```

```
              precision    recall  f1-score   support

         AMD       0.98      0.93      0.95       488
      NVIDIA       0.93      0.96      0.95       159
         ATI       0.91      0.98      0.94       406
       intel       0.97      0.97      0.97       388
Other Vnedor       1.00      0.88      0.93        16

    accuracy                           0.95      1457
   macro avg       0.96      0.94      0.95      1457
weighted avg       0.96      0.95      0.95      1457
```

In [12]:

```python
print ("Did you like it? Send me an Email: hagay92@gmail.com")
```

```
Did you like it? Send me an Email: hagay92@gmail.com
```