```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

saldf = pd.read_csv('/content/drive/MyDrive/archive (3) (1).zip')

new_saldf = saldf [["Year","Engine Size"]]
print(new_saldf)
```

```
      Year  Engine Size
0     2016          2.3
1     2018          4.4
2     2013          4.5
3     2011          4.1
4     2009          2.6
...    ...          ...
2495  2020          2.4
2496  2001          5.7
2497  2021          1.1
2498  2002          4.5
2499  2005          4.6

[2500 rows x 2 columns]
```

```
{"type":"dataframe","variable_name":"saldf"}
```

```
{'type': 'dataframe', 'variable_name': 'saldf'}
```

```
new_saldf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Year         2500 non-null   int64  
 1   Engine Size  2500 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 39.2 KB
```

```
inp = new_saldf[['Year']]
out = new_saldf['Engine Size']

LR = LinearRegression()

LR.fit(inp,out)
```

```
LinearRegression()
```

```
LinearRegression()
```

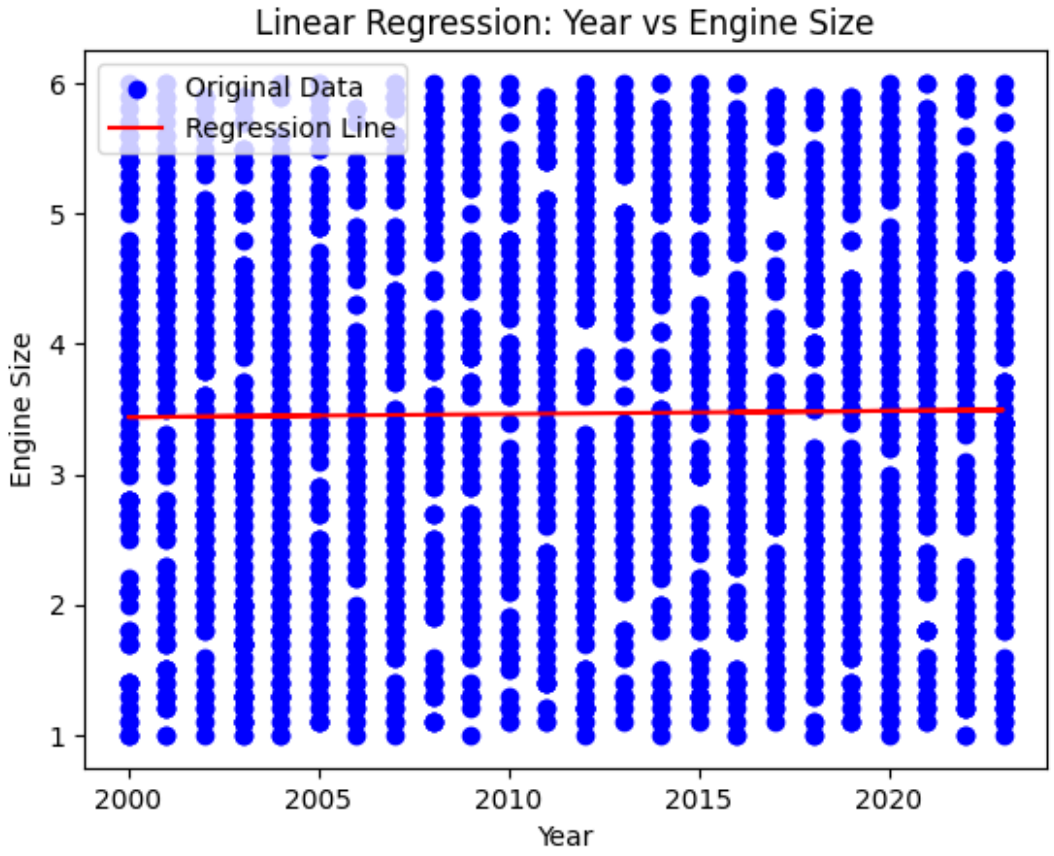
```
LinearRegression()
```

```
LR.predict([[2016]])
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/
validation.py:2739: UserWarning: X does not have valid feature names,
but LinearRegression was fitted with feature names
  warnings.warn(
```

```
array([3.47583889])
```

```
plt.scatter(inp, out, color='blue', label='Original Data')
plt.plot(inp, LR.predict(inp), color='red', label='Regression Line')
plt.xlabel('Year')
plt.ylabel('Engine Size')
plt.title('Linear Regression: Year vs Engine Size')
plt.legend()
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder

status_encoder = LabelEncoder()
```

```
status_encoder.fit(np.array([sample_status_to_encode]))

encoded_value =
status_encoder.transform(np.array([sample_status_to_encode]))[0]
print(f"Encoded value for '{sample_status_to_encode}':\
n{encoded_value}")
```

```
Encoded value for 'Developed':
0
```

```
decoded_status =
status_encoder.inverse_transform(np.array([encoded_value]))[0]
print(f"Decoded value for {encoded_value}: {decoded_status}")
```

```
Decoded value for 0: Developed
```