Batch: HO-ML 1 **Experiment Number: 06**

Roll Number: 16010422234 **Name: Chandana Galgali**

Aim of the Experiment: To demonstrate the use of PCA for dimensionality reduction.

Program/ Steps:

1. Refer to: https://www.kaggle.com/code/avikumart/pca-principal-component-analysis-from-scratch for the code and the description for the PCA implementation. Use VScode and Matplotlib to reproduce the results on your machine.

Output/Result:

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import os
import shutil
import kagglehub
adityadesai13_used_car_dataset_ford_and_mercedes_path =
kagglehub.dataset_download('adityadesai13/used-car-dataset-ford-and-merced
es')
print('Data source import complete.')
```

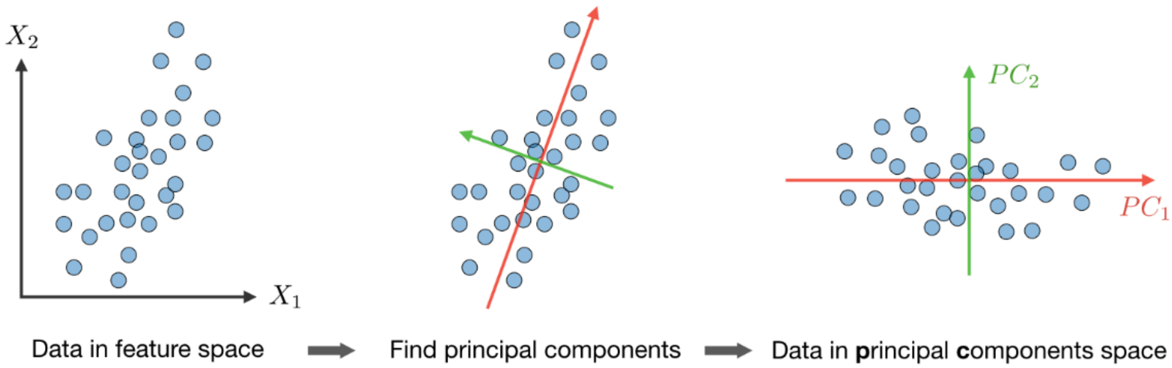
```
Downloading from https://www.kaggle.com/api/v1/datasets/download/adityadesai13/used-car-dataset-ford-and-mercedes?dataset_version_number=3...
100%|██████████| 1.10M/1.10M [00:00<00:00, 38.1MB/s]Extracting files...

Data source import complete.
```

Introduction

> Principal component analysis is a dimention reduction technique that finds the variance maximizing directions onto which to project the data.

> Eigenvalue, eigenvector-- Given a matrix A\in\mathbb{R}^{n\times n}A∈R n×n , \lambdaλ is said to be an eigenvalue of AA if there exists a vector z\in\mathbb{R}^n\backslash{0}z∈R n {0}, called eigenvector.

> Principal components are calculated to reduce variance of features and thus reducing dimentionality of features




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.feature_selection import mutual_info_regression

# matplotlib defaults

plt.style.use("seaborn-darkgrid")
plt.rc("figure", autolayout=True)
plt.rc(
    "axes",
    labelweight="bold",
    labelsize="large",
    titleweight="bold",
    titlesize=14,
    titlepad=10,
)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
<ipython-input-2-8eddb638de36>:10: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6
  plt.style.use("seaborn-darkgrid")
```

```python
ford = pd.read_csv('/content/ford.csv')
ford
```

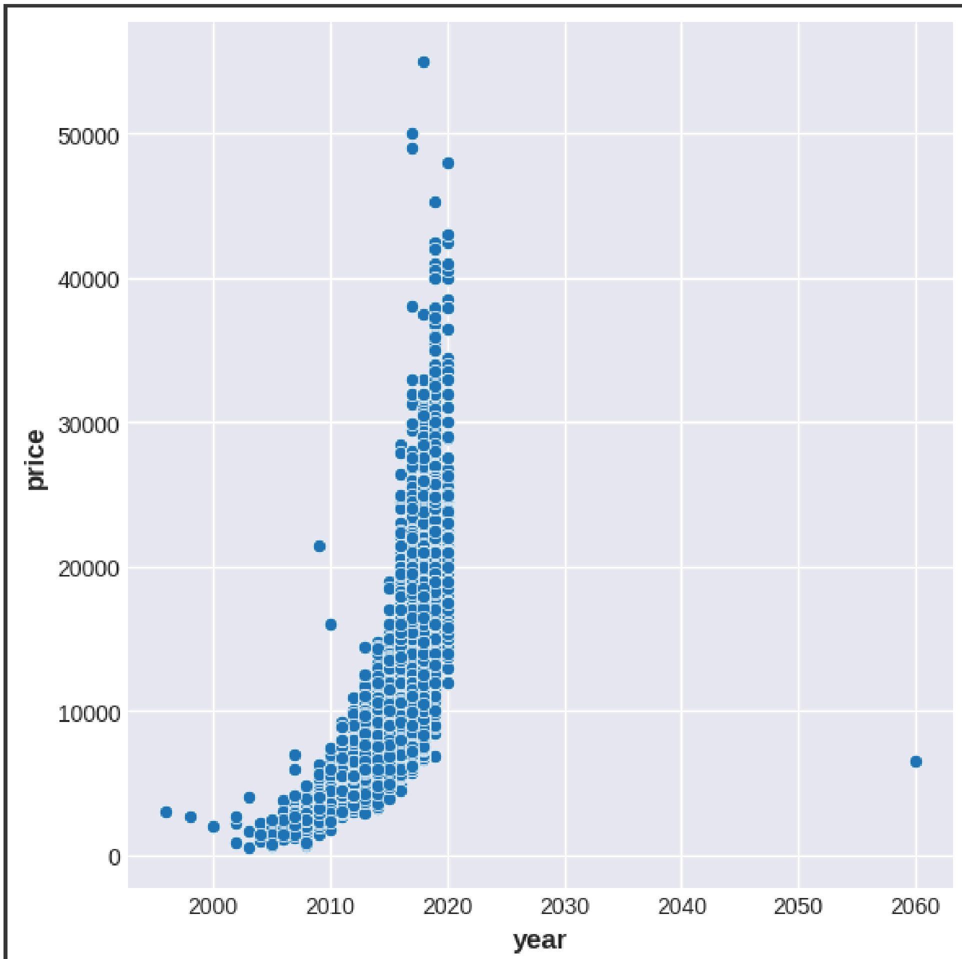
	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	Fiesta	2017	12000	Automatic	15944	Petrol	150	57.7	1.0
1	Focus	2018	14000	Manual	9083	Petrol	150	57.7	1.0
2	Focus	2017	13000	Manual	12456	Petrol	150	57.7	1.0
3	Fiesta	2019	17500	Manual	10460	Petrol	145	40.3	1.5
4	Fiesta	2019	16500	Automatic	1482	Petrol	145	48.7	1.0
...	...	...	...	...	...	...	...	...	...
17960	Fiesta	2016	7999	Manual	31348	Petrol	125	54.3	1.2
17961	B-MAX	2017	8999	Manual	16700	Petrol	150	47.1	1.4
17962	B-MAX	2014	7499	Manual	40700	Petrol	30	57.7	1.0
17963	Focus	2015	9999	Manual	7010	Diesel	20	67.3	1.6
17964	KA	2018	8299	Manual	5007	Petrol	145	57.7	1.2

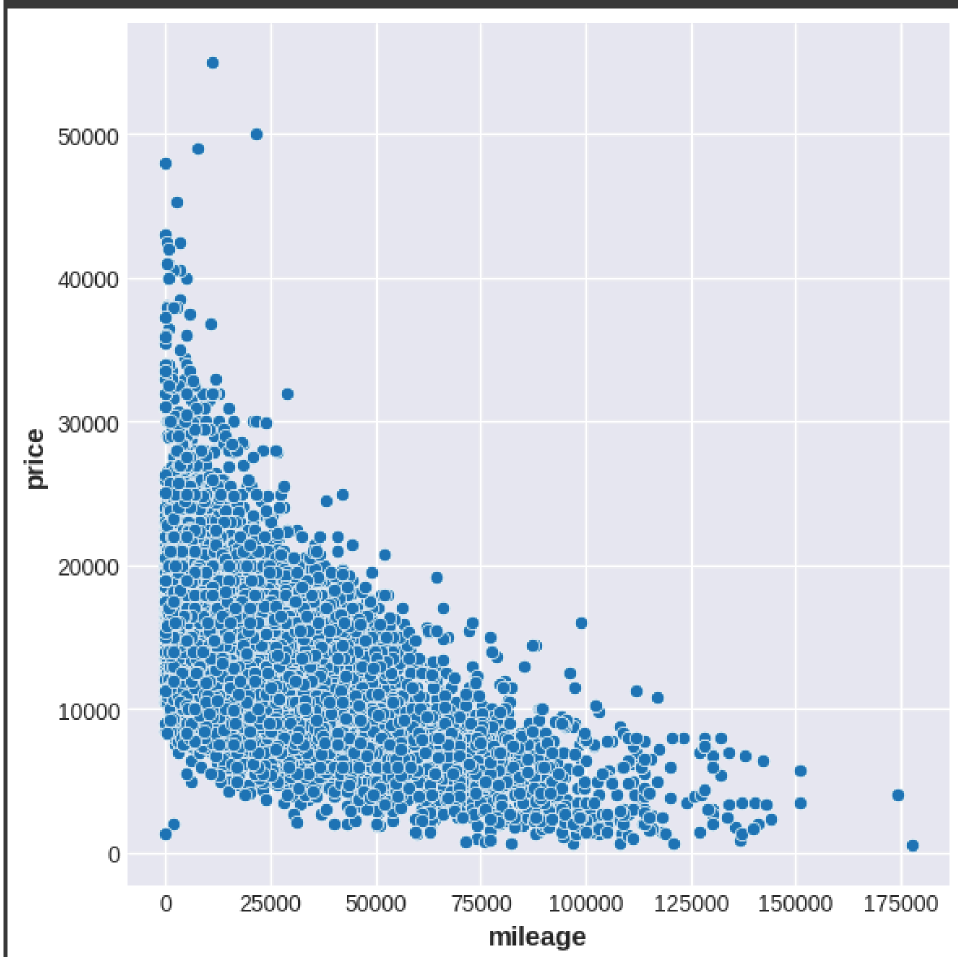
17965 rows × 9 columns

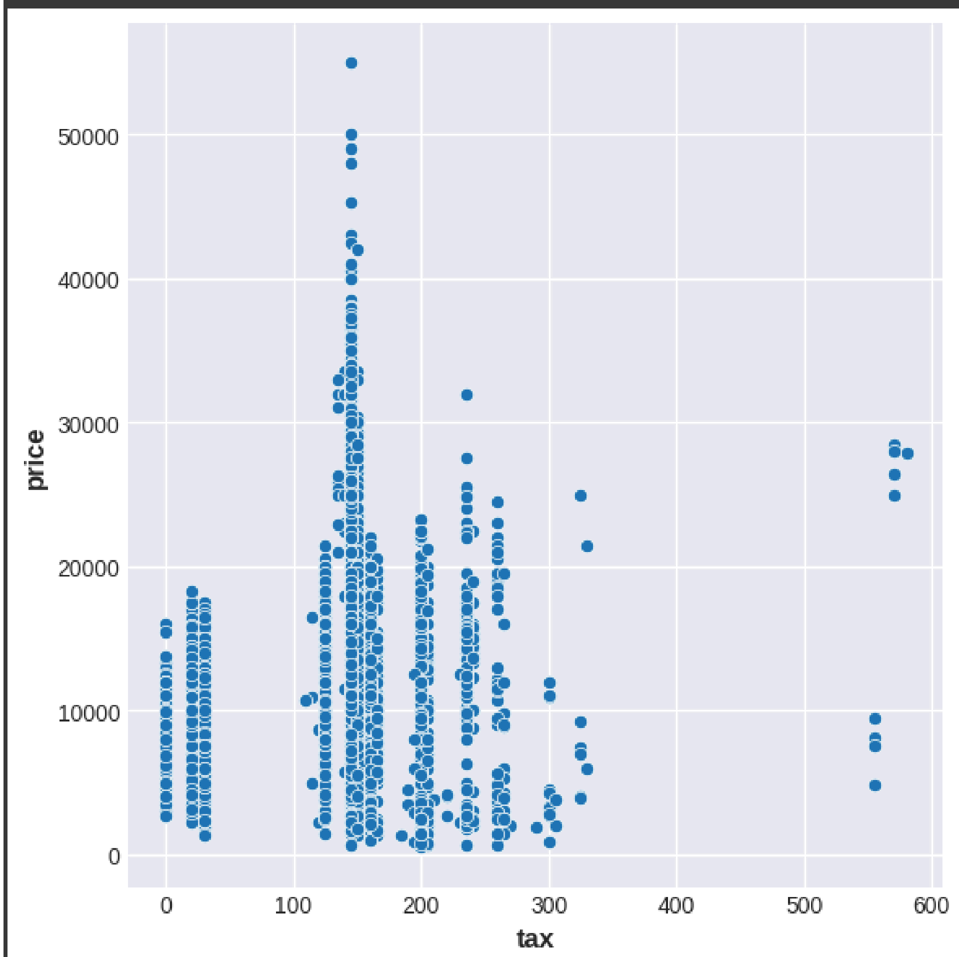
```python
# variance among numircal features
ford.var()
```

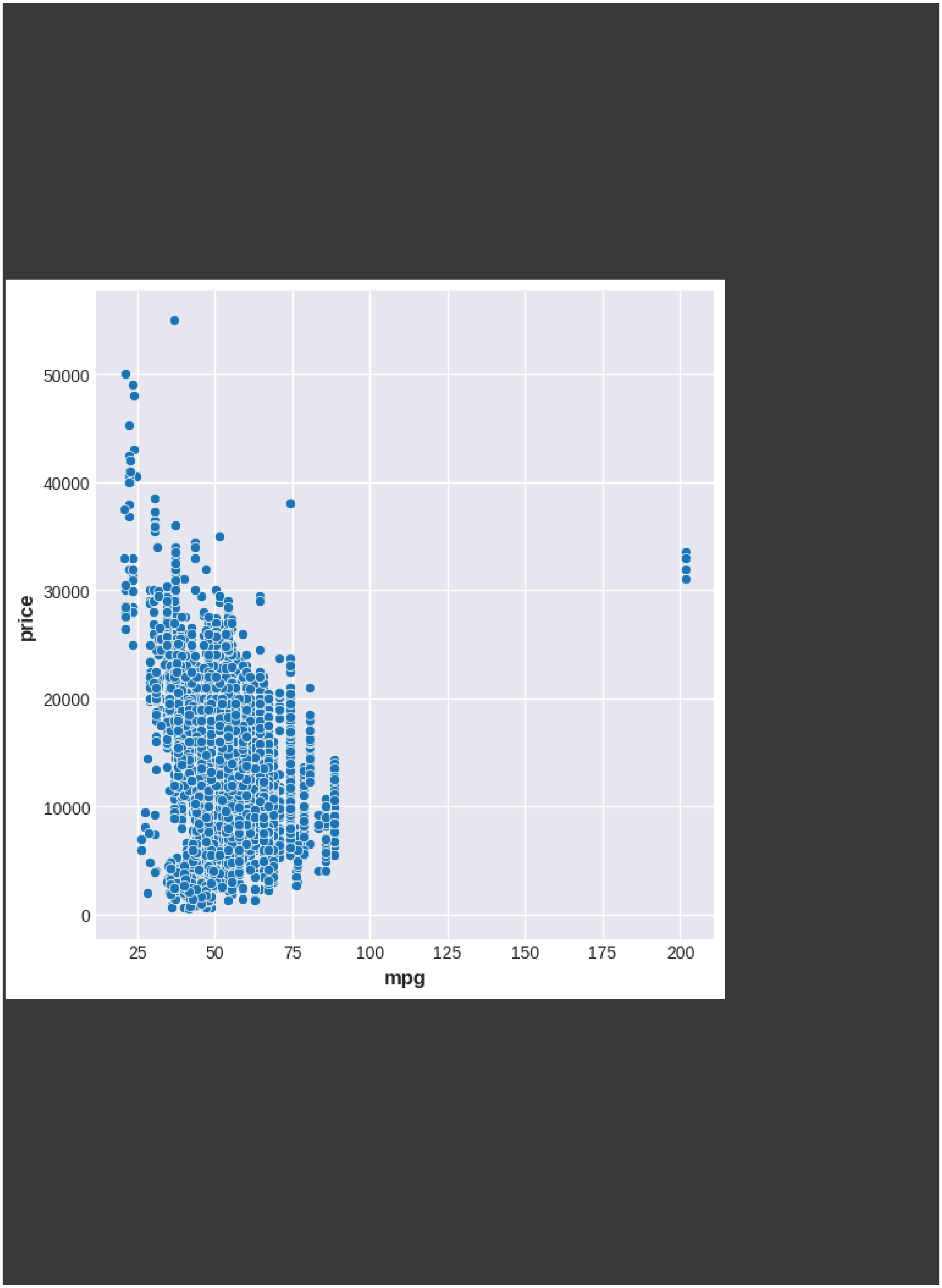
```
year          4.203918e+00
price         2.248071e+07
mileage       3.791633e+08
tax           3.845294e+03
mpg           1.025354e+02
engineSize    1.869450e-01
dtype: float64
```

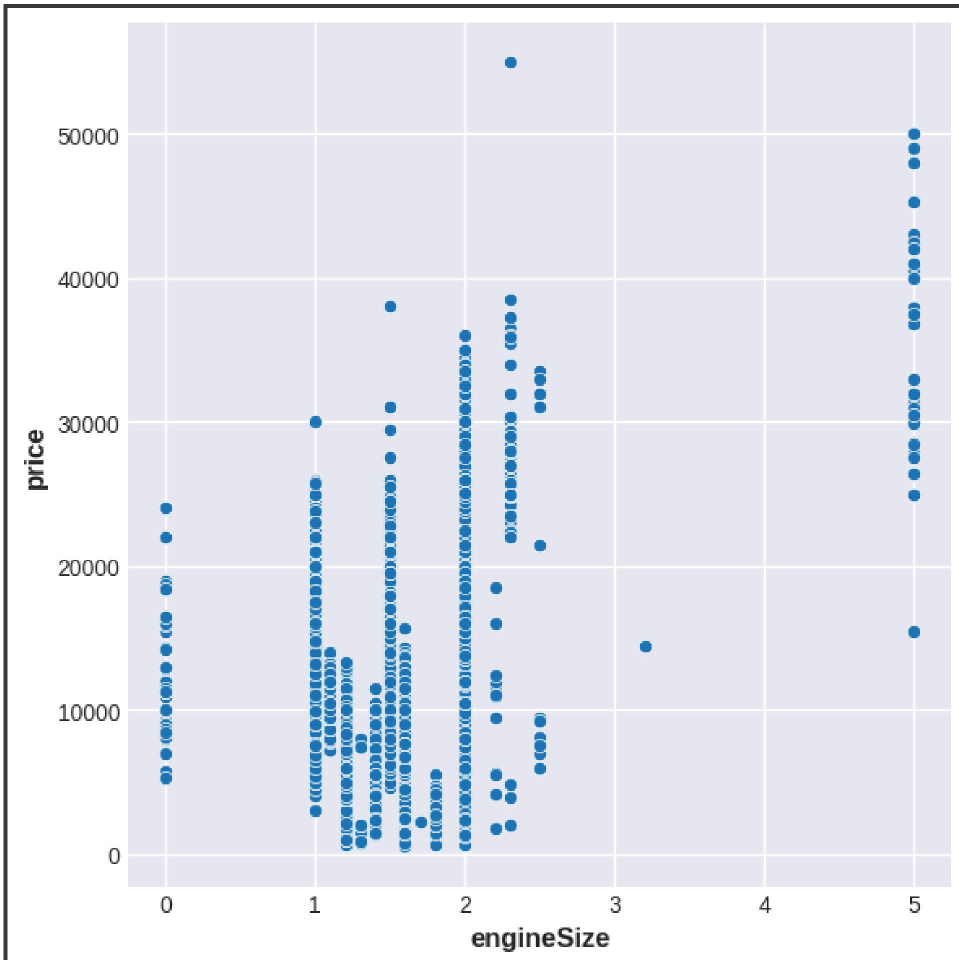
```
y = ford['price']
ford_ = ford.drop('price', axis=1)

cols = [col for col in ford_.columns if ford_[col].dtype in
['int64','float64']]

for idx, col in enumerate(cols):
    plt.figure(idx, figsize=(6,6))
    sns.scatterplot(x=col, y=y, data=ford_)
    plt.show
```




price
mileage
0
10000
20000
30000
40000
50000
0
25000
50000
75000
100000
125000
150000
175000


price
50000
40000
30000
20000
10000
0
0
100
200
300
400
500
600
tax


price
50000
40000
30000
20000
10000
0
25
50
75
100
125
150
175
200
mpg



> As per above plots there are some outliears in in dataset which can affect model performance
>
> We can mileage feature having largest varince among all features
>
> We will do principal component analysis on this features to reduce variance and and reduce effects of outliears

```
features = ['mileage','year','mpg','tax','engineSize']

X = ford_[features]

# normalizing features
X_norm = (X - X.mean(axis=0))/X.std(axis=0)

# principal component analysis on features
pca = PCA()

# fit and transform X_norm to PCA dataframe
X_pca = pca.fit_transform(X_norm)
```

```python
# converting to dataframe
names = [f"PC{i+1}" for i in range(X_pca.shape[1])]
X_pcadf = pd.DataFrame(X_pca, columns=names)


print(X_pcadf.head())
print("++++++++++++++++++++++++++++++++++++++++++++++++++++")
print("shape of pca df:", X_pcadf.shape)
```

```
        PC1       PC2       PC3       PC4       PC5
0 -0.575611 -0.368421 -0.708761  0.371575 -0.224397
1 -1.059305 -0.623569 -0.499118  0.275858 -0.125147
2 -0.679683 -0.423364 -0.687725  0.308659 -0.342023
3 -1.771358  0.858840 -0.159534 -0.833414  0.574171
4 -1.810537 -0.426270 -0.659147 -0.402589  0.187178
++++++++++++++++++++++++++++++++++++++++++++++++++++
shape of pca df: (17965, 5)
```

```python
pca.singular_values_
```

```
array([189.10834173, 167.05350962, 113.43209754,  92.58589638,
        68.64525512])
```

```python
X_norm.T
```

	0	1	2	3	4	5	6	7	8	9	...	17955	17956	17957	17958	17959	17960	17961	17962	17963	17964
mileage	-0.381039	-0.733389	-0.560167	-0.662672	-1.123742	0.619777	-1.095650	-0.529456	-0.845806	1.272454	...	-0.837949	-0.950777	1.168819	-0.513793	0.630305	0.410041	-0.342214	0.890318	-0.839849	-0.942714

year	0.065075	0.552798	0.065075	1.040520	1.040520	-0.910370	1.040520	0.065075	1.040520	0.552798	...	0.552798	0.552798	-0.910370	1.040520	-0.422648	-0.422648	0.065075	-1.398093	-0.910370	0.552798
mpg	-0.020442	-0.020442	-0.020442	-1.738794	-0.909245	-0.988249	-0.741360	-0.356212	-1.551158	0.344955	...	-0.909245	-1.067254	-0.454968	-0.909245	1.085624	-0.356212	-1.067254	-0.020442	0.927615	-0.020442
tax	0.591279	0.591279	0.591279	0.510647	0.510647	0.510647	0.510647	0.510647	0.510647	0.510647	...	0.510647	0.510647	1.888121	0.510647	-1.505142	1.888121	0.591279	-1.343879	-1.505142	0.510647
engineSize	-0.811401	-0.811401	-0.811401	0.345012	-0.811401	0.576295	-0.811401	-0.348836	1.501425	-0.811401	...	-0.811401	-0.811401	-0.811401	-0.811401	0.345012	-0.348836	0.113730	-0.811401	0.576295	-0.348836

5 rows × 17965 columns

```
# we will perform PCA from scratch using numpy library
# X_norm is the Z-scoretransformed dataframe in our dataset

# convert cov_matrix from the X_norm
cov_matrix = np.cov(X_norm.T)
print("Convariance matrix: ", cov_matrix)
```

```
Convariance matrix:  [[ 1.         -0.70789926  0.12007683 -0.26055045  0.2150014 ]
 [-0.70789926  1.         -0.02296881  0.29845652 -0.13735825]
 [ 0.12007683 -0.02296881  1.         -0.50301254 -0.26052712]
 [-0.26055045  0.29845652 -0.50301254  1.          0.18431146]
 [ 0.2150014  -0.13735825 -0.26052712  0.18431146  1.        ]]
```

```
# from COV_MATRIX calculate eigenvectors and eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print("Eigenvectors:", eigenvectors)
print("Eigenvalues:", eigenvalues)
```

```
Eigenvectors: [[ 0.5809945  -0.30672577  0.65666384  0.35123476 -0.11743699]
 [-0.57200807  0.30155231  0.6778961   0.05749211  0.34499929]
 [ 0.3198565   0.55811472 -0.25087814  0.57436467  0.43973307]
 [-0.48160177 -0.39623711 -0.18663366  0.73028686 -0.20713489]
 [ 0.03168146 -0.58863369 -0.10713317 -0.10057401  0.79430096]]
Eigenvalues: [1.99075734 1.55348893 0.2623119  0.47718483 0.716257  ]
```

```
# sort the eigen values and eigen vectors in descending order
eig_pairs = [(eigenvalues[index],eigenvectors[:,index]) for index in
range(len(eigenvalues))]

# sort the pairs
eig_pairs.sort()

# reverse to make it in correct order
eig_pairs.reverse()
print(eig_pairs)

# extract the sorted eiganvalues and eiganvectors
eigenvalues_sorted = [eig_pairs[index][0] for index in
range(len(eigenvalues))]
eigenvectors_sorted = [eig_pairs[index][1] for index in
range(len(eigenvalues))]

# print sorted eigan values
print("Sorted eigan values:", eigenvalues_sorted)
```

```
[(1.9907573430794159, array([ 0.5809945 , -0.57200807,  0.3198565 ,
-0.48160177,  0.03168146])), (1.5534889266247038, array([-0.30672577,
0.30155231,  0.55811472, -0.39623711, -0.58863369])), (0.7162570002651933,
array([-0.11743699,  0.34499929,  0.43973307, -0.20713489,  0.79430096])),
(0.47718482565372217, array([ 0.35123476,  0.05749211,  0.57436467,
0.73028686, -0.10057401])), (0.26231190437677343, array([ 0.65666384,
0.6778961 , -0.25087814, -0.18663366, -0.10713317]))]
```

```
Sorted eigan values: [1.9907573430794159, 1.5534889266247038,
0.7162570002651933, 0.47718482565372217, 0.26231190437677343]
print(eigenvectors_sorted)
[array([ 0.5809945 , -0.57200807,  0.3198565 , -0.48160177,  0.03168146]),
array([-0.30672577,  0.30155231,  0.55811472, -0.39623711, -0.58863369]),
array([-0.11743699,  0.34499929,  0.43973307, -0.20713489,  0.79430096]),
array([ 0.35123476,  0.05749211,  0.57436467,  0.73028686, -0.10057401]),
array([ 0.65666384,  0.6778961 , -0.25087814, -0.18663366, -0.10713317])]
# plot the variance plots using sorted eigenvalues and eigenvectors
total = sum(eigenvalues_sorted)
var_explained = [(i/total) for i in eigenvalues_sorted]

# calculate cumulative variance
cum_var_exp = np.cumsum(var_explained)
print(var_explained)
print(cum_var_exp)
```

```
[0.3981514686158945, 0.3106977853249527, 0.14325140005304415, 0.09543696513074809, 0.0524623808753567]
[0.39815147 0.70884925 0.85210065 0.94753762 1.        ]
```

```
# transforming original dataframe into PCA
vect = np.array(eigenvectors_sorted)

# dot product to create principal components analysis
X_vect_pca = np.dot(X_norm,vect.T)

# pca dataframe
pd.DataFrame(X_vect_pca)
```

	0	1	2	3	4
0	-0.575611	0.368421	-0.708761	0.371575	-0.224397
1	-1.059305	0.623569	-0.499118	0.275858	-0.125147
2	-0.679683	0.423364	-0.687725	0.308659	-0.342023
3	-1.771358	-0.858840	-0.159534	-0.833414	0.574171
4	-1.810537	0.426270	-0.659147	-0.402589	0.187178
...	...	...	...	...	...
17960	0.264402	-0.321232	-0.666652	0.087592	0.074376
17961	-0.858574	-0.772293	-0.438807	-0.309084	-0.035392
17962	1.931960	0.304023	-0.962019	-0.679221	-0.020253
17963	1.072632	0.757961	0.961974	-0.971680	-1.182183
17964	-1.127435	0.447442	-0.090418	0.096930	-0.297110

17965 rows × 5 columns

```
X_norm
```

	mileage	year	mpg	tax	engineSize
0	-0.381039	0.065075	-0.020442	0.591279	-0.811401
1	-0.733389	0.552798	-0.020442	0.591279	-0.811401
2	-0.560167	0.065075	-0.020442	0.591279	-0.811401
3	-0.662672	1.040520	-1.738794	0.510647	0.345012
4	-1.123742	1.040520	-0.909245	0.510647	-0.811401
...	...	...	...	...	...
17960	0.410041	-0.422648	-0.356212	0.188121	-0.348836
17961	-0.342214	0.065075	-1.067254	0.591279	0.113730
17962	0.890318	-1.398093	-0.020442	-1.343879	-0.811401
17963	-0.839849	-0.910370	0.927615	-1.505142	0.576295
17964	-0.942714	0.552798	-0.020442	0.510647	-0.348836

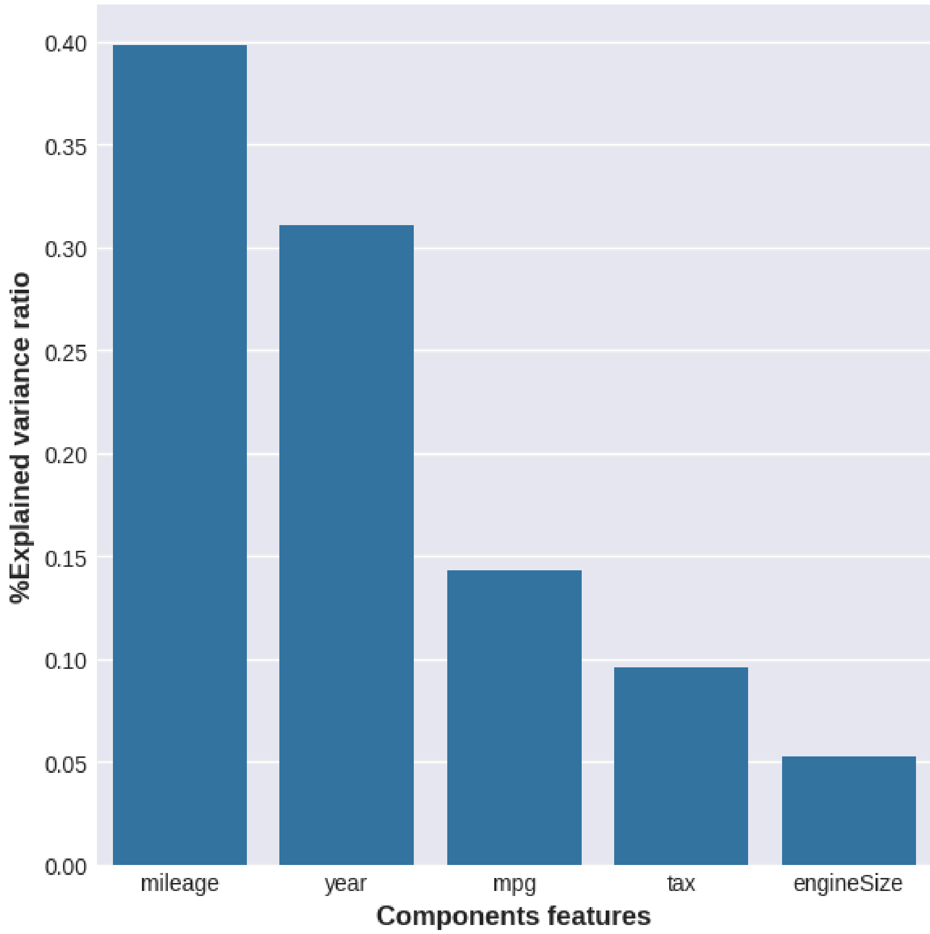
17965 rows × 5 columns

```
print(vect.T)
```

```
[[ 0.5809945  -0.30672577 -0.11743699  0.35123476  0.65666384]
 [-0.57200807  0.30155231  0.34499929  0.05749211  0.6778961 ]
 [ 0.3198565   0.55811472  0.43973307  0.57436467 -0.25087814]
 [-0.48160177 -0.39623711 -0.20713489  0.73028686 -0.18663366]
 [ 0.03168146 -0.58863369  0.79430096 -0.10057401 -0.10713317]]
```

```
evr = pca.explained_variance_ratio_
print(evr)
features = ['mileage','year','mpg','tax','engineSize']

# plot the EVR using matplotlib pyplot
plt.figure(figsize=(6,6))
sns.barplot(x=np.array(features), y=evr)
plt.xlabel("Components features")
plt.ylabel("%Explained variance ratio")
plt.show
```

```
[0.39815147 0.31069779 0.1432514  0.09543697 0.05246238]

matplotlib.pyplot.show
def show(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
----------
block : bool, optional
```

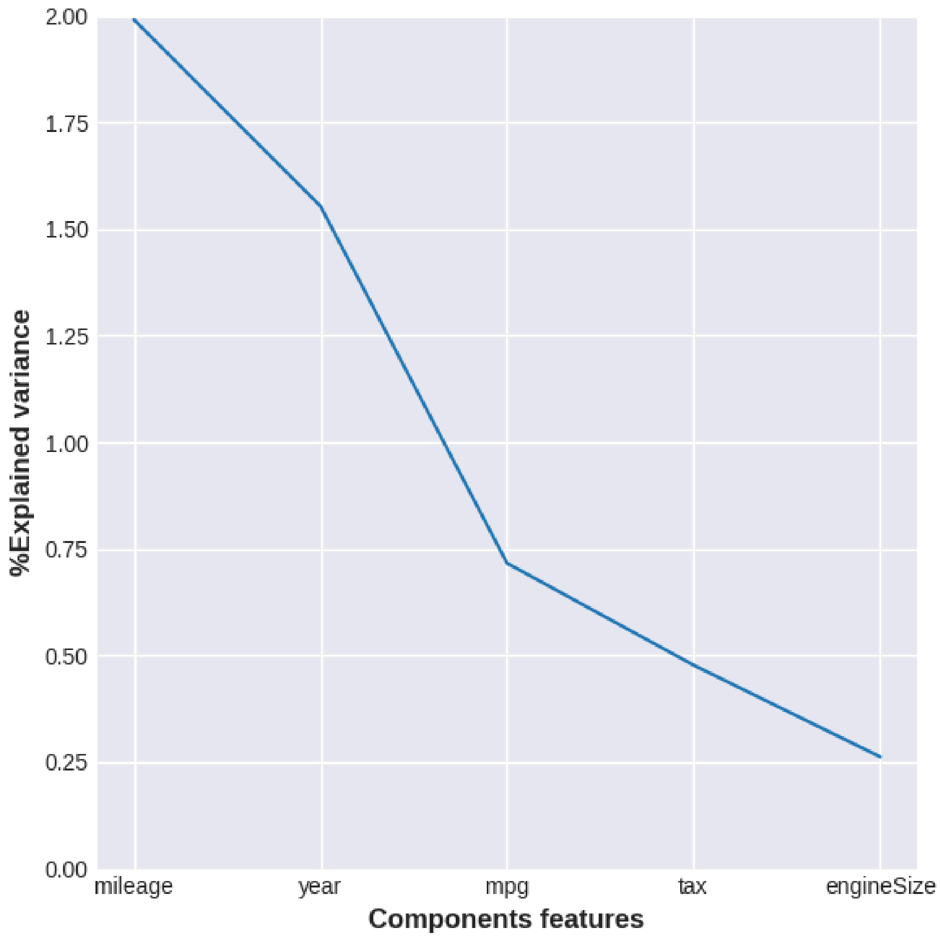


Reduced variances of each features after dimentionality reduction

Plotting explained variance

```
ev = pca.explained_variance_
print(ev)

features = ['mileage','year','mpg','tax','engineSize']


plt.figure(figsize=(6,6))
sns.lineplot(x=np.array(features), y=ev)
plt.xlabel("Components features")
plt.ylabel("%Explained variance")
plt.ylim(0,2)
plt.show
```

```
[1.99075734 1.55348893 0.716257   0.47718483 0.2623119 ]
matplotlib.pyplot.show
def show(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
----------
block : bool, optional
```

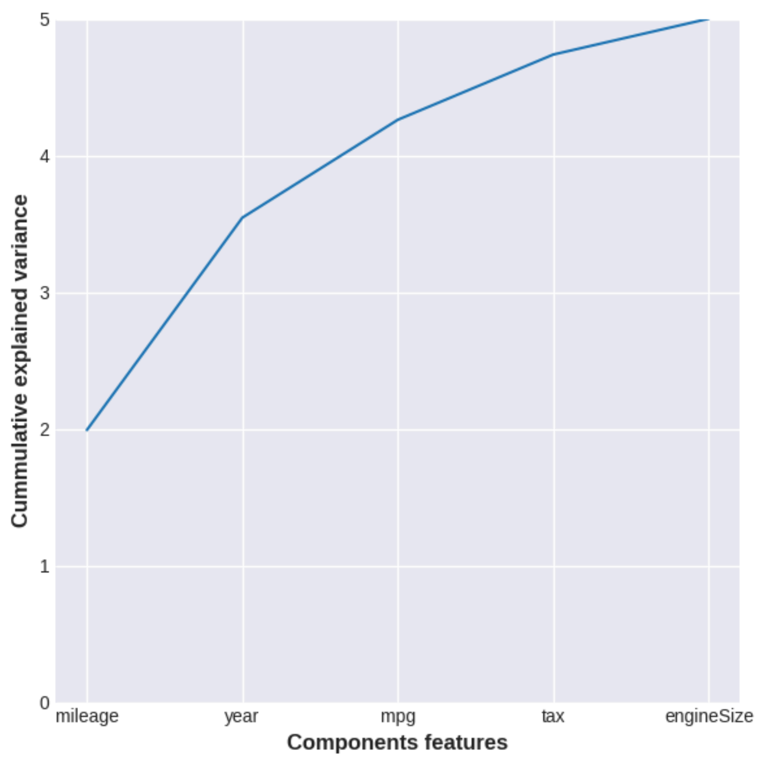


Plotting Cummulative explained variance

```
evc = np.cumsum(pca.explained_variance_)
print(evc)

features = ['mileage','year','mpg','tax','engineSize']


plt.figure(figsize=(6,6))
sns.lineplot(x=np.array(features), y=evc)
```

```
plt.xlabel("Components features")
plt.ylabel("Cummulative explained variance")
plt.ylim(0,5)
plt.show
```

```
[1.99075734 3.54424627 4.26050327 4.7376881  5.        ]
matplotlib.pyplot.show
def show(*args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
----------
block : bool, optional
```



Cummulative variance is at about 5

```
loadings = pd.DataFrame(pca.components_.T ,
                        index=np.array(features),
                        columns=names)

loadings
```

	PC1	PC2	PC3	PC4	PC5
mileage	0.580994	0.306726	-0.117437	0.351235	0.656664
year	-0.572008	-0.301552	0.344999	0.057492	0.677896
mpg	0.319856	-0.558115	0.439733	0.574365	-0.250878
tax	-0.481602	0.396237	-0.207135	0.730287	-0.186634
engineSize	0.031681	0.588634	0.794301	-0.100574	-0.107133

```
pca.noise_variance_
```

```
0.0
```

```
# covariance matrix of principal components
pca.get_covariance()
```

```
array([[ 1.        , -0.70789926,  0.12007683, -0.26055045,  0.2150014 ],
       [-0.70789926,  1.        , -0.02296881,  0.29845652, -0.13735825],
       [ 0.12007683, -0.02296881,  1.        , -0.50301254, -0.26052712],
       [-0.26055045,  0.29845652, -0.50301254,  1.        ,  0.18431146],
       [ 0.2150014 , -0.13735825, -0.26052712,  0.18431146,  1.        ]])
```

```
y = ford['price']

mi_score = mutual_info_regression(X_pcadf,y, discrete_features=False)
mi_score = pd.Series(mi_score, index=X_pcadf.columns, name="MI_SCORE")

print(mi_score)
```

```
PC1    0.537128
PC2    0.342437
PC3    0.336322
PC4    0.239213
PC5    0.109343
Name: MI_SCORE, dtype: float64
```

As we can see after PCA, PC1 which has a higest explained varience among all features has the most feature imporatance to predict target variable

Post Lab Question-Answers:

1. Explain the term Eigenvalues in your own words.

Eigenvalues are numbers that give us information about how much variance exists in the data along the direction of its corresponding eigenvector. When performing Principal Component Analysis (PCA), eigenvalues represent the magnitude of the variance captured by each principal component. The larger the eigenvalue, the more important that component is in explaining the variability of the data.

Outcomes: Comprehend radial-basis-function (RBF) networks and Kernel learning method

---

Conclusion (based on the Results and outcomes achieved):
In this experiment, the Principal Component Analysis (PCA) method was successfully applied to reduce the dimensionality of a dataset. By identifying and focusing on the principal components with the highest eigenvalues, we managed to retain the most significant information while removing less important features, which reduced the dataset's complexity. The reduction in dimensions made the data more manageable without losing crucial patterns, thus enhancing computational efficiency. The results achieved the objective of dimensionality reduction with minimal information loss.

References:

Books/ Journals/ Websites:

1. Han, Kamber, "Data Mining Concepts and Techniques", Morgan Kaufmann 3nd Edition

---