MAHARAJA SURAJMAL INSTITUTE

DEPARTMENT OF COMPUTER APPLICATION

BCA I SHIFT

Subject Code: BCA 312

Course Name:

Data Visualization & Analytics

Submitted by:

Name: Rahul Bera

Enrolment no: 09914902022

BCA 6A (Morning Shift)

Submitted to:

Dr. Neetu Narwal

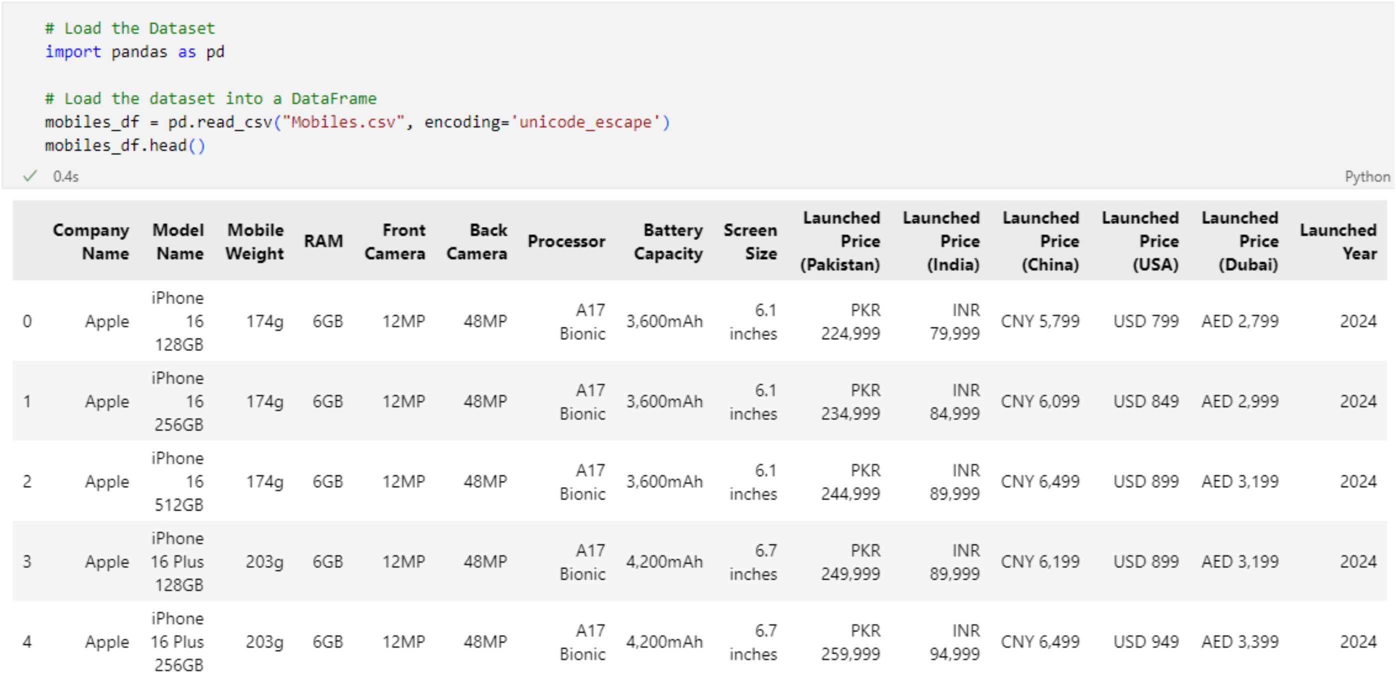
Signature:

Table of Content

Practical Assignment 1

On the Mobile 2025 dataset: https://www.kaggle.com/datasets/abdulmalik1518/mobiles-dataset-2025 : Perform the following tasks

1) Open the dataset

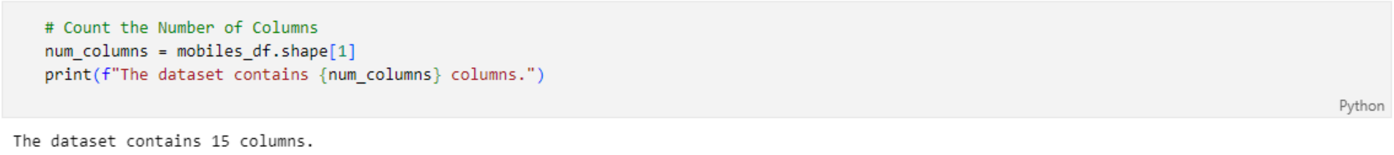


```python
# Load the Dataset
import pandas as pd

# Load the dataset into a DataFrame
mobiles_df = pd.read_csv("Mobiles.csv", encoding='unicode_escape')
mobiles_df.head()
```

✓ 0.4s Python

	Company Name	Model Name	Mobile Weight	RAM	Front Camera	Back Camera	Processor	Battery Capacity	Screen Size	Launched Price (Pakistan)	Launched Price (India)	Launched Price (China)	Launched Price (USA)	Launched Price (Dubai)	Launched Year
0	Apple	iPhone 16 128GB	174g	6GB	12MP	48MP	A17 Bionic	3,600mAh	6.1 inches	PKR 224,999	INR 79,999	CNY 5,799	USD 799	AED 2,799	2024
1	Apple	iPhone 16 256GB	174g	6GB	12MP	48MP	A17 Bionic	3,600mAh	6.1 inches	PKR 234,999	INR 84,999	CNY 6,099	USD 849	AED 2,999	2024
2	Apple	iPhone 16 512GB	174g	6GB	12MP	48MP	A17 Bionic	3,600mAh	6.1 inches	PKR 244,999	INR 89,999	CNY 6,499	USD 899	AED 3,199	2024
3	Apple	iPhone 16 Plus 128GB	203g	6GB	12MP	48MP	A17 Bionic	4,200mAh	6.7 inches	PKR 249,999	INR 89,999	CNY 6,199	USD 899	AED 3,199	2024
4	Apple	iPhone 16 Plus 256GB	203g	6GB	12MP	48MP	A17 Bionic	4,200mAh	6.7 inches	PKR 259,999	INR 94,999	CNY 6,499	USD 949	AED 3,399	2024

2) How many columns are there.

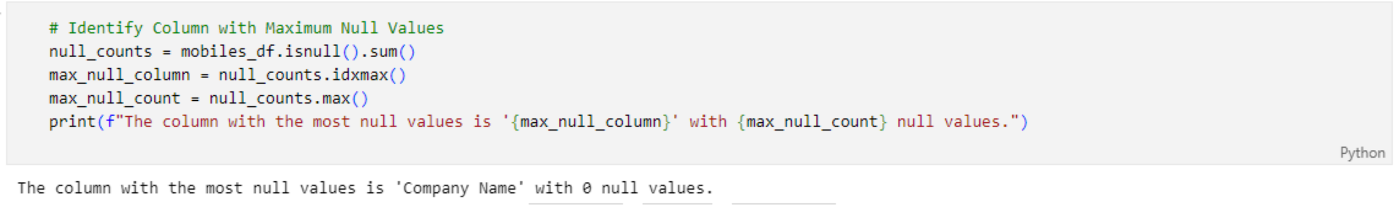


```python
# Count the Number of Columns
num_columns = mobiles_df.shape[1]
print(f"The dataset contains {num_columns} columns.")
```

Python

```
The dataset contains 15 columns.
```

3) Which column has maximum null values

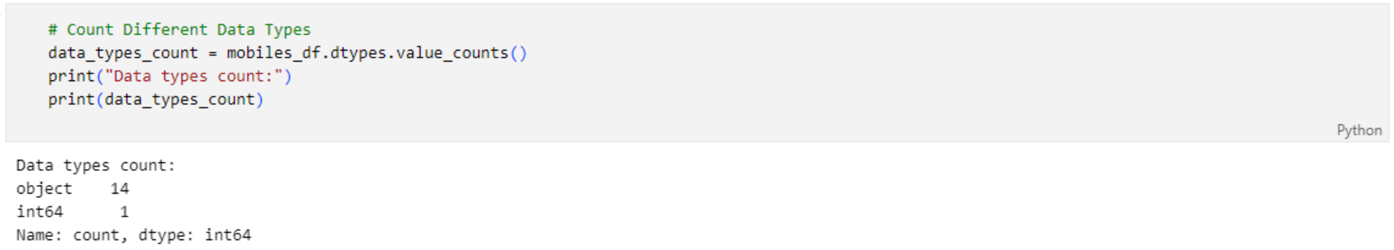


```python
# Identify Column with Maximum Null Values
null_counts = mobiles_df.isnull().sum()
max_null_column = null_counts.idxmax()
max_null_count = null_counts.max()
print(f"The column with the most null values is '{max_null_column}' with {max_null_count} null values.")
```

Python

```
The column with the most null values is 'Company Name' with 0 null values.
```

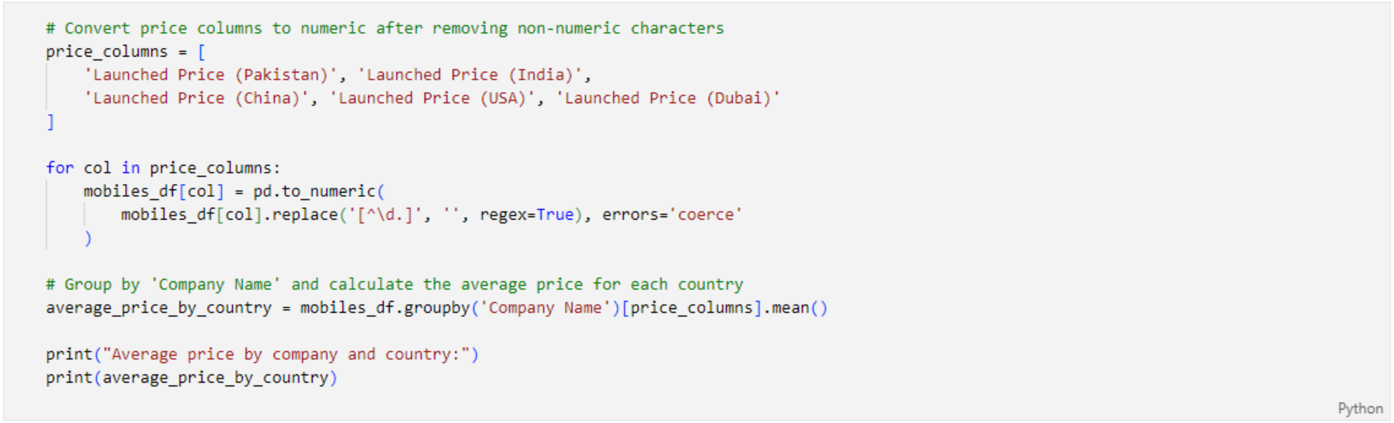
4) How many different datatypes are there.



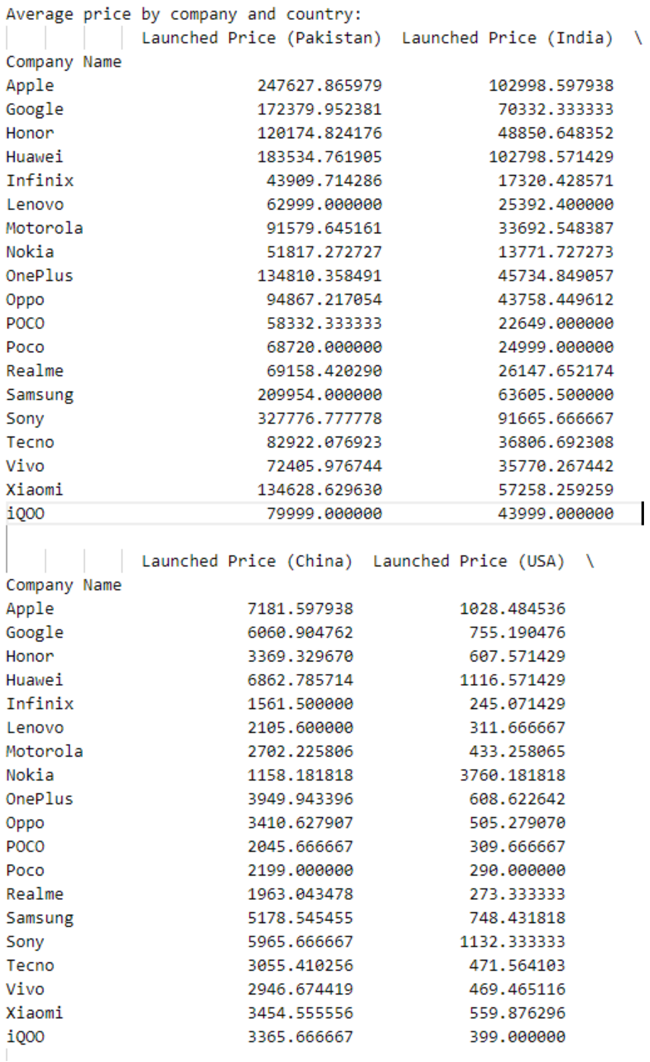
```
# Count Different Data Types
data_types_count = mobiles_df.dtypes.value_counts()
print("Data types count:")
print(data_types_count)
```

```
Data types count:
object    14
int64      1
Name: count, dtype: int64
```

5) What is average launched price in different countries

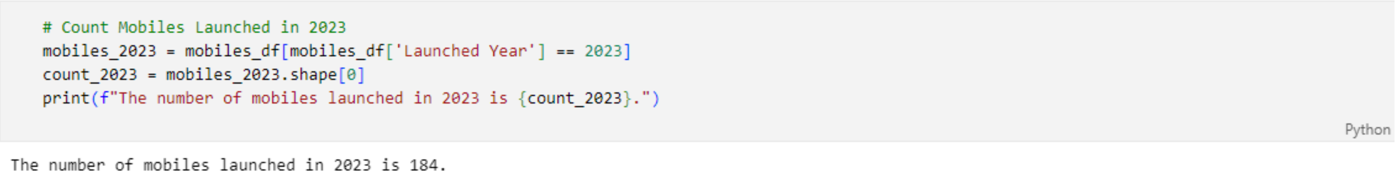


```
# Convert price columns to numeric after removing non-numeric characters
price_columns = [
    'Launched Price (Pakistan)', 'Launched Price (India)',
    'Launched Price (China)', 'Launched Price (USA)', 'Launched Price (Dubai)'
]

for col in price_columns:
    mobiles_df[col] = pd.to_numeric(
        mobiles_df[col].replace('[^\d.]', '', regex=True), errors='coerce'
    )

# Group by 'Company Name' and calculate the average price for each country
average_price_by_country = mobiles_df.groupby('Company Name')[price_columns].mean()

print("Average price by company and country:")
print(average_price_by_country)
```



```
Average price by company and country:
              Launched Price (Pakistan)  Launched Price (India)  \
Company Name
Apple                     247627.865979           102998.597938
Google                    172379.952381            70332.333333
Honor                     120174.824176            48850.648352
Huawei                    183534.761905           102798.571429
Infinix                    43909.714286            17320.428571
Lenovo                     62999.000000            25392.400000
Motorola                   91579.645161            33692.548387
Nokia                      51817.272727            13771.727273
OnePlus                   134810.358491            45734.849057
Oppo                       94867.217054            43758.449612
POCO                       58332.333333            22649.000000
Poco                       68720.000000            24999.000000
Realme                     69158.420290            26147.652174
Samsung                   209954.000000            63605.500000
Sony                      327776.777778            91665.666667
Tecno                      82922.076923            36806.692308
Vivo                       72405.976744            35770.267442
Xiaomi                    134628.629630            57258.259259
iQOO                       79999.000000            43999.000000

              Launched Price (China)  Launched Price (USA)  \
Company Name
Apple                    7181.597938           1028.484536
Google                   6060.904762            755.190476
Honor                    3369.329670            607.571429
Huawei                   6862.785714           1116.571429
Infinix                  1561.500000            245.071429
Lenovo                   2105.600000            311.666667
Motorola                 2702.225806            433.258065
Nokia                    1158.181818           3760.181818
OnePlus                  3949.943396            608.622642
Oppo                     3410.627907            505.279070
POCO                     2045.666667            309.666667
Poco                     2199.000000            290.000000
Realme                   1963.043478            273.333333
Samsung                  5178.545455            748.431818
Sony                     5965.666667           1132.333333
Tecno                    3055.410256            471.564103
Vivo                     2946.674419            469.465116
Xiaomi                   3454.555556            559.876296
iQOO                     3365.666667            399.000000
```

6) How many mobiles were launched in 2023.

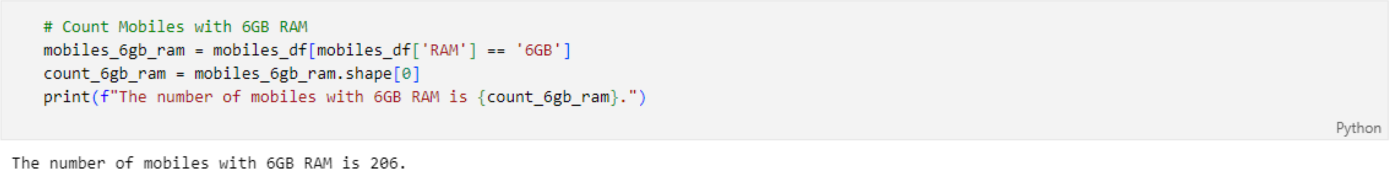


```python
# Count Mobiles Launched in 2023
mobiles_2023 = mobiles_df[mobiles_df['Launched Year'] == 2023]
count_2023 = mobiles_2023.shape[0]
print(f"The number of mobiles launched in 2023 is {count_2023}.")
```

Python

```
The number of mobiles launched in 2023 is 184.
```

7) How many mobile have 6GB RAM

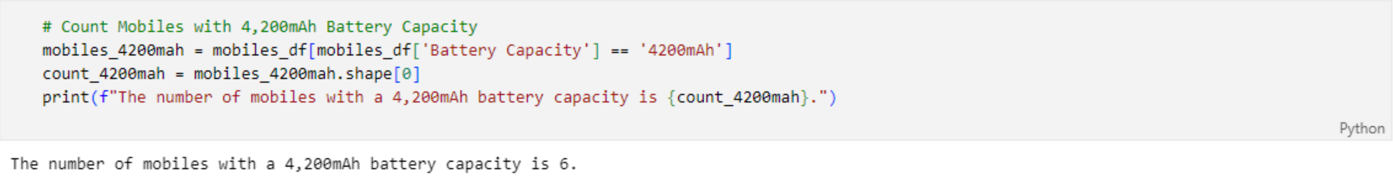


```python
# Count Mobiles with 6GB RAM
mobiles_6gb_ram = mobiles_df[mobiles_df['RAM'] == '6GB']
count_6gb_ram = mobiles_6gb_ram.shape[0]
print(f"The number of mobiles with 6GB RAM is {count_6gb_ram}.")
```

Python

```
The number of mobiles with 6GB RAM is 206.
```

8) How many mobiles have Battery Capacity of 4,200mAh

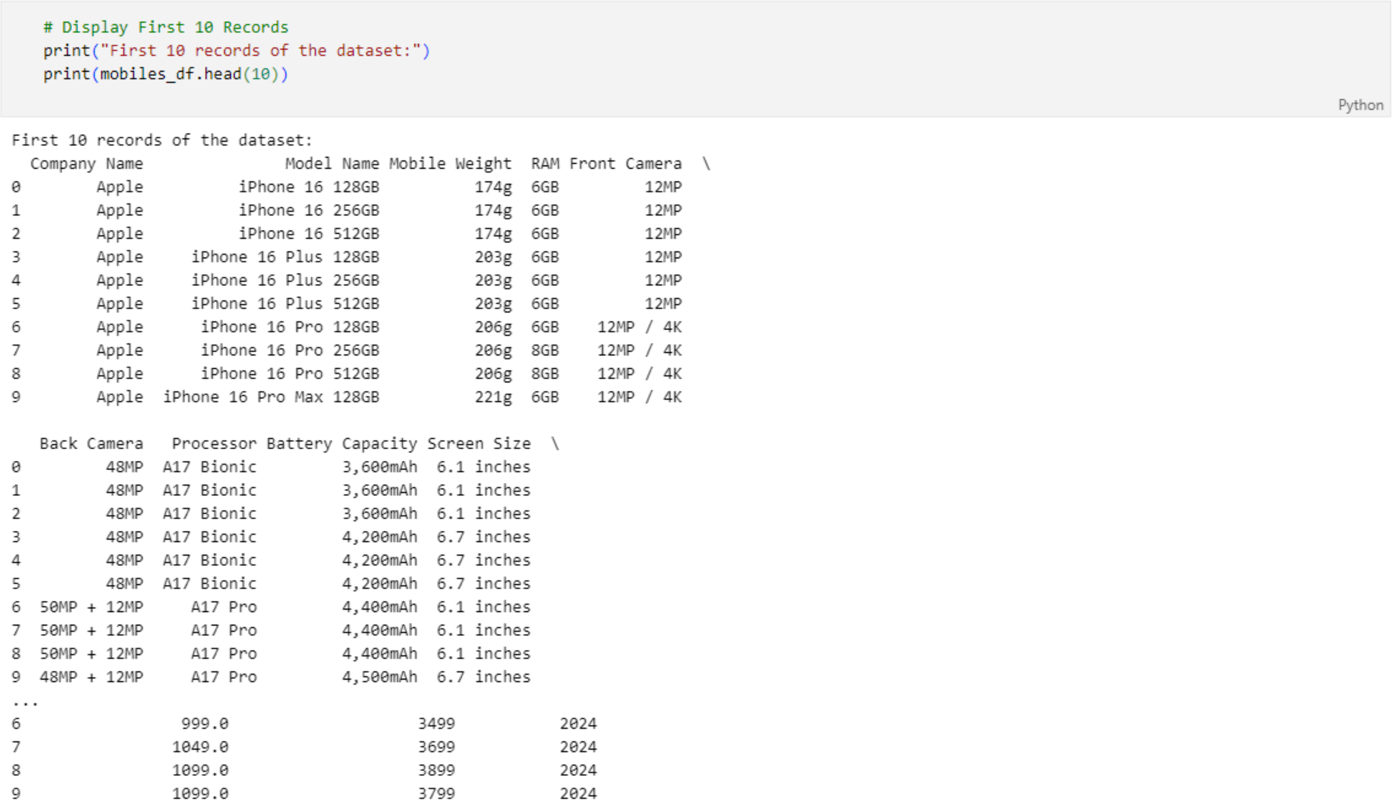


```python
# Count Mobiles with 4,200mAh Battery Capacity
mobiles_4200mah = mobiles_df[mobiles_df['Battery Capacity'] == '4200mAh']
count_4200mah = mobiles_4200mah.shape[0]
print(f"The number of mobiles with a 4,200mAh battery capacity is {count_4200mah}.")
```

Python

```
The number of mobiles with a 4,200mAh battery capacity is 6.
```

9) Display first 10 records

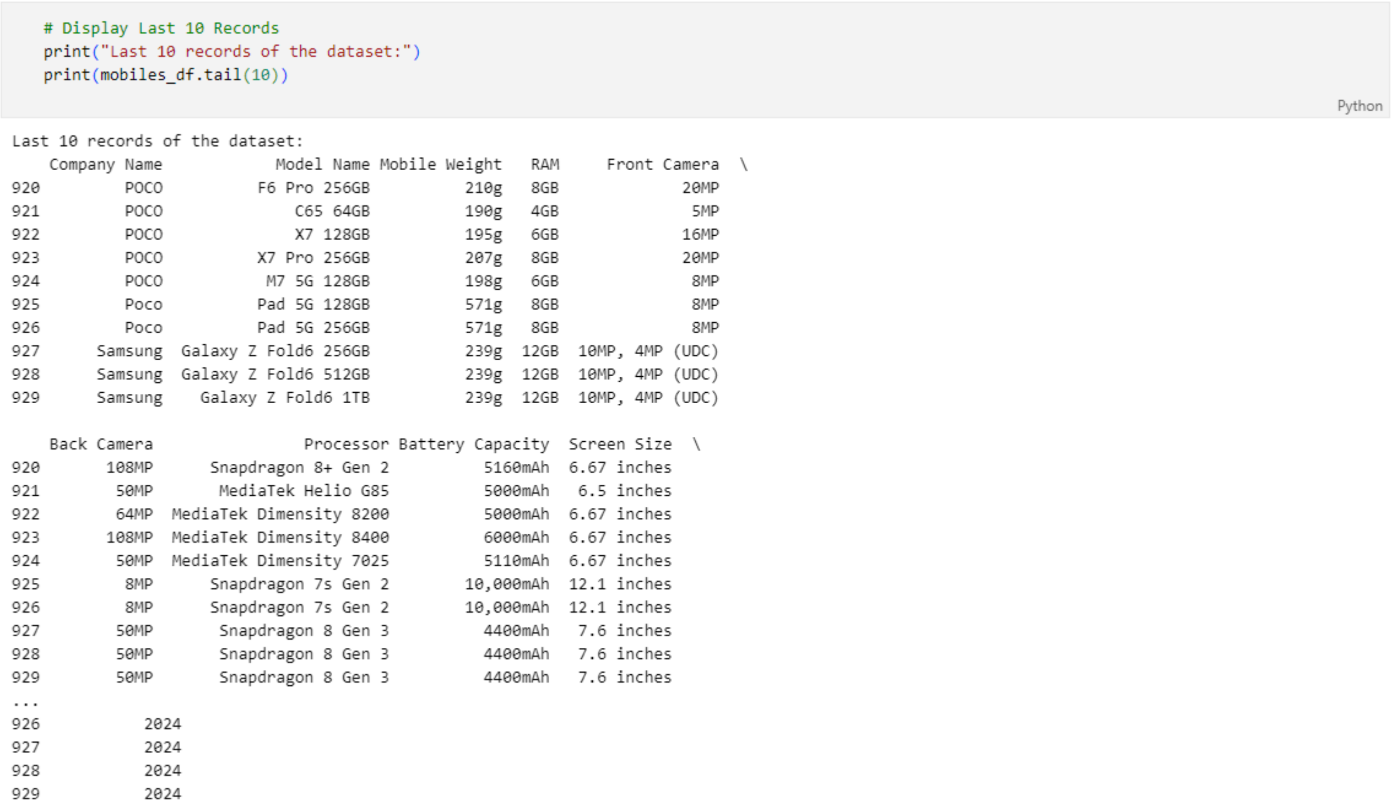


```python
# Display First 10 Records
print("First 10 records of the dataset:")
print(mobiles_df.head(10))
```

Python

```
First 10 records of the dataset:
  Company Name               Model Name Mobile Weight  RAM Front Camera  \
0        Apple          iPhone 16 128GB          174g  6GB         12MP   
1        Apple          iPhone 16 256GB          174g  6GB         12MP   
2        Apple          iPhone 16 512GB          174g  6GB         12MP   
3        Apple     iPhone 16 Plus 128GB          203g  6GB         12MP   
4        Apple     iPhone 16 Plus 256GB          203g  6GB         12MP   
5        Apple     iPhone 16 Plus 512GB          203g  6GB         12MP   
6        Apple      iPhone 16 Pro 128GB          206g  6GB    12MP / 4K   
7        Apple      iPhone 16 Pro 256GB          206g  8GB    12MP / 4K   
8        Apple      iPhone 16 Pro 512GB          206g  8GB    12MP / 4K   
9        Apple  iPhone 16 Pro Max 128GB          221g  6GB    12MP / 4K   

   Back Camera   Processor Battery Capacity Screen Size  \
0         48MP  A17 Bionic         3,600mAh  6.1 inches   
1         48MP  A17 Bionic         3,600mAh  6.1 inches   
2         48MP  A17 Bionic         3,600mAh  6.1 inches   
3         48MP  A17 Bionic         4,200mAh  6.7 inches   
4         48MP  A17 Bionic         4,200mAh  6.7 inches   
5         48MP  A17 Bionic         4,200mAh  6.7 inches   
6  50MP + 12MP     A17 Pro         4,400mAh  6.1 inches   
7  50MP + 12MP     A17 Pro         4,400mAh  6.1 inches   
8  50MP + 12MP     A17 Pro         4,400mAh  6.1 inches   
9  48MP + 12MP     A17 Pro         4,500mAh  6.7 inches   
...
6                 999.0                    3499           2024  
7                1049.0                    3699           2024  
8                1099.0                    3899           2024  
9                1099.0                    3799           2024  
```

10) Display last 10 records



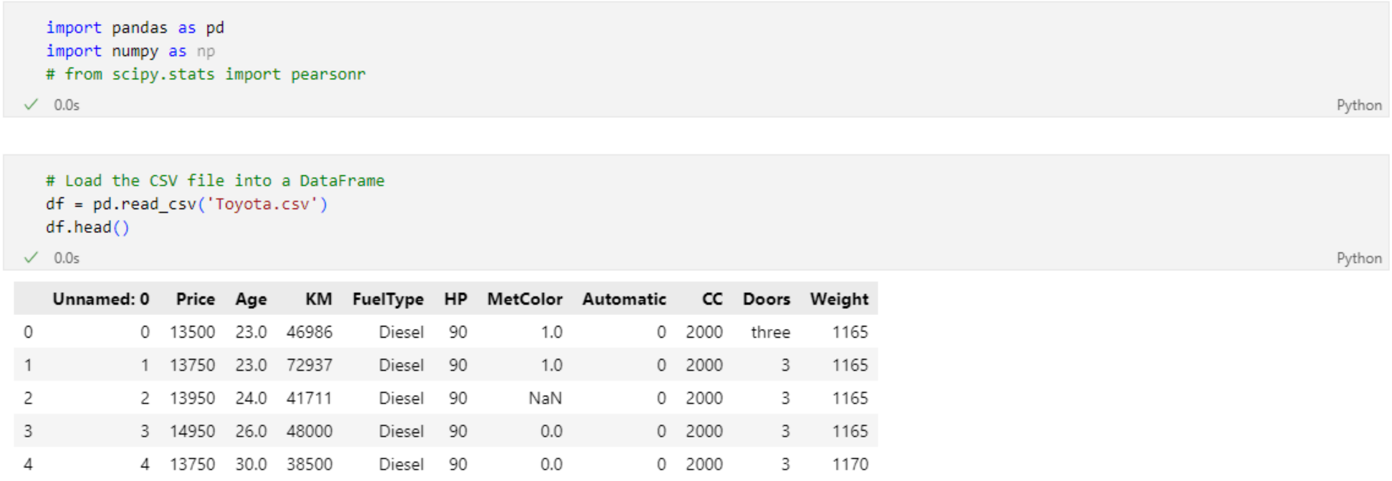
```python
# Display Last 10 Records
print("Last 10 records of the dataset:")
print(mobiles_df.tail(10))
```

Python

```
Last 10 records of the dataset:
    Company Name            Model Name Mobile Weight   RAM     Front Camera  \
920         POCO          F6 Pro 256GB          210g   8GB             20MP
921         POCO              C65 64GB          190g   4GB              5MP
922         POCO              X7 128GB          195g   6GB             16MP
923         POCO          X7 Pro 256GB          207g   8GB             20MP
924         POCO           M7 5G 128GB          198g   6GB              8MP
925         Poco          Pad 5G 128GB          571g   8GB              8MP
926         Poco          Pad 5G 256GB          571g   8GB              8MP
927      Samsung  Galaxy Z Fold6 256GB          239g  12GB  10MP, 4MP (UDC)
928      Samsung  Galaxy Z Fold6 512GB          239g  12GB  10MP, 4MP (UDC)
929      Samsung    Galaxy Z Fold6 1TB          239g  12GB  10MP, 4MP (UDC)

    Back Camera                Processor Battery Capacity  Screen Size  \
920       108MP      Snapdragon 8+ Gen 2          5160mAh  6.67 inches
921        50MP       MediaTek Helio G85          5000mAh   6.5 inches
922        64MP  MediaTek Dimensity 8200          5000mAh  6.67 inches
923       108MP  MediaTek Dimensity 8400          6000mAh  6.67 inches
924        50MP  MediaTek Dimensity 7025          5110mAh  6.67 inches
925         8MP      Snapdragon 7s Gen 2        10,000mAh  12.1 inches
926         8MP      Snapdragon 7s Gen 2        10,000mAh  12.1 inches
927        50MP       Snapdragon 8 Gen 3          4400mAh   7.6 inches
928        50MP       Snapdragon 8 Gen 3          4400mAh   7.6 inches
929        50MP       Snapdragon 8 Gen 3          4400mAh   7.6 inches
...
926           2024
927           2024
928           2024
929           2024
```

Practical Assignment 2

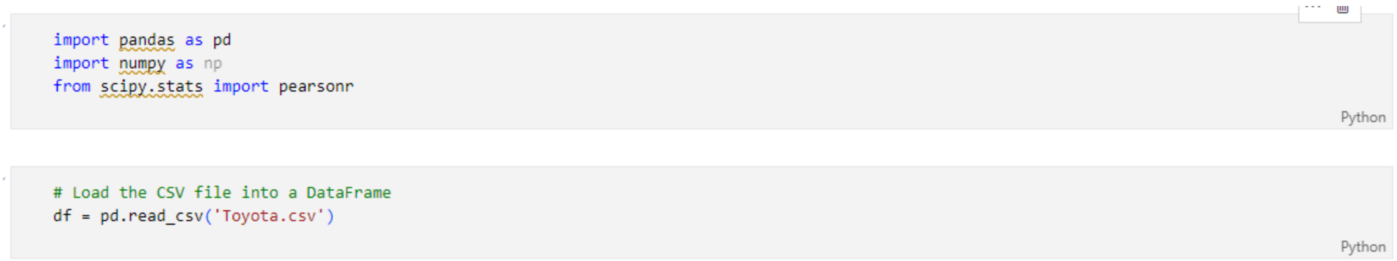
1) Upload Toyota.csv in dataframe df.



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

```
# Load the CSV file into a DataFrame
df = pd.read_csv('Toyota.csv')
df.head()
```

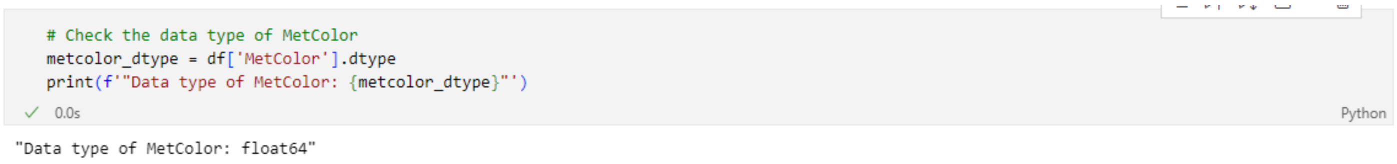
	Unnamed: 0	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	0	13500	23.0	46986	Diesel	90	1.0	0	2000	three	1165
1	1	13750	23.0	72937	Diesel	90	1.0	0	2000	3	1165
2	2	13950	24.0	41711	Diesel	90	NaN	0	2000	3	1165
3	3	14950	26.0	48000	Diesel	90	0.0	0	2000	3	1165
4	4	13750	30.0	38500	Diesel	90	0.0	0	2000	3	1170



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

```
# Load the CSV file into a DataFrame
df = pd.read_csv('Toyota.csv')
```

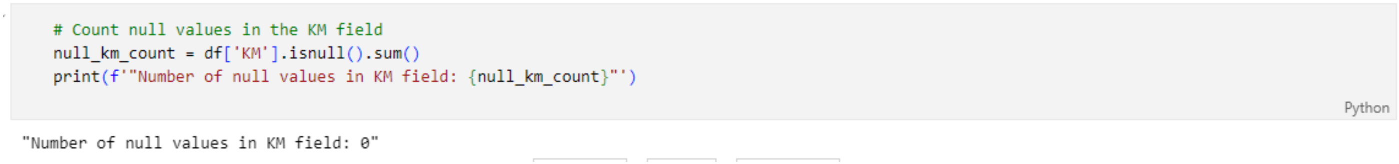
2) What is the data type of MetColor?



```
# Check the data type of MetColor
metcolor_dtype = df['MetColor'].dtype
print(f'"Data type of MetColor: {metcolor_dtype}"')
```

```
"Data type of MetColor: float64"
```

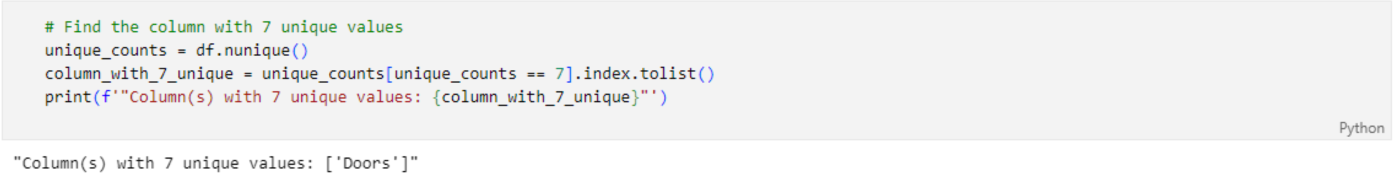
3) How many null value are there in KM field?



```
# Count null values in the KM field
null_km_count = df['KM'].isnull().sum()
print(f'"Number of null values in KM field: {null_km_count}"')
```

```
"Number of null values in KM field: 0"
```

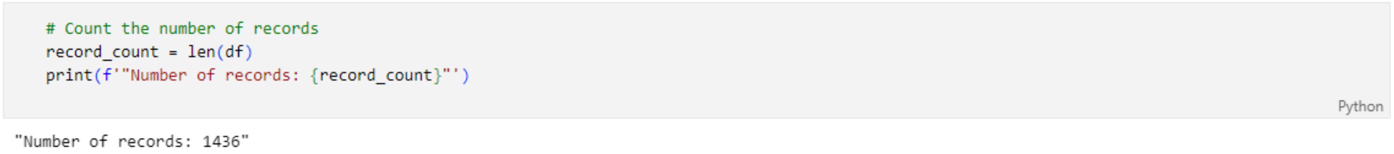
4) Which column has 7 unique values.



```
# Find the column with 7 unique values
unique_counts = df.nunique()
column_with_7_unique = unique_counts[unique_counts == 7].index.tolist()
print(f'"Column(s) with 7 unique values: {column_with_7_unique}"')
```

```
"Column(s) with 7 unique values: ['Doors']"
```

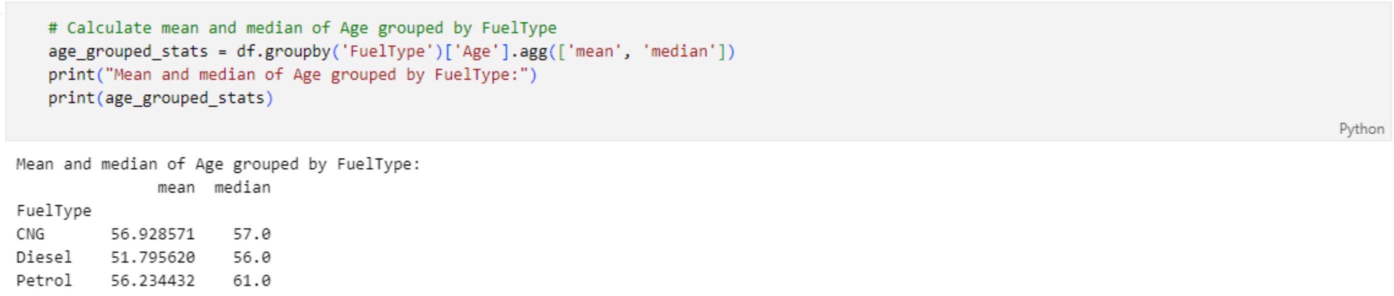
5) How many records are there?



```python
# Count the number of records
record_count = len(df)
print(f'"Number of records: {record_count}"')
```

```
"Number of records: 1436"
```

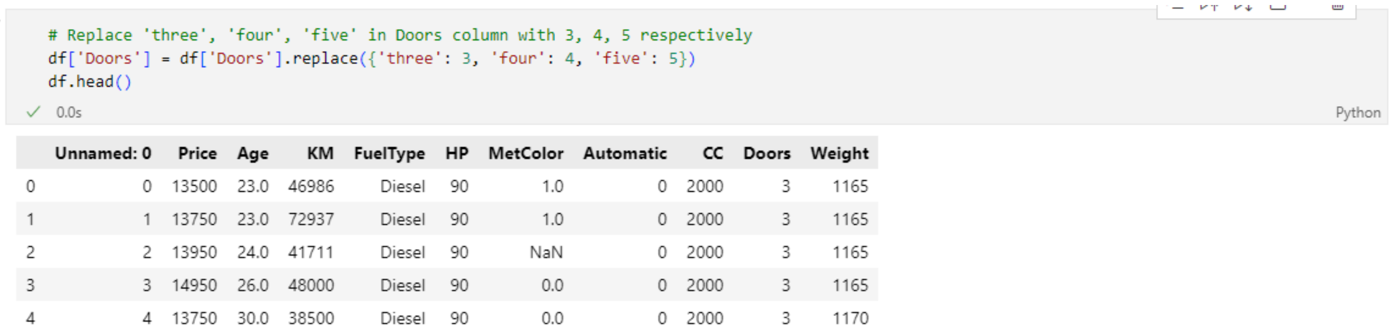
6) What is mean, median of age grouped by FuelType?



```python
# Calculate mean and median of Age grouped by FuelType
age_grouped_stats = df.groupby('FuelType')['Age'].agg(['mean', 'median'])
print("Mean and median of Age grouped by FuelType:")
print(age_grouped_stats)
```

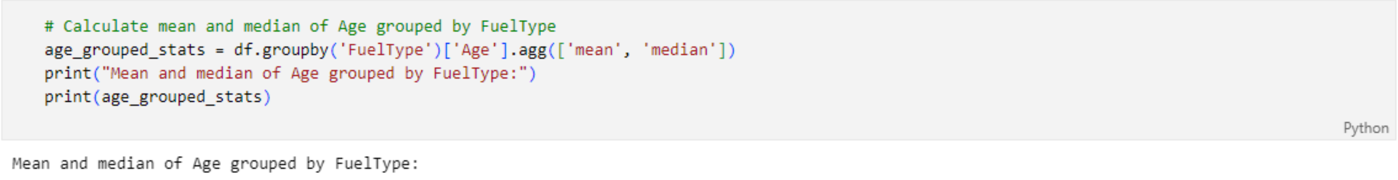
```
Mean and median of Age grouped by FuelType:
               mean  median
FuelType
CNG       56.928571    57.0
Diesel    51.795620    56.0
Petrol    56.234432    61.0
```

7) Replace three, four, five value in Doors column to 3,4,5 respectively.



```python
# Replace 'three', 'four', 'five' in Doors column with 3, 4, 5 respectively
df['Doors'] = df['Doors'].replace({'three': 3, 'four': 4, 'five': 5})
df.head()
```

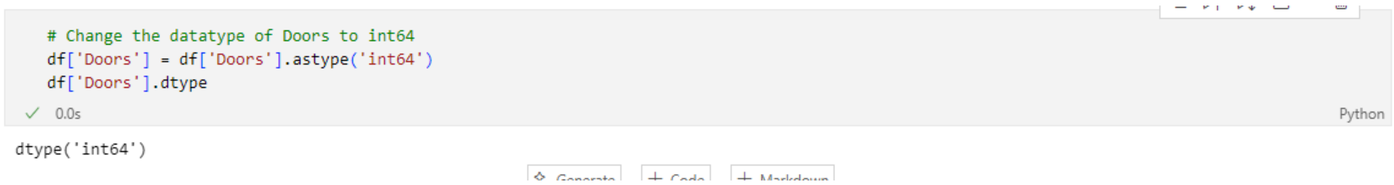
	Unnamed: 0	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	0	13500	23.0	46986	Diesel	90	1.0	0	2000	3	1165
1	1	13750	23.0	72937	Diesel	90	1.0	0	2000	3	1165
2	2	13950	24.0	41711	Diesel	90	NaN	0	2000	3	1165
3	3	14950	26.0	48000	Diesel	90	0.0	0	2000	3	1165
4	4	13750	30.0	38500	Diesel	90	0.0	0	2000	3	1170



```python
# Calculate mean and median of Age grouped by FuelType
age_grouped_stats = df.groupby('FuelType')['Age'].agg(['mean', 'median'])
print("Mean and median of Age grouped by FuelType:")
print(age_grouped_stats)
```

```
Mean and median of Age grouped by FuelType:
```

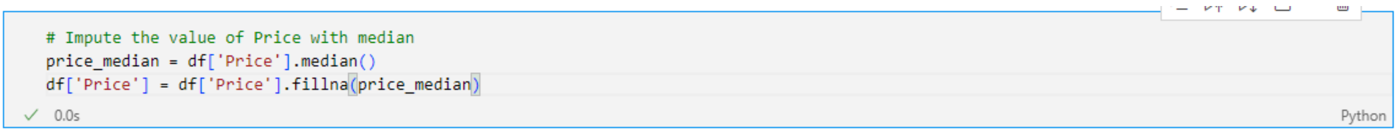
8) Change the datatype of Doors to int64.



```python
# Change the datatype of Doors to int64
df['Doors'] = df['Doors'].astype('int64')
df['Doors'].dtype
```

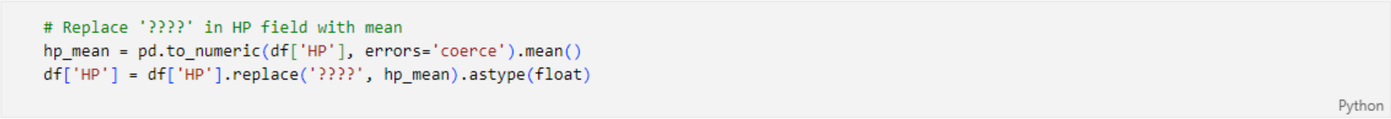
```
dtype('int64')
```

9) Impute the value of Price with median.



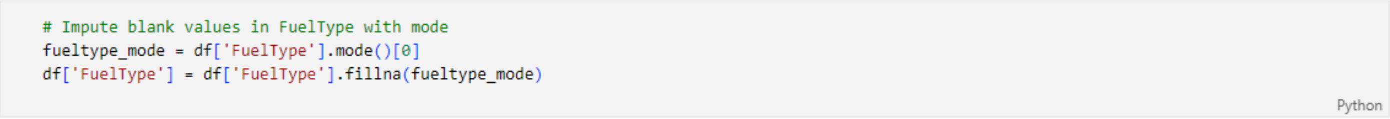
```python
# Impute the value of Price with median
price_median = df['Price'].median()
df['Price'] = df['Price'].fillna(price_median)
```

10) Replace ???? in HP field with mean.



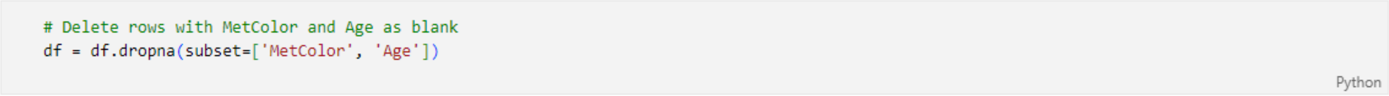
```python
# Replace '????' in HP field with mean
hp_mean = pd.to_numeric(df['HP'], errors='coerce').mean()
df['HP'] = df['HP'].replace('????', hp_mean).astype(float)
```

11) Impute blank values in FuelType with Mode.



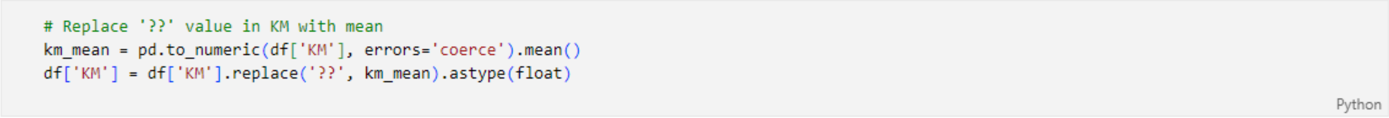
```python
# Impute blank values in FuelType with mode
fueltype_mode = df['FuelType'].mode()[0]
df['FuelType'] = df['FuelType'].fillna(fueltype_mode)
```

12) Delete the rows with MetColor and Age as blank.



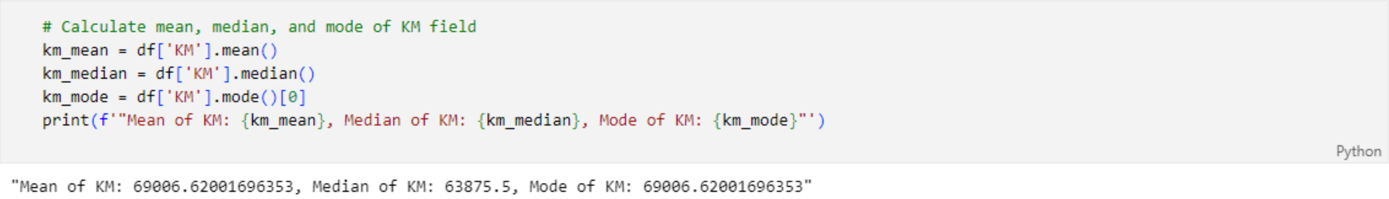
```python
# Delete rows with MetColor and Age as blank
df = df.dropna(subset=['MetColor', 'Age'])
```

13) Replace ?? value in KM with Mean.



```python
# Replace '??' value in KM with mean
km_mean = pd.to_numeric(df['KM'], errors='coerce').mean()
df['KM'] = df['KM'].replace('??', km_mean).astype(float)
```

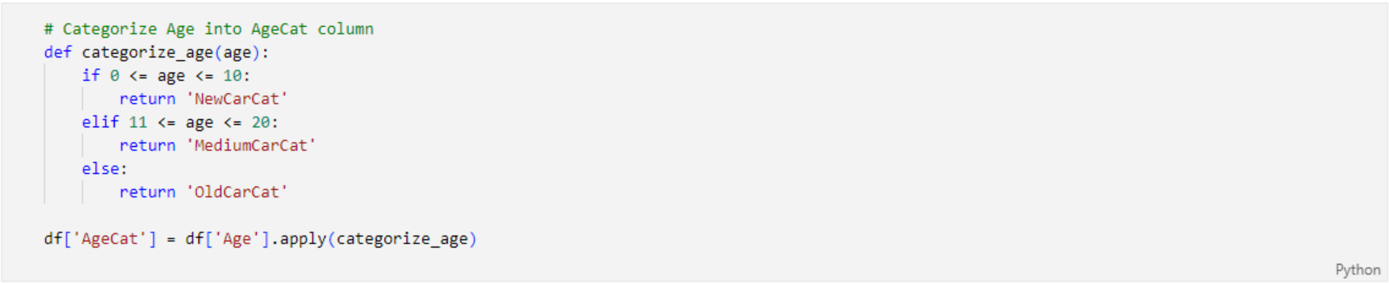
14) What is the mean, median and mode of KM field.



```python
# Calculate mean, median, and mode of KM field
km_mean = df['KM'].mean()
km_median = df['KM'].median()
km_mode = df['KM'].mode()[0]
print(f'"Mean of KM: {km_mean}, Median of KM: {km_median}, Mode of KM: {km_mode}"')
```

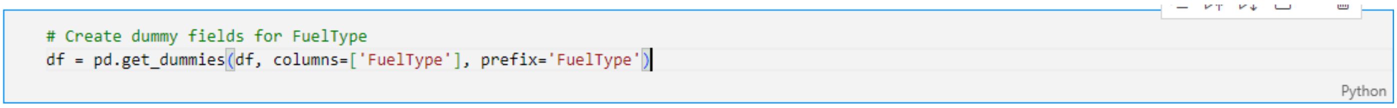
```
"Mean of KM: 69006.62001696353, Median of KM: 63875.5, Mode of KM: 69006.62001696353"
```

15) Categorise Age into AgeCat column with 0-10 NewCarCat, 11-20 MediumCarCat, 21- highest value – OldCarCat.



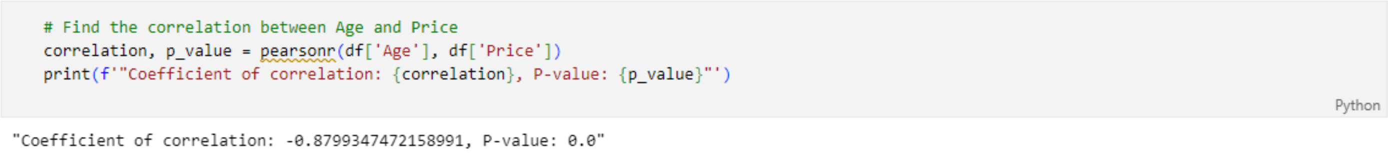
```python
# Categorize Age into AgeCat column
def categorize_age(age):
    if 0 <= age <= 10:
        return 'NewCarCat'
    elif 11 <= age <= 20:
        return 'MediumCarCat'
    else:
        return 'OldCarCat'

df['AgeCat'] = df['Age'].apply(categorize_age)
```

16) Create Dummy fields for FuelType.



```python
# Create dummy fields for FuelType
df = pd.get_dummies(df, columns=['FuelType'], prefix='FuelType')
```

17) Find the correlation between age and price., what is coefficient of correlation and p-value.



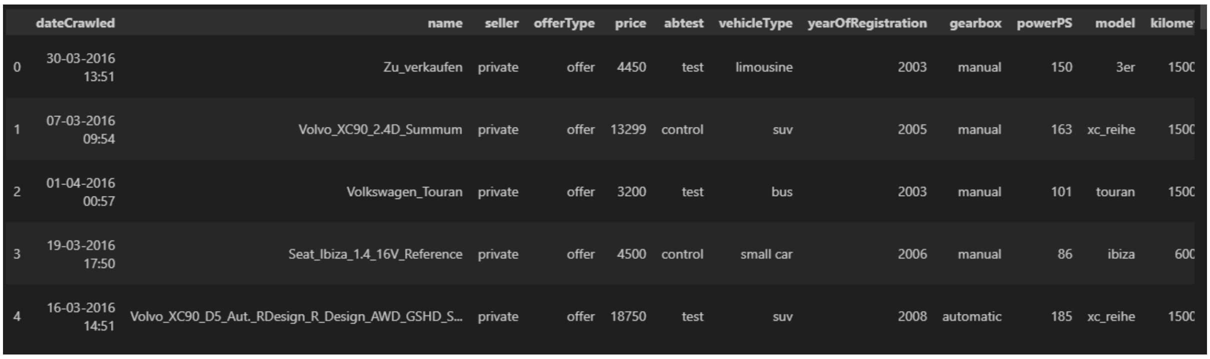
```
# Find the correlation between Age and Price
correlation, p_value = pearsonr(df['Age'], df['Price'])
print(f'"Coefficient of correlation: {correlation}, P-value: {p_value}"')
```

```
"Coefficient of correlation: -0.8799347472158991, P-value: 0.0"
```

Practical Assignment 3

Q1. Read cars_sampled.csv

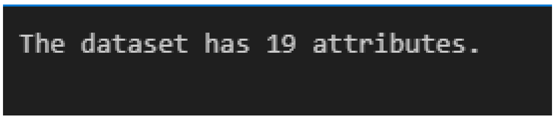
```
import pandas as pd
import seaborn as sns
cars = pd.read_csv('cars_sampled.csv')
cars.head()
```



	dateCrawled	name	seller	offerType	price	abtest	vehicleType	yearOfRegistration	gearbox	powerPS	model	kilome
0	30-03-2016 13:51	Zu_verkaufen	private	offer	4450	test	limousine	2003	manual	150	3er	150C
1	07-03-2016 09:54	Volvo_XC90_2.4D_Summum	private	offer	13299	control	suv	2005	manual	163	xc_reihe	150C
2	01-04-2016 00:57	Volkswagen_Touran	private	offer	3200	test	bus	2003	manual	101	touran	150C
3	19-03-2016 17:50	Seat_Ibiza_1.4_16V_Reference	private	offer	4500	control	small car	2006	manual	86	ibiza	60C
4	16-03-2016 14:51	Volvo_XC90_D5_Aut._RDesign_R_Design_AWD_GSHD_S...	private	offer	18750	test	suv	2008	automatic	185	xc_reihe	150C

Q2. How many attributes are there?

```
num_attributes = cars.shape[1]
print(f'The dataset has {num_attributes} attributes.')
```

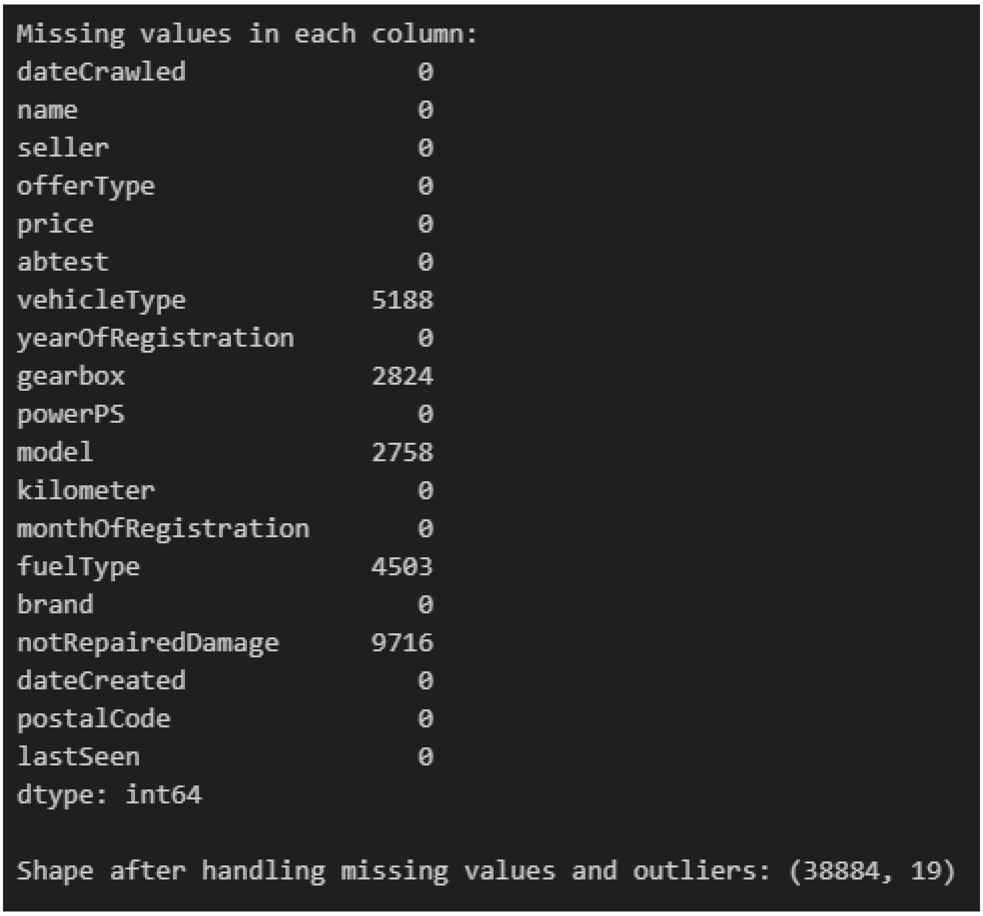


```
The dataset has 19 attributes.
```

Q3. Fill the missing values, impute data, check and eliminate for outlier.

```
# Check missing values
print('Missing values in each column:')
print(cars.isnull().sum())

# Fill missing values with mode for categorical variables
cars['vehicleType'] = cars['vehicleType'].fillna(cars['vehicleType'].mode()[0])
cars['gearbox'] = cars['gearbox'].fillna(cars['gearbox'].mode()[0])
cars['model'] = cars['model'].fillna(cars['model'].mode()[0])
cars['fuelType'] = cars['fuelType'].fillna(cars['fuelType'].mode()[0])
cars['notRepairedDamage'] =
cars['notRepairedDamage'].fillna(cars['notRepairedDamage'].mode()[0])
```

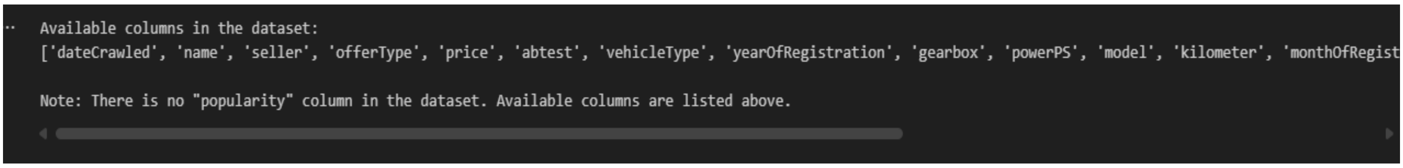
```
# Fill missing values with median for numerical variables
numeric_cols = ['price', 'powerPS', 'kilometer']
for col in numeric_cols:
   cars[col] = cars[col].fillna(cars[col].median())

# Handle outliers using IQR method for numeric columns
for col in numeric_cols:
   Q1 = cars[col].quantile(0.25)
   Q3 = cars[col].quantile(0.75)
   IQR = Q3 - Q1
   lower_bound = Q1 - 1.5 * IQR
   upper_bound = Q3 + 1.5 * IQR

   # Remove outliers
   cars = cars[(cars[col] >= lower_bound) & (cars[col] <= upper_bound)]

print('\nShape after handling missing values and outliers:', cars.shape)
```



```
Missing values in each column:
dateCrawled               0
name                      0
seller                    0
offerType                 0
price                     0
abtest                    0
vehicleType            5188
yearOfRegistration        0
gearbox                2824
powerPS                   0
model                  2758
kilometer                 0
monthOfRegistration       0
fuelType               4503
brand                     0
notRepairedDamage      9716
dateCreated               0
postalCode                0
lastSeen                  0
dtype: int64

Shape after handling missing values and outliers: (38884, 19)
```

Q4. How many unique values are there in popularity column?

```
print('Available columns in the dataset:')
print(cars.columns.tolist())
print('\nNote: There is no "popularity" column in the dataset. Available columns are listed above.')
```

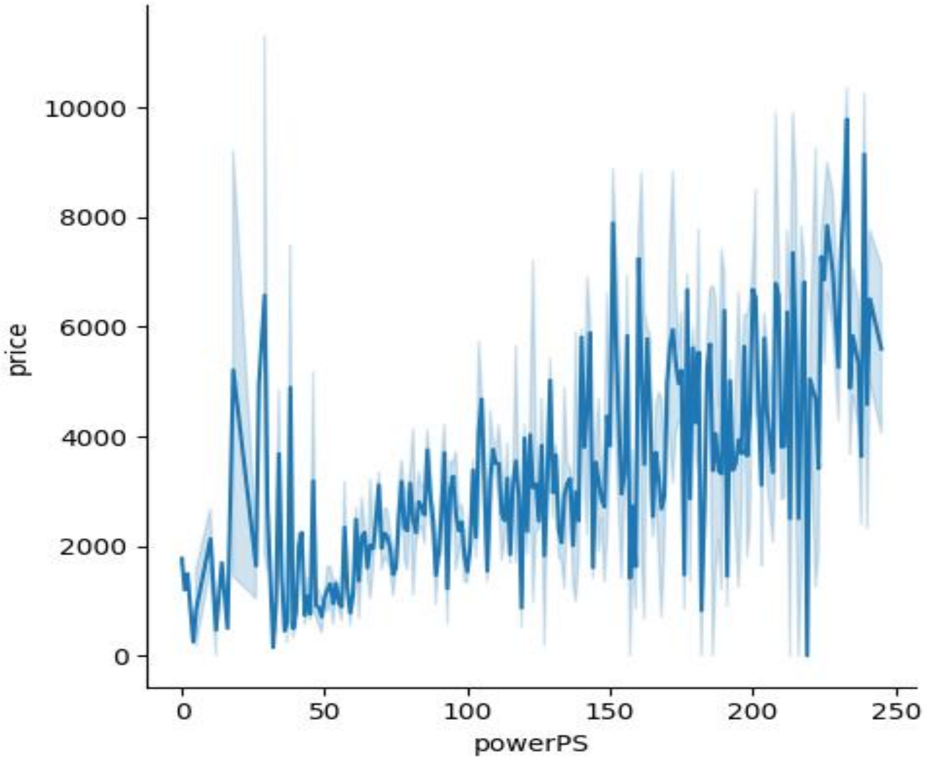


```
Available columns in the dataset:
['dateCrawled', 'name', 'seller', 'offerType', 'price', 'abtest', 'vehicleType', 'yearOfRegistration', 'gearbox', 'powerPS', 'model', 'kilometer', 'monthOfRegist

Note: There is no "popularity" column in the dataset. Available columns are listed above.
```

Q5. Using matplot/seaborn

Draw lineplot using relplot function of seaborn between price and powerPS

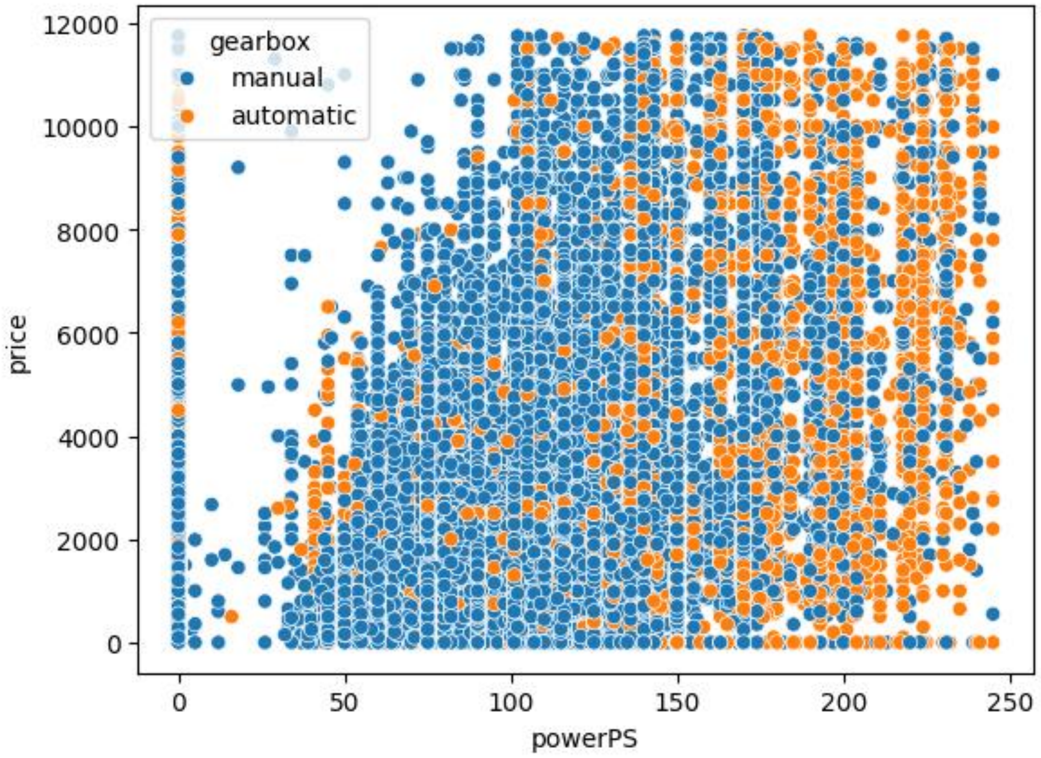
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.relplot(x='powerPS', y='price', kind='line', data=cars)
plt.show()
```



/

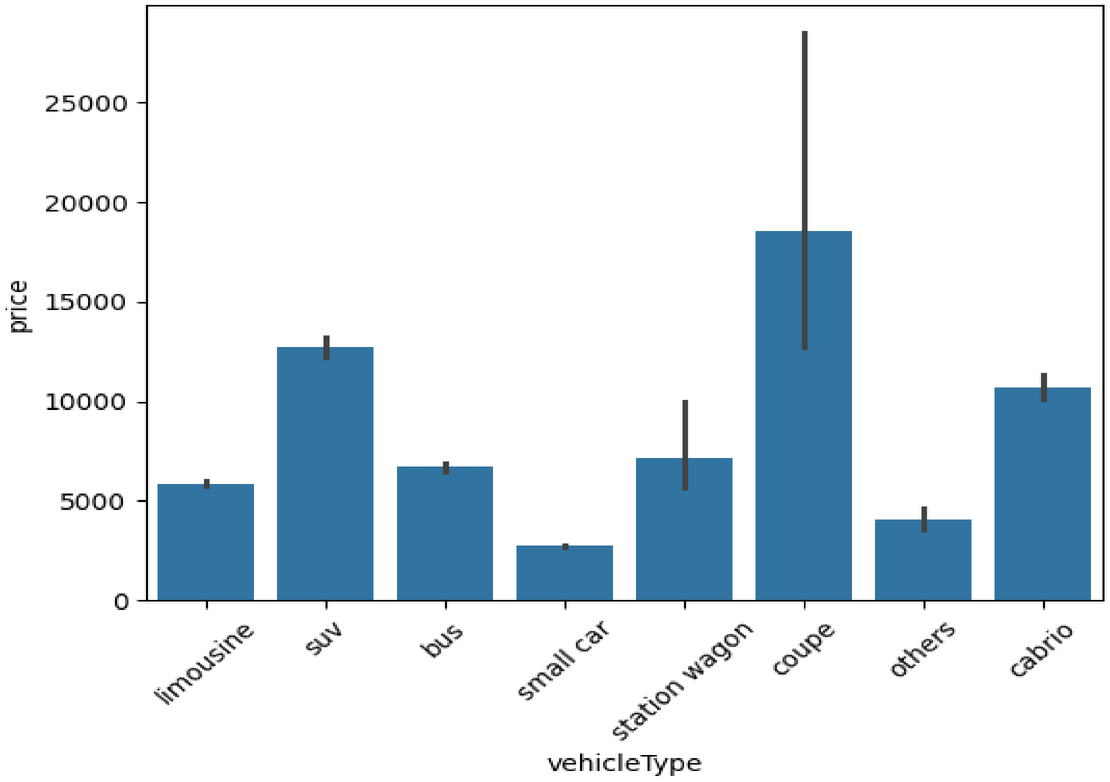
Draw scatter plot between price and powerPS, categorise them on gearbox

```
sns.scatterplot(x='powerPS', y='price', hue='gearbox', data=cars)
plt.show()
```



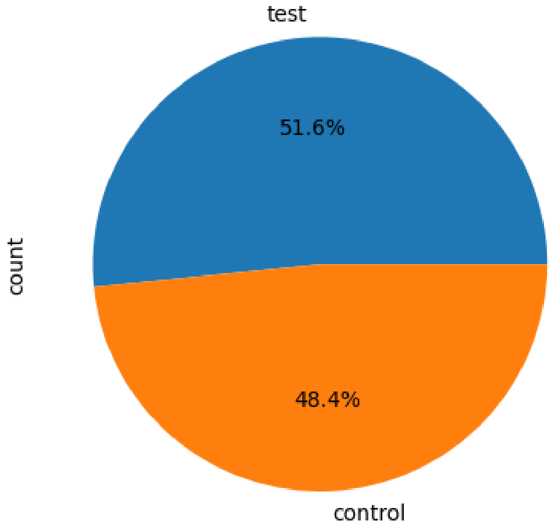
Draw barplot between VehicleType and price using seaborn function

```
import matplotlib.pyplot as plt
sns.barplot(x='vehicleType', y='price', data=cars)
plt.xticks(rotation=45)
plt.show()
```



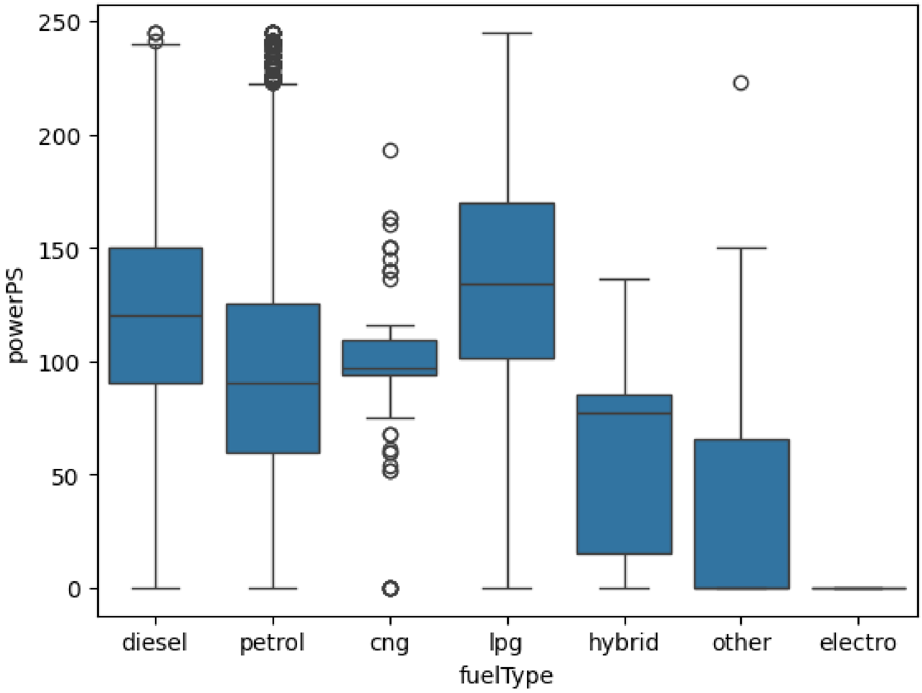
Draw piechart on the basis of abtest.

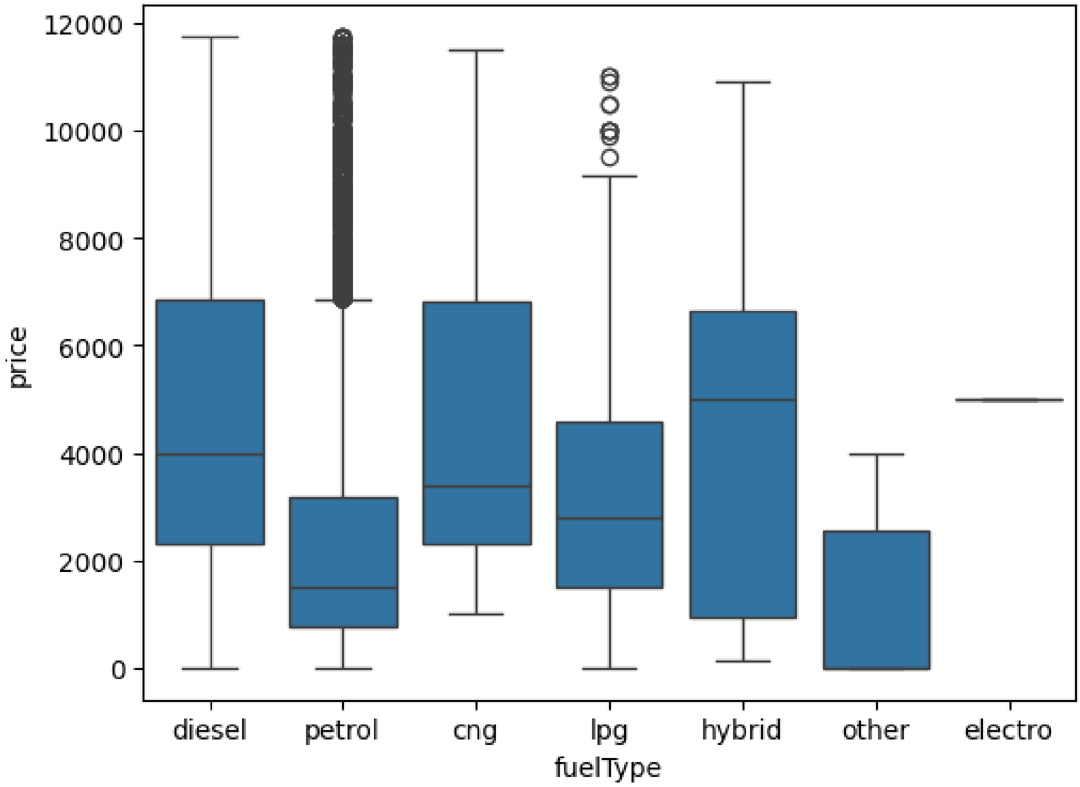
```
abtest_counts = cars['abtest'].value_counts()
abtest_counts.plot.pie(autopct='%1.1f%%')
plt.show()
```



Draw boxplot price, powerPS on the basis of fuelType

```
sns.boxplot(x='fuelType', y='price', data=cars)
plt.show()
sns.boxplot(x='fuelType', y='powerPS', data=cars)
plt.show()
```

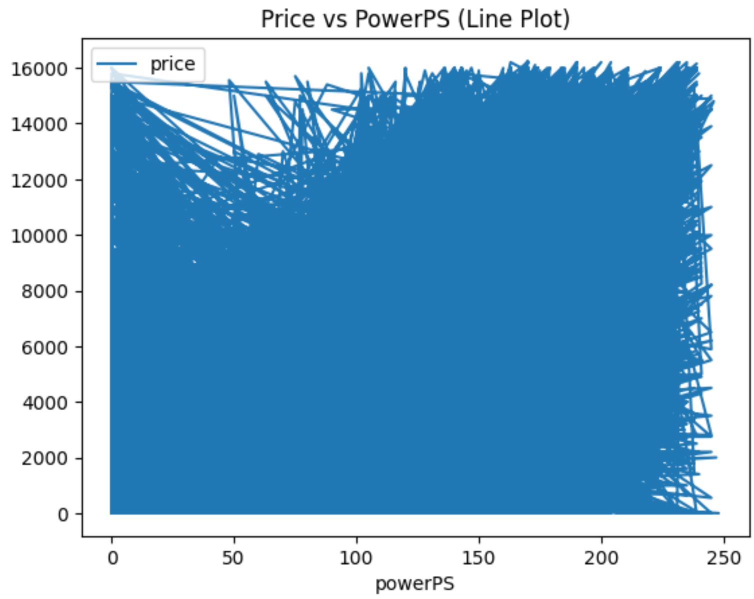



12000
10000
8000
6000
4000
2000
0
price
diesel
petrol
cng
lpg
hybrid
other
electro
fuelType

Q6. Using pandas draw the above four plots.

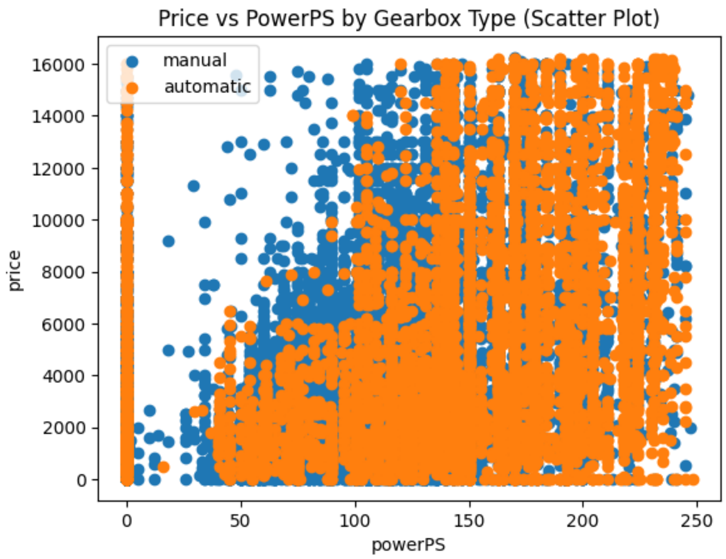
Line plot of price vs powerPS

```
cars.plot(x='powerPS', y='price', kind='line')
plt.title('Price vs PowerPS (Line Plot)')
plt.show()
```



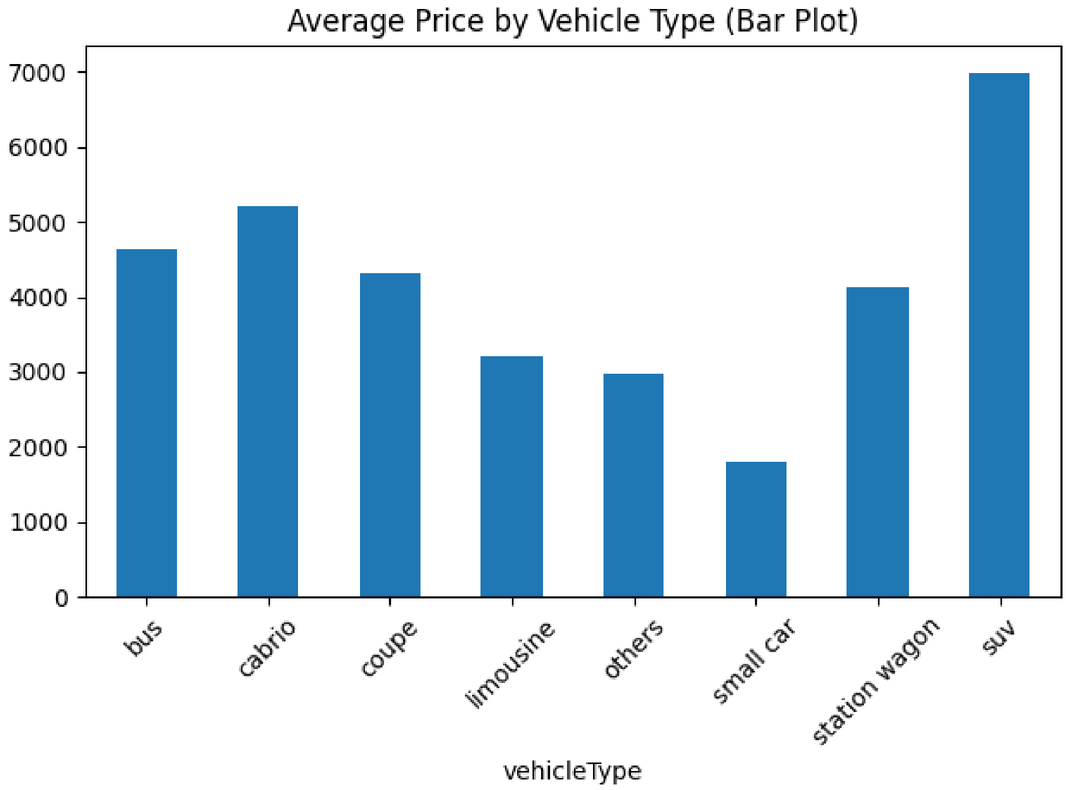
Scatter plot between price and powerPS, categorized by gearbox

```
for gearbox_type in cars['gearbox'].unique():
subset = cars[cars['gearbox'] == gearbox_type]
plt.scatter(subset['powerPS'], subset['price'], label=gearbox_type)
plt.xlabel('powerPS')
plt.ylabel('price')
plt.legend()
plt.title('Price vs PowerPS by Gearbox Type (Scatter Plot)')
plt.show()
```



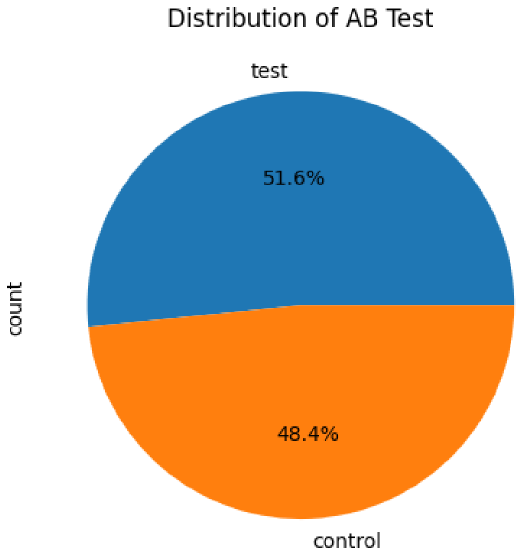
Bar plot between VehicleType and price

```
cars.groupby('vehicleType')['price'].mean().plot(kind='bar')
plt.xticks(rotation=45)
plt.title('Average Price by Vehicle Type (Bar Plot)')
plt.tight_layout()
plt.show()
```



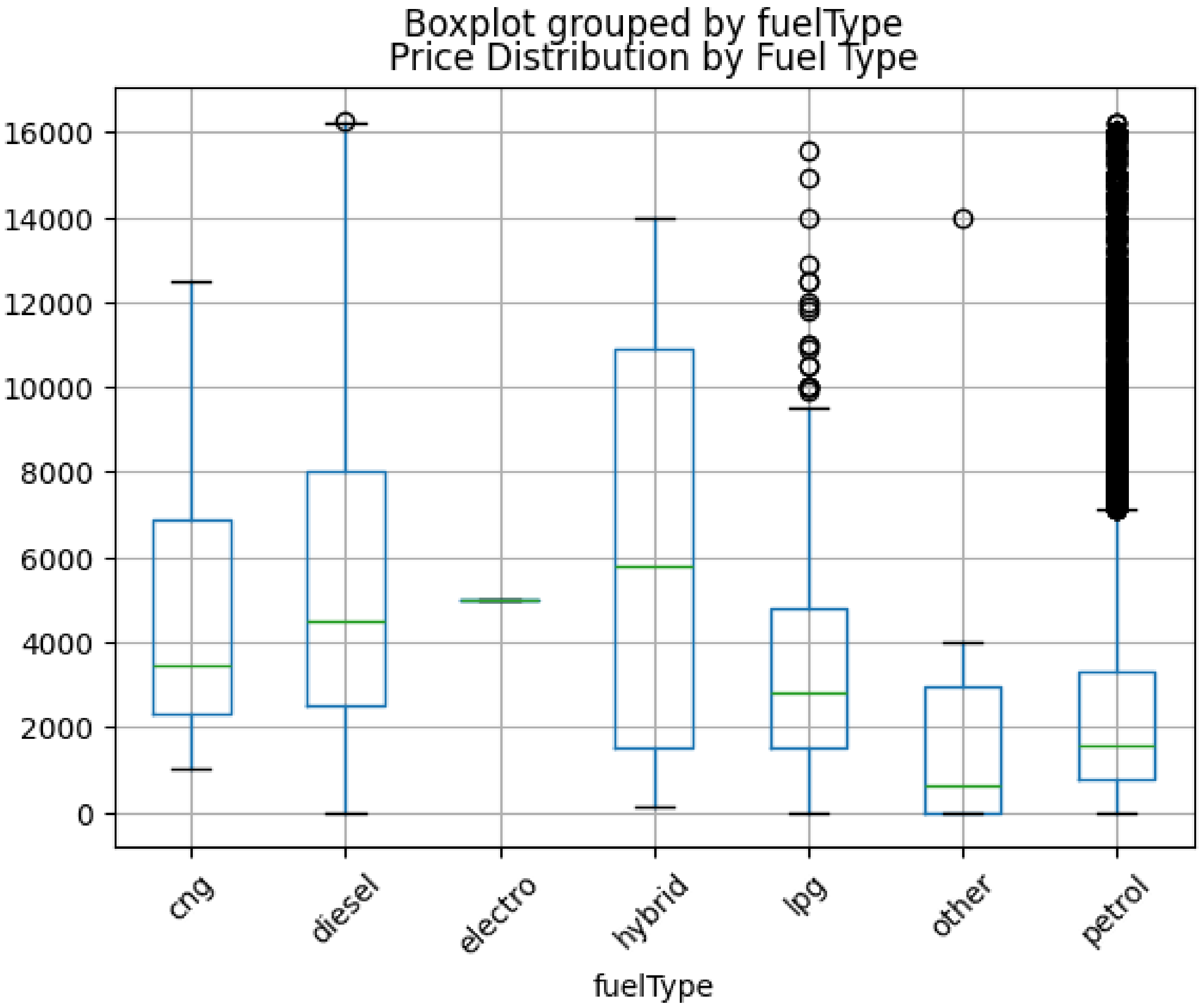
Pie chart for abtest

```
cars['abtest'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Distribution of AB Test')
plt.show()
```



Box plots for price and powerPS by fuelType

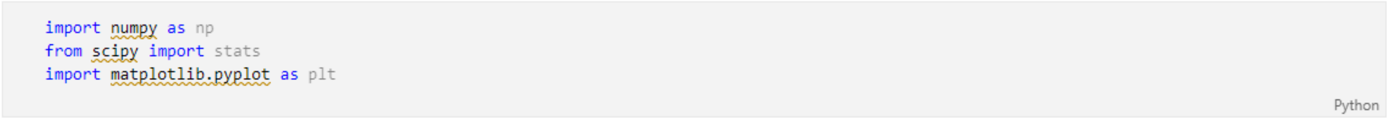
```
cars.boxplot(column='price', by='fuelType')
plt.xticks(rotation=45)
plt.title('Price Distribution by Fuel Type')
plt.show()

cars.boxplot(column='powerPS', by='fuelType')
plt.xticks(rotation=45)
plt.title('PowerPS Distribution by Fuel Type')
plt.show()
```



Practical Assignment 4

Q1) Perform one sample z-test in python for the given problem:

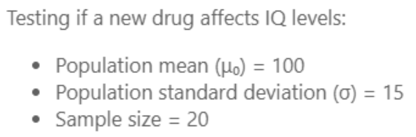
In this notebook, we'll perform one sample z-test and one sample t-test on different datasets.



```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

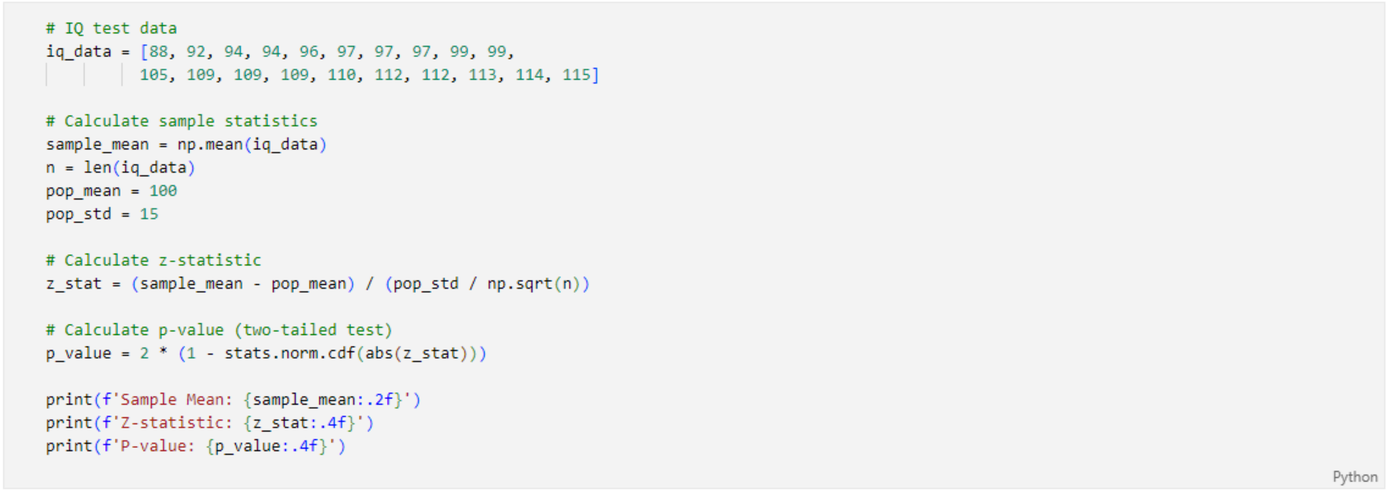
Suppose the IQ in a certain population is normally distributed with a mean of μ = 100 and standard deviation of σ = 15.

A researcher wants to know if a new drug affects IQ levels, so he recruits 20 patients to try it and records their IQ levels.
data = [88, 92, 94, 94, 96, 97, 97, 97, 99, 99,
105, 109, 109, 109, 110, 112, 112, 113, 114, 115]



Testing if a new drug affects IQ levels:

- Population mean (μ_0) = 100
- Population standard deviation (σ) = 15
- Sample size = 20



```
# IQ test data
iq_data = [88, 92, 94, 94, 96, 97, 97, 97, 99, 99,
           105, 109, 109, 109, 110, 112, 112, 113, 114, 115]

# Calculate sample statistics
sample_mean = np.mean(iq_data)
n = len(iq_data)
pop_mean = 100
pop_std = 15

# Calculate z-statistic
z_stat = (sample_mean - pop_mean) / (pop_std / np.sqrt(n))

# Calculate p-value (two-tailed test)
p_value = 2 * (1 - stats.norm.cdf(abs(z_stat)))

print(f'Sample Mean: {sample_mean:.2f}')
print(f'Z-statistic: {z_stat:.4f}')
print(f'P-value: {p_value:.4f}')
```

Explain the result of z-test for the above problem.

Z-Test Results Explanation

Hypotheses:

- H_0: μ = 100 (The drug has no effect on IQ)

- H_1: μ ≠ 100 (The drug affects IQ)

Using $\alpha = 0.05$:

- If the p-value < 0.05, we reject H_0

- If the p-value ≥ 0.05, we fail to reject H_0

The sample mean is slightly higher than the population mean, but we need to check if this difference is statistically significant. The z-test helps us determine this by considering both the sample size and population standard deviation.

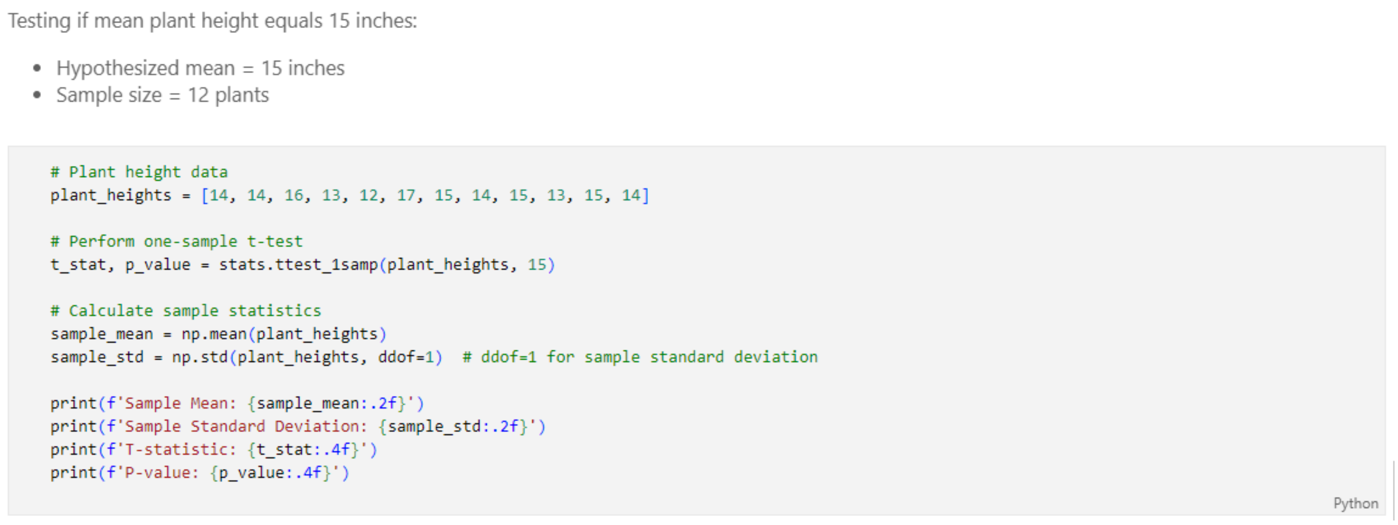
Based on the p-value, if it's less than 0.05, we would conclude that the drug has a significant effect on IQ levels. If it's greater than 0.05, we would conclude that there isn't enough evidence to say the drug affects IQ levels.

Q2) Perform One Sample t-Test in Python for the given problem

Suppose a botanist wants to know if the mean height of a certain species of plant is equal to 15 inches. She collects a random sample of 12 plants and records each of their heights in inches.

data = [14, 14, 16, 13, 12, 17, 15, 14, 15, 13, 15, 14]

Explain the result of t-test on the above sample.



Testing if mean plant height equals 15 inches:

- Hypothesized mean = 15 inches
- Sample size = 12 plants

```python
# Plant height data
plant_heights = [14, 14, 16, 13, 12, 17, 15, 14, 15, 13, 15, 14]

# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(plant_heights, 15)

# Calculate sample statistics
sample_mean = np.mean(plant_heights)
sample_std = np.std(plant_heights, ddof=1)  # ddof=1 for sample standard deviation

print(f'Sample Mean: {sample_mean:.2f}')
print(f'Sample Standard Deviation: {sample_std:.2f}')
print(f'T-statistic: {t_stat:.4f}')
print(f'P-value: {p_value:.4f}')
```

T-Test Results Explanation

Hypotheses:
- H_0: $\mu = 15$ (The mean height equals 15 inches)
- H_1: $\mu \neq 15$ (The mean height differs from 15 inches)

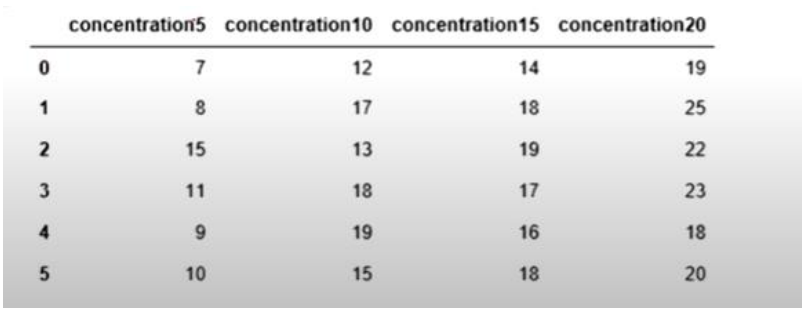
Using $\alpha = 0.05$:
- If the p-value < 0.05, we reject H_0
- If the p-value ≥ 0.05, we fail to reject H_0

We use a t-test instead of a z-test here because:
1. We don't know the population standard deviation
2. The sample size is small ($n < 30$)

The t-test takes into account the uncertainty in estimating the population standard deviation from the sample. If the p-value is less than 0.05, we would conclude that the mean height is significantly different from 15 inches. If it's greater than 0.05, we would conclude that there isn't enough evidence to say the mean height differs from 15 inches.

Practical Assignment 5

Create TensileStrength.xlsx file with the given data

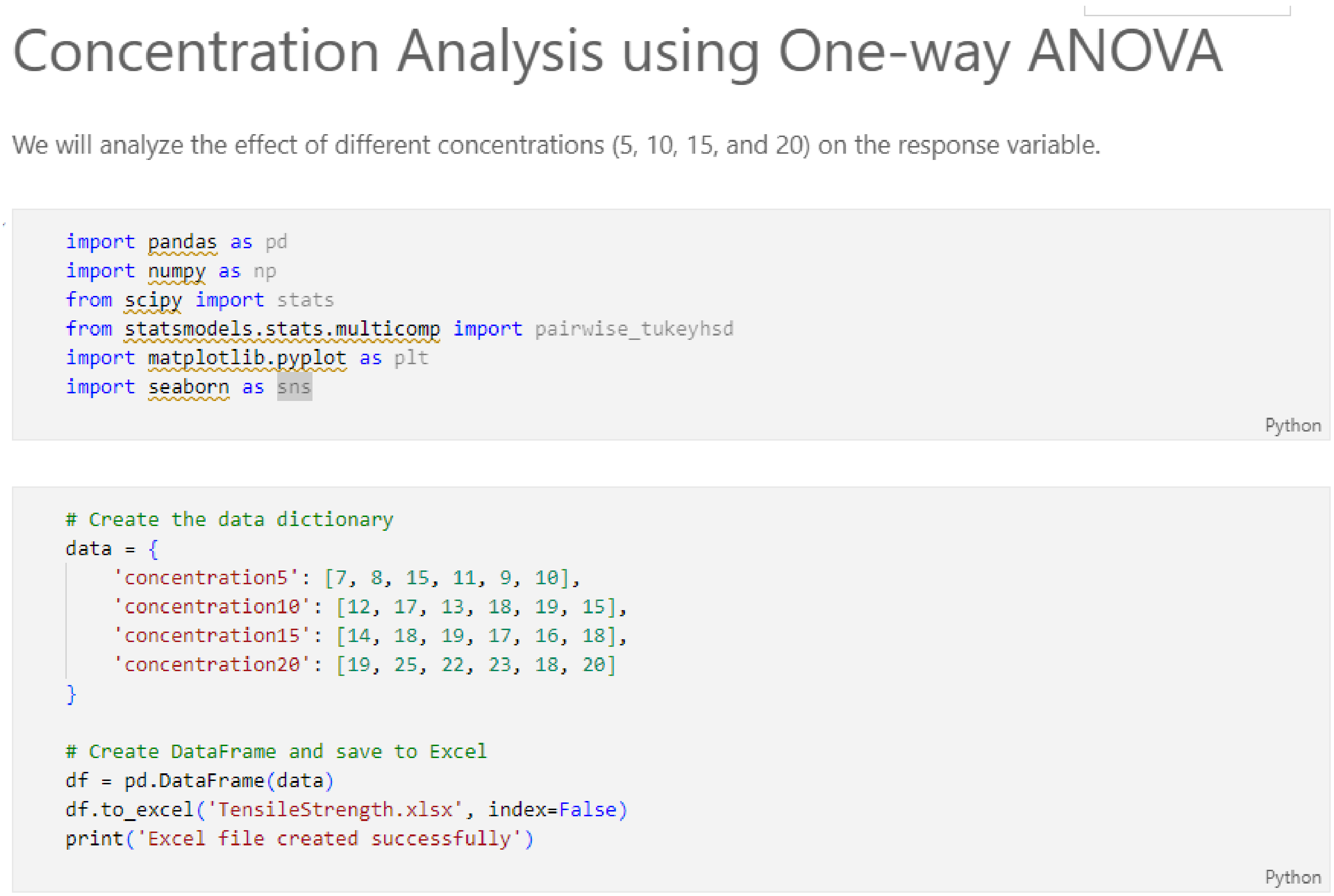


	concentration5	concentration10	concentration15	concentration20
0	7	12	14	19
1	8	17	18	25
2	15	13	19	22
3	11	18	17	23
4	9	19	16	18
5	10	15	18	20

Read the excel file in dataframe, use the melt command of pandas to pivot the table

Run one way anova to check whether the null hypothesis is rejected or not rejected.

If null hypothesis is rejected test for Posthoc test (Tukey's) and discuss the result.

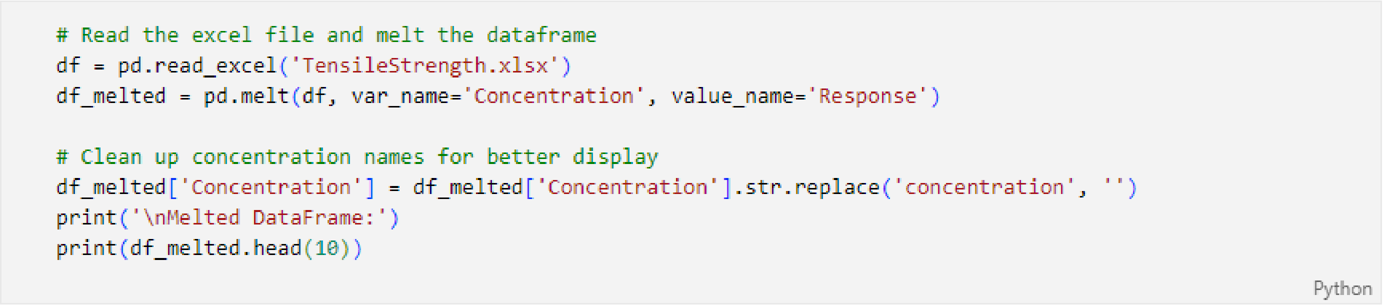


Concentration Analysis using One-way ANOVA

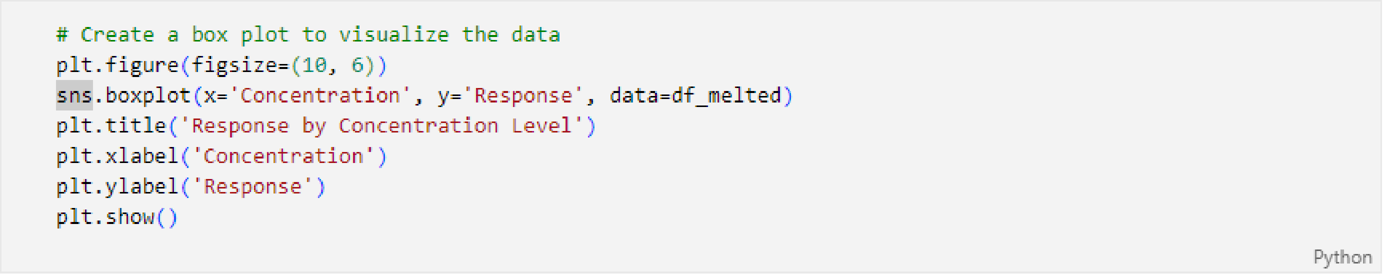
We will analyze the effect of different concentrations (5, 10, 15, and 20) on the response variable.

```python
import pandas as pd
import numpy as np
from scipy import stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import matplotlib.pyplot as plt
import seaborn as sns
```

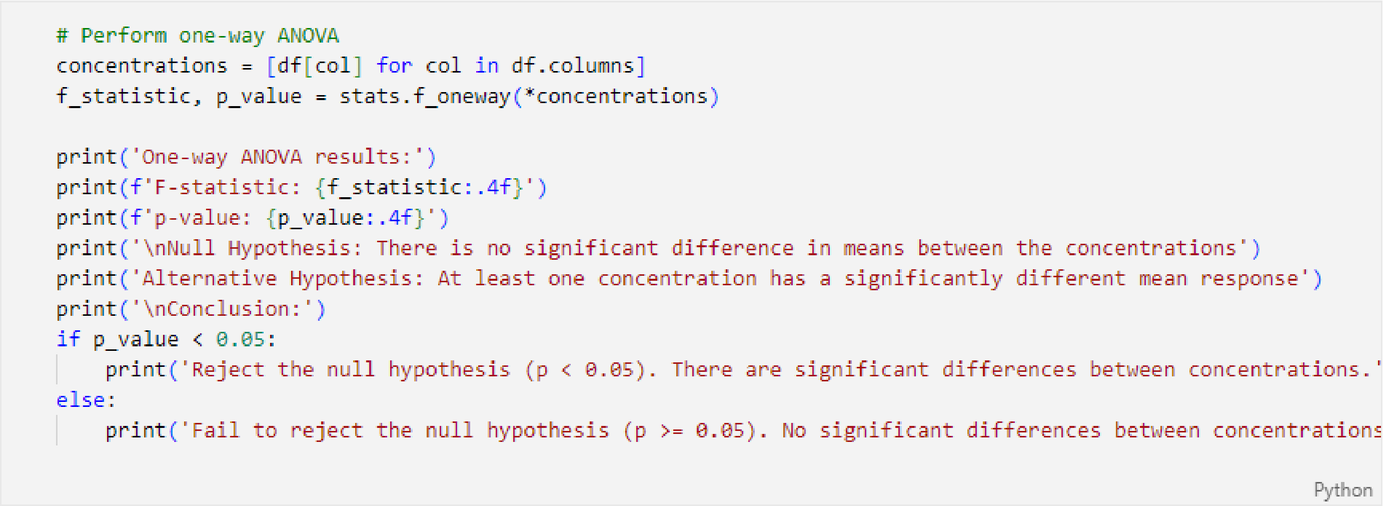
```python
# Create the data dictionary
data = {
    'concentration5': [7, 8, 15, 11, 9, 10],
    'concentration10': [12, 17, 13, 18, 19, 15],
    'concentration15': [14, 18, 19, 17, 16, 18],
    'concentration20': [19, 25, 22, 23, 18, 20]
}

# Create DataFrame and save to Excel
df = pd.DataFrame(data)
df.to_excel('TensileStrength.xlsx', index=False)
print('Excel file created successfully')
```



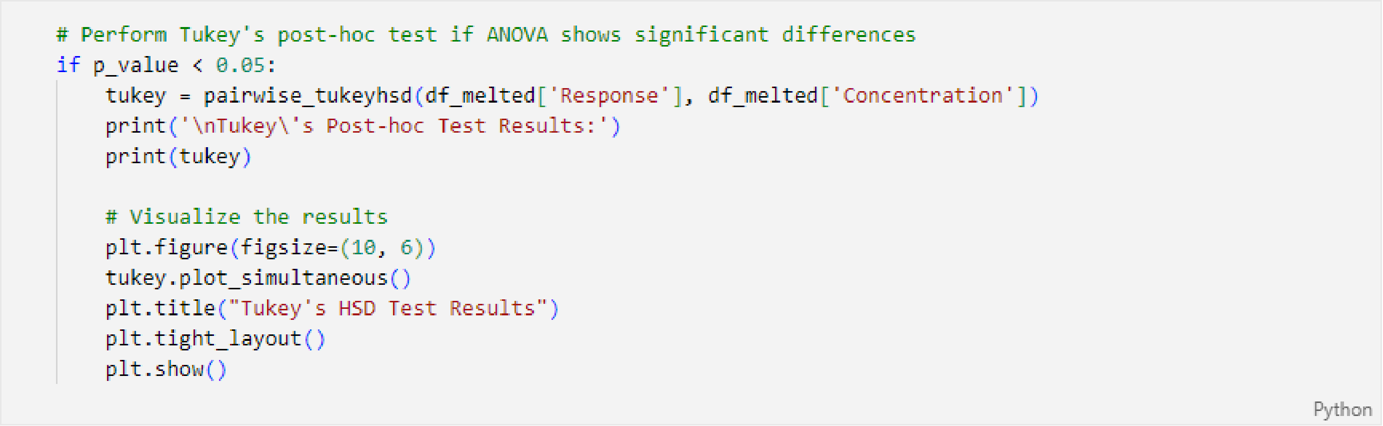
```
# Read the excel file and melt the dataframe
df = pd.read_excel('TensileStrength.xlsx')
df_melted = pd.melt(df, var_name='Concentration', value_name='Response')

# Clean up concentration names for better display
df_melted['Concentration'] = df_melted['Concentration'].str.replace('concentration', '')
print('\nMelted DataFrame:')
print(df_melted.head(10))
```



```
# Create a box plot to visualize the data
plt.figure(figsize=(10, 6))
sns.boxplot(x='Concentration', y='Response', data=df_melted)
plt.title('Response by Concentration Level')
plt.xlabel('Concentration')
plt.ylabel('Response')
plt.show()
```



```
# Perform one-way ANOVA
concentrations = [df[col] for col in df.columns]
f_statistic, p_value = stats.f_oneway(*concentrations)

print('One-way ANOVA results:')
print(f'F-statistic: {f_statistic:.4f}')
print(f'p-value: {p_value:.4f}')
print('\nNull Hypothesis: There is no significant difference in means between the concentrations')
print('Alternative Hypothesis: At least one concentration has a significantly different mean response')
print('\nConclusion:')
if p_value < 0.05:
    print('Reject the null hypothesis (p < 0.05). There are significant differences between concentrations.'
else:
    print('Fail to reject the null hypothesis (p >= 0.05). No significant differences between concentrations
```



```
# Perform Tukey's post-hoc test if ANOVA shows significant differences
if p_value < 0.05:
    tukey = pairwise_tukeyhsd(df_melted['Response'], df_melted['Concentration'])
    print('\nTukey\'s Post-hoc Test Results:')
    print(tukey)

    # Visualize the results
    plt.figure(figsize=(10, 6))
    tukey.plot_simultaneous()
    plt.title("Tukey's HSD Test Results")
    plt.tight_layout()
    plt.show()
```

Practical Assignment 6

Create a database in SQLite “College” and create a table Student with five fields : Name, EnrolmentNo, Percentage, Course, Batch.

Create an interface in tkinter to insert new data in Student table, through Entry text, radio buttons and on click of “’Add” button, the record should be added to the table.

Create “Display” and “Display All”’ buttons, and on click of button- display the current record in above text fields and on click of “display all” button , show all the records in Listbox.

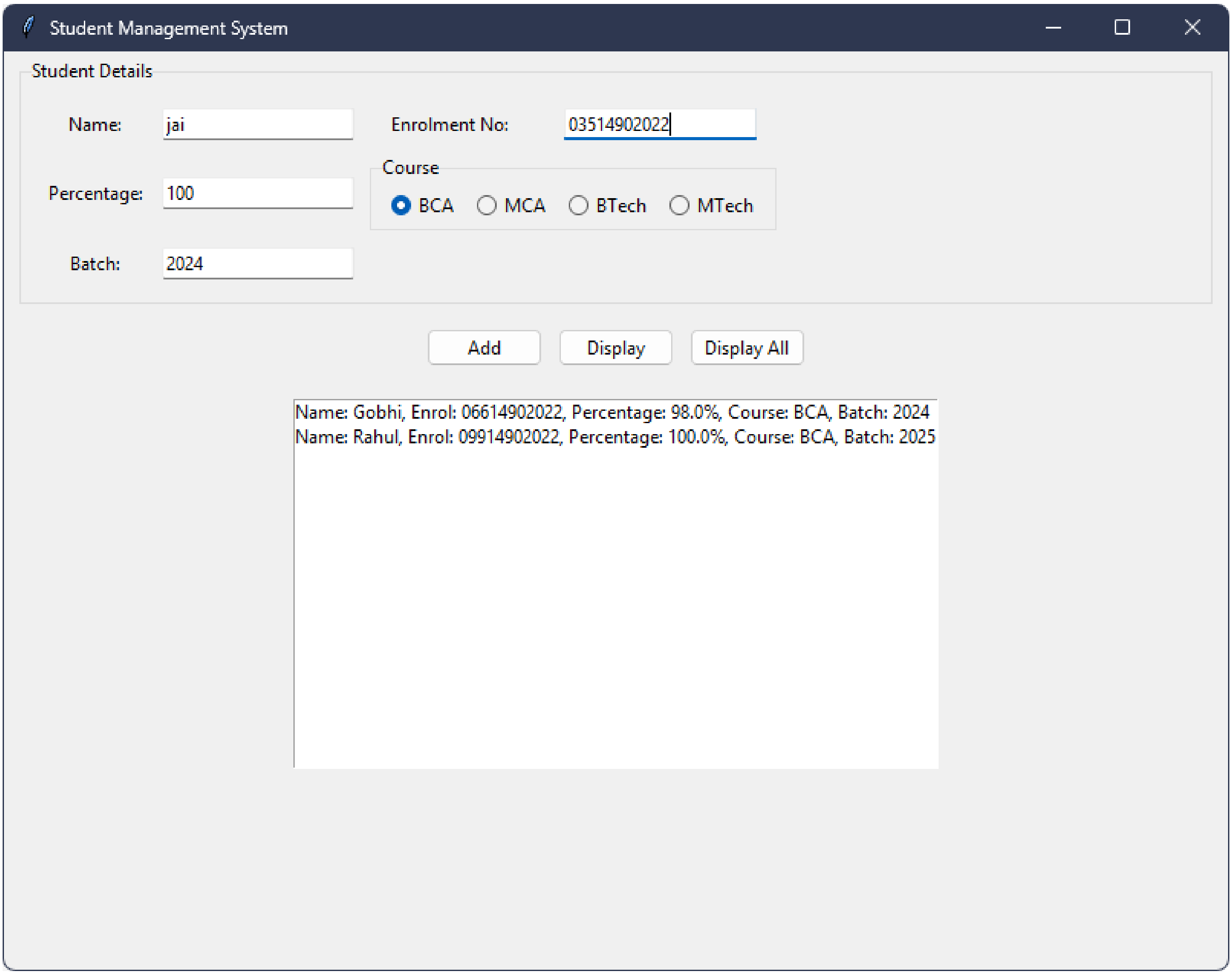
Code:

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
from typing import Optional

class StudentManagementSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Student Management System")
        self.root.geometry("800x600")

        # Create database and table
        self.create_database()

        # Variables for entry fields
        self.name_var = tk.StringVar()
        self.enrol_var = tk.StringVar()
        self.percentage_var = tk.StringVar()
        self.course_var = tk.StringVar()
        self.batch_var = tk.StringVar()

        self.create_widgets()

    def create_database(self):
        conn = sqlite3.connect('College.db')
        cursor = conn.cursor()

        # Create Student table if it doesn't exist
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS Student (
                Name TEXT,
                EnrolmentNo TEXT PRIMARY KEY,
                Percentage REAL,
                Course TEXT,
                Batch TEXT
            )
        ''')

        conn.commit()
        conn.close()

    def create_widgets(self):
        # Entry Fields Frame
        entry_frame = ttk.LabelFrame(self.root, text="Student Details", padding=10)
        entry_frame.pack(fill="x", padx=10, pady=5)

        # Name
        ttk.Label(entry_frame, text="Name:").grid(row=0, column=0, padx=5, pady=5)
        ttk.Entry(entry_frame, textvariable=self.name_var).grid(row=0, column=1, padx=5, pady=5)

        # Enrolment No
        ttk.Label(entry_frame, text="Enrolment No:").grid(row=0, column=2, padx=5, pady=5)
        ttk.Entry(entry_frame, textvariable=self.enrol_var).grid(row=0, column=3, padx=5, pady=5)

        # Percentage
        ttk.Label(entry_frame, text="Percentage:").grid(row=1, column=0, padx=5, pady=5)
        ttk.Entry(entry_frame, textvariable=self.percentage_var).grid(row=1, column=1, padx=5,
pady=5)

        # Course (Radio Buttons)
        course_frame = ttk.LabelFrame(entry_frame, text="Course", padding=5)
        course_frame.grid(row=1, column=2, columnspan=2, padx=5, pady=5)

        courses = ['BCA', 'MCA', 'BTech', 'MTech']
        self.course_var.set(courses[0])  # Default selection
        for i, course in enumerate(courses):
            ttk.Radiobutton(course_frame, text=course, variable=self.course_var,
                           value=course).grid(row=0, column=i, padx=5)

        # Batch
        ttk.Label(entry_frame, text="Batch:").grid(row=2, column=0, padx=5, pady=5)
        ttk.Entry(entry_frame, textvariable=self.batch_var).grid(row=2, column=1, padx=5, pady=5)
        # Buttons Frame
        btn_frame = ttk.Frame(self.root)
        btn_frame.pack(pady=10)

        ttk.Button(btn_frame, text="Add", command=self.add_record).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Display", command=self.display_record).pack(side=tk.LEFT,
padx=5)
        ttk.Button(btn_frame, text="Display All", command=self.display_all).pack(side=tk.LEFT, padx=5)

        # Listbox for displaying all records
        self.listbox = tk.Listbox(self.root, width=70, height=15)
        self.listbox.pack(padx=10, pady=10)

    def add_record(self):
        try:
            conn = sqlite3.connect('College.db')
            cursor = conn.cursor()

            # Validate input
            if not all([self.name_var.get(), self.enrol_var.get(),
                        self.percentage_var.get(), self.course_var.get(),
                        self.batch_var.get()]):
                messagebox.showerror("Error", "All fields are required!")
                return

            # Insert record
            cursor.execute('''
                INSERT INTO Student (Name, EnrolmentNo, Percentage, Course, Batch)
                VALUES (?, ?, ?, ?, ?)
            ''', (self.name_var.get(), self.enrol_var.get(),
                 float(self.percentage_var.get()), self.course_var.get(),
                 self.batch_var.get()))

            conn.commit()
            messagebox.showinfo("Success", "Record added successfully!")
            self.clear_fields()

        except sqlite3.IntegrityError:
            messagebox.showerror("Error", "Enrolment number already exists!")
        except ValueError:
            messagebox.showerror("Error", "Please enter valid percentage!")
        finally:
            conn.close()
```

```
def display_record(self):
    enrol = self.enrol_var.get()
    if not enrol:
        messagebox.showerror("Error", "Please enter Enrolment No to display!")
        return

    conn = sqlite3.connect('College.db')
    cursor = conn.cursor()

    cursor.execute('SELECT * FROM Student WHERE EnrolmentNo = ?', (enrol,))
    record = cursor.fetchone()

    if record:
        self.name_var.set(record[0])
        self.enrol_var.set(record[1])
        self.percentage_var.set(record[2])
        self.course_var.set(record[3])
        self.batch_var.set(record[4])
    else:
        messagebox.showinfo("Info", "No record found!")

    conn.close()

def display_all(self):
    conn = sqlite3.connect('College.db')
    cursor = conn.cursor()

    cursor.execute('SELECT * FROM Student')
    records = cursor.fetchall()

    self.listbox.delete(0, tk.END)
    for record in records:
        self.listbox.insert(tk.END, f"Name: {record[0]}, Enrol: {record[1]}, "
                          f"Percentage: {record[2]}%, Course: {record[3]}, "
                          f"Batch: {record[4]}")

    conn.close()

def clear_fields(self):
    self.name_var.set("")
    self.enrol_var.set("")
    self.percentage_var.set("")
    self.batch_var.set("")
```

```
if __name__ == "__main__":
    root = tk.Tk()
    app = StudentManagementSystem(root)
    root.mainloop()
```

OUTPUT:



Student Management System

Student Details

Name: jai

Enrolment No: 03514902022

Percentage: 100

Course
● BCA ○ MCA ○ BTech ○ MTech

Batch: 2024

```
Name: Gobhi, Enrol: 06614902022, Percentage: 98.0%, Course: BCA, Batch: 2024
Name: Rahul, Enrol: 09914902022, Percentage: 100.0%, Course: BCA, Batch: 2025
```