```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler #for feature scaling
from sklearn.model_selection import train_test_split #train-test

#for simple linear regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
```

data exprolartion

```python
df =
pd.read_csv('/kaggle/input/auto-scout-car-price/final_scout_not_dummy.
csv')
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (15915, 23)
```

```python
df.head()
```

```
  make_model body_type  price               vat       km  Type
Fuel  \
0    Audi A1    Sedans  15770    VAT deductible  56013.0  Used
Diesel
1    Audi A1    Sedans  14500  Price negotiable  80000.0  Used
Benzine
2    Audi A1    Sedans  14640    VAT deductible  83450.0  Used
Diesel
3    Audi A1    Sedans  14500    VAT deductible  73000.0  Used
Diesel
4    Audi A1    Sedans  16790    VAT deductible  16200.0  Used
Diesel

   Gears                               Comfort_Convenience  \
0    7.0  Air conditioning,Armrest,Automatic climate con...
1    7.0  Air conditioning,Automatic climate control,Hil...
2    7.0  Air conditioning,Cruise control,Electrical sid...
3    6.0  Air suspension,Armrest,Auxiliary heating,Elect...
4    7.0  Air conditioning,Armrest,Automatic climate con...

                                 Entertainment_Media  ...
Previous_Owners  \
0  Bluetooth,Hands-free equipment,On-board comput...  ...
2.0
```

```
1  Bluetooth,Hands-free equipment,On-board comput...  ...
1.0
2                            MP3,On-board computer  ...
1.0
3  Bluetooth,CD player,Hands-free equipment,MP3,O...  ...
1.0
4  Bluetooth,CD player,Hands-free equipment,MP3,O...  ...
1.0

   hp_kW  Inspection_new  Paint_Type  Upholstery_type  Gearing_Type  \
0   66.0               1    Metallic            Cloth     Automatic
1  141.0               0    Metallic            Cloth     Automatic
2   85.0               0    Metallic            Cloth     Automatic
3   66.0               0    Metallic            Cloth     Automatic
4   66.0               1    Metallic            Cloth     Automatic

  Displacement_cc Weight_kg Drive_chain  cons_comb
0          1422.0    1220.0       front        3.8
1          1798.0    1255.0       front        5.6
2          1598.0    1135.0       front        3.8
3          1422.0    1195.0       front        3.8
4          1422.0    1135.0       front        4.1

[5 rows x 23 columns]
```

```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15915 entries, 0 to 15914
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   make_model            15915 non-null  object
 1   body_type             15915 non-null  object
 2   price                 15915 non-null  int64
 3   vat                   15915 non-null  object
 4   km                    15915 non-null  float64
 5   Type                  15915 non-null  object
 6   Fuel                  15915 non-null  object
 7   Gears                 15915 non-null  float64
 8   Comfort_Convenience   15915 non-null  object
 9   Entertainment_Media   15915 non-null  object
 10  Extras                15915 non-null  object
 11  Safety_Security       15915 non-null  object
 12  age                   15915 non-null  float64
 13  Previous_Owners       15915 non-null  float64
 14  hp_kW                 15915 non-null  float64
 15  Inspection_new        15915 non-null  int64
 16  Paint_Type            15915 non-null  object
```

```
 17  Upholstery_type      15915 non-null  object 
 18  Gearing_Type         15915 non-null  object 
 19  Displacement_cc      15915 non-null  float64
 20  Weight_kg            15915 non-null  float64
 21  Drive_chain          15915 non-null  object 
 22  cons_comb            15915 non-null  float64
dtypes: float64(8), int64(2), object(13)
memory usage: 2.8+ MB

              price             km         Gears           age  \
count  15915.000000   15915.000000  15915.000000  15915.000000   
mean   18024.380584   32089.995708      5.937355      1.389695   
std     7381.679318   36977.214964      0.704772      1.121306   
min     4950.000000       0.000000      5.000000      0.000000   
25%    12850.000000    1920.500000      5.000000      0.000000   
50%    16900.000000   20413.000000      6.000000      1.000000   
75%    21900.000000   46900.000000      6.000000      2.000000   
max    74600.000000  317000.000000      8.000000      3.000000   

       Previous_Owners         hp_kW  Inspection_new  Displacement_cc
\
count     15915.000000  15915.000000    15915.000000     15915.000000

mean          1.042853     88.499340        0.247063      1428.661891

std           0.339178     26.674341        0.431317       275.804272

min           0.000000     40.000000        0.000000       890.000000

25%           1.000000     66.000000        0.000000      1229.000000

50%           1.000000     85.000000        0.000000      1461.000000

75%           1.000000    103.000000        0.000000      1598.000000

max           4.000000    294.000000        1.000000      2967.000000


          Weight_kg     cons_comb  
count  15915.000000  15915.000000  
mean    1337.700534      4.832124  
std      199.682385      0.867530  
min      840.000000      3.000000  
25%     1165.000000      4.100000  
50%     1295.000000      4.800000  
75%     1472.000000      5.400000  
max     2471.000000      9.100000  
```

data cleaning

```
df.isnull().sum()
```

```
make_model              0
body_type               0
price                   0
vat                     0
km                      0
Type                    0
Fuel                    0
Gears                   0
Comfort_Convenience     0
Entertainment_Media     0
Extras                  0
Safety_Security         0
age                     0
Previous_Owners         0
hp_kW                   0
Inspection_new          0
Paint_Type              0
Upholstery_type         0
Gearing_Type            0
Displacement_cc         0
Weight_kg               0
Drive_chain             0
cons_comb               0
dtype: int64
```

```
df.isna().sum()
```

```
make_model              0
body_type               0
price                   0
vat                     0
km                      0
Type                    0
Fuel                    0
Gears                   0
Comfort_Convenience     0
Entertainment_Media     0
Extras                  0
Safety_Security         0
age                     0
Previous_Owners         0
hp_kW                   0
Inspection_new          0
Paint_Type              0
Upholstery_type         0
Gearing_Type            0
```

```
Displacement_cc        0
Weight_kg              0
Drive_chain            0
cons_comb              0
dtype: int64
```

```
df.duplicated().sum()
```

```
1673
```

```
cleaned_df=df.drop_duplicates()
cleaned_df.duplicated().sum()
```

```
0
```

```
#need only the numerical values
cleaned_df=df.select_dtypes(include=[np.number])
cleaned_df.info()
```

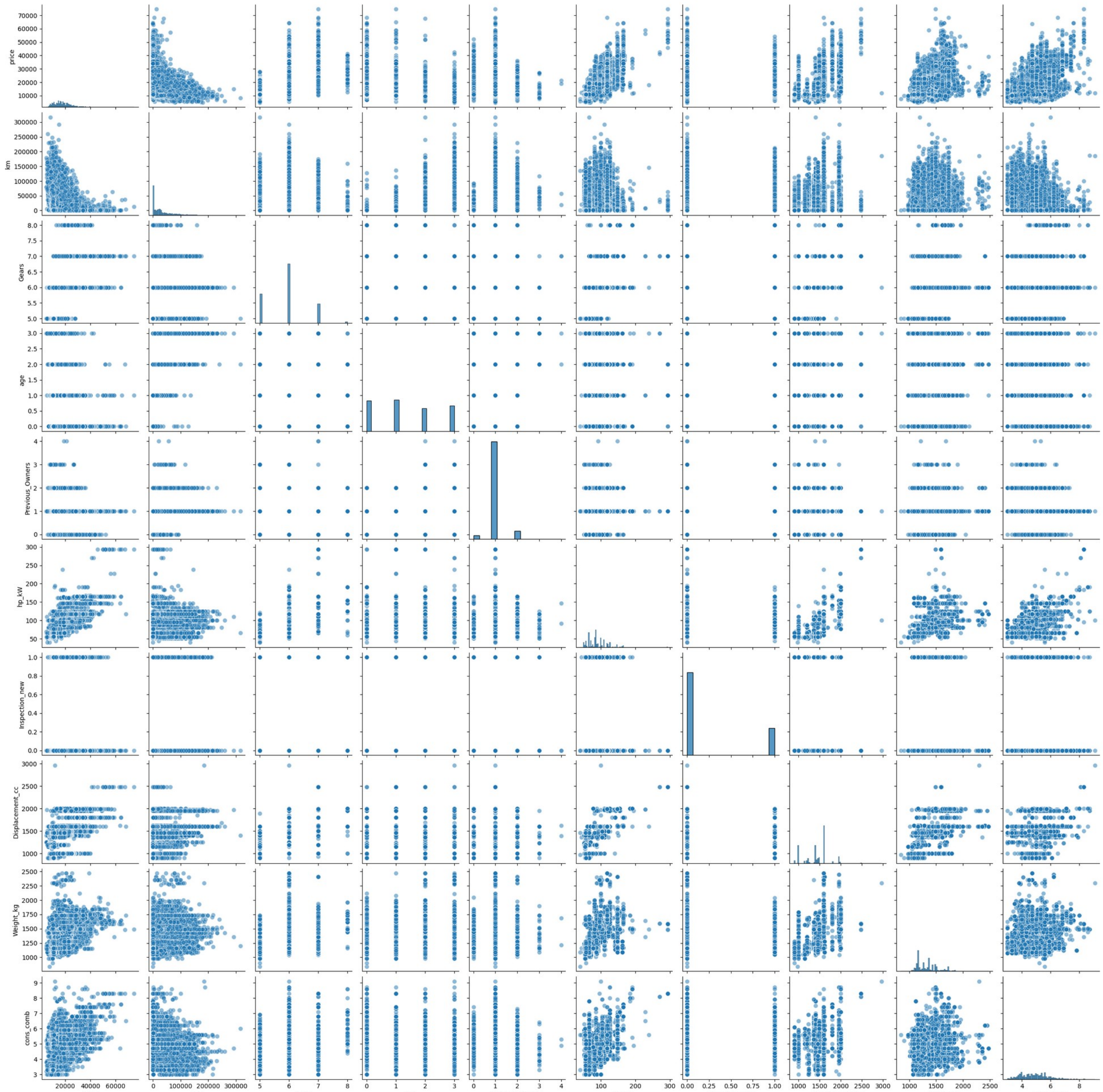
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15915 entries, 0 to 15914
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   price            15915 non-null  int64  
 1   km               15915 non-null  float64
 2   Gears            15915 non-null  float64
 3   age              15915 non-null  float64
 4   Previous_Owners  15915 non-null  float64
 5   hp_kW            15915 non-null  float64
 6   Inspection_new   15915 non-null  int64  
 7   Displacement_cc  15915 non-null  float64
 8   Weight_kg        15915 non-null  float64
 9   cons_comb        15915 non-null  float64
dtypes: float64(8), int64(2)
memory usage: 1.2 MB
```

data visualization

```
# Now create the pairplot
sns.pairplot(cleaned_df, kind='scatter', plot_kws={'s': 50, 'alpha':
0.5})
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
```

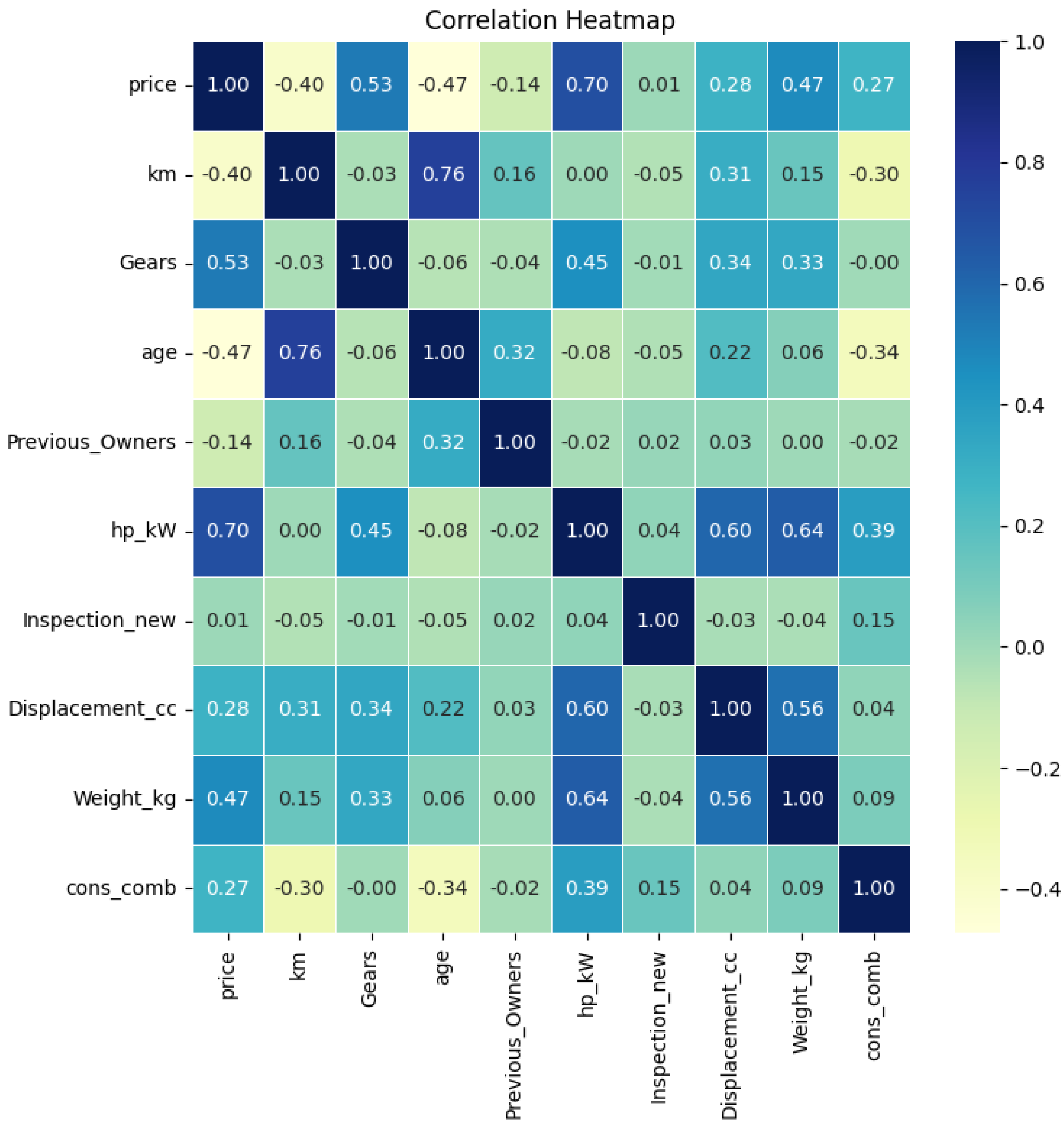
```
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):

<seaborn.axisgrid.PairGrid at 0x79d6a7f56f20>
```



we can observe that: 'km' will be good at polynomial regresstion and 'hp_kW', 'age', 'Weight_kg' have strong correlation will use at linear regression

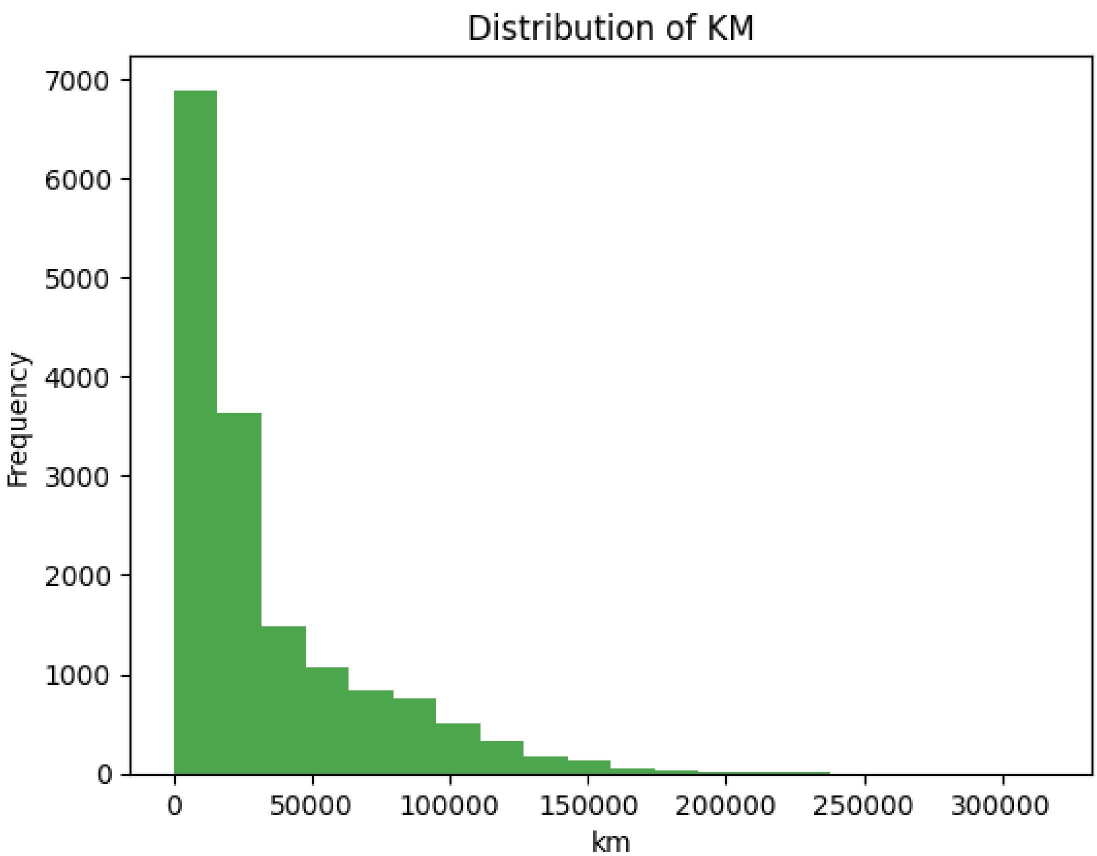
```
# Calculate the correlation matrix
correlation_matrix = cleaned_df.corr()

# Create the heatmap
plt.figure(figsize=(8, 8))  # Set the size of the plot
sns.heatmap(correlation_matrix, annot=True, cmap='YlGnBu', fmt='.2f',
linewidths=0.5)

# Show the plot
```

```
plt.title('Correlation Heatmap')
plt.show()
```




```
#best predictors for price in order
print(cleaned_df.corr().abs().nlargest(10,'price').index)

Index(['price', 'hp_kW', 'Gears', 'age', 'Weight_kg', 'km',
'Displacement_cc',
       'cons_comb', 'Previous_Owners', 'Inspection_new'],
      dtype='object')
```

```
import matplotlib.pyplot as plt

plt.hist(cleaned_df['km'], bins=20, color='green', alpha=0.7)
plt.xlabel('km')
plt.ylabel('Frequency')
plt.title('Distribution of KM')
plt.show()
```



linear regression

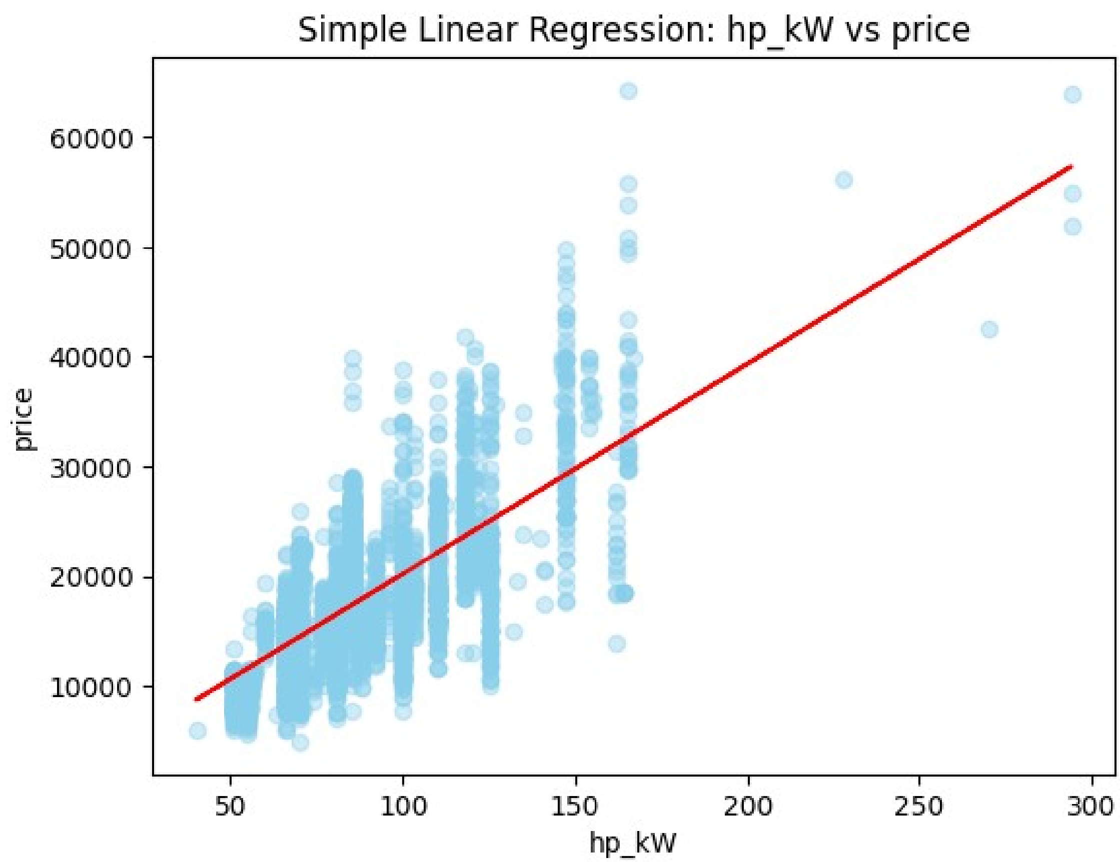
```
features = ['km', 'hp_kW', 'Displacement_cc', 'Weight_kg',
'Previous_Owners', 'cons_comb', 'Gears', 'age']
target = 'price'

X = cleaned_df[features]
y = cleaned_df[target]

# Splitting dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

simple linear regression

```
y =cleaned_df['price']
X_simple = cleaned_df[['hp_kW']]  # Best-correlated feature with price

X_train_hp, X_test_hp, y_train_hp, y_test_hp = 
train_test_split(X_simple, y, test_size=0.2, random_state=42)

model_simple = LinearRegression()
model_simple.fit(X_train_hp, y_train_hp)
y_pred_s = model_simple.predict(X_test_hp)

r2 = r2_score(y_test_hp, y_pred_s)
mse = mean_squared_error(y_test_hp, y_pred_s)
print("Simple Linear Regression:")
print("R^2 Score:", r2)
print("MSE:", mse)

# Visualization
plt.scatter(X_test_hp, y_test_hp, color='skyblue', alpha=0.4)
plt.plot(X_test_hp, y_pred_s, color='red')
plt.xlabel('hp_kW')
plt.ylabel('price')
plt.title('Simple Linear Regression: hp_kW vs price')
plt.show()
```

```
Simple Linear Regression:
R^2 Score: 0.5122324660161587
MSE: 26323633.657338135
```




```
# The coefficients in a dataframe
cdf =
pd.DataFrame(model_simple.coef_,X_simple.columns,columns=['Coef'])
cdf

             Coef
hp_kW  191.32341
```

mulltiple features linear regression

```
X_multi = cleaned_df[features]

X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_multi,
y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_m_scaled = scaler.fit_transform(X_train_m)
X_test_m_scaled = scaler.transform(X_test_m)

model_multi = LinearRegression()
model_multi.fit(X_train_m_scaled, y_train_m)
y_pred_m = model_multi.predict(X_test_m_scaled)
```

```
r2_m = r2_score(y_test_m, y_pred_m)
mse_m = mean_squared_error(y_test_m, y_pred_m)
print("Multiple Linear Regression:")
print("R^2 Score:", r2_m)
print("MSE:", mse_m)
```

```
Multiple Linear Regression:
R^2 Score: 0.757866741684256
MSE: 13067346.11076517
```

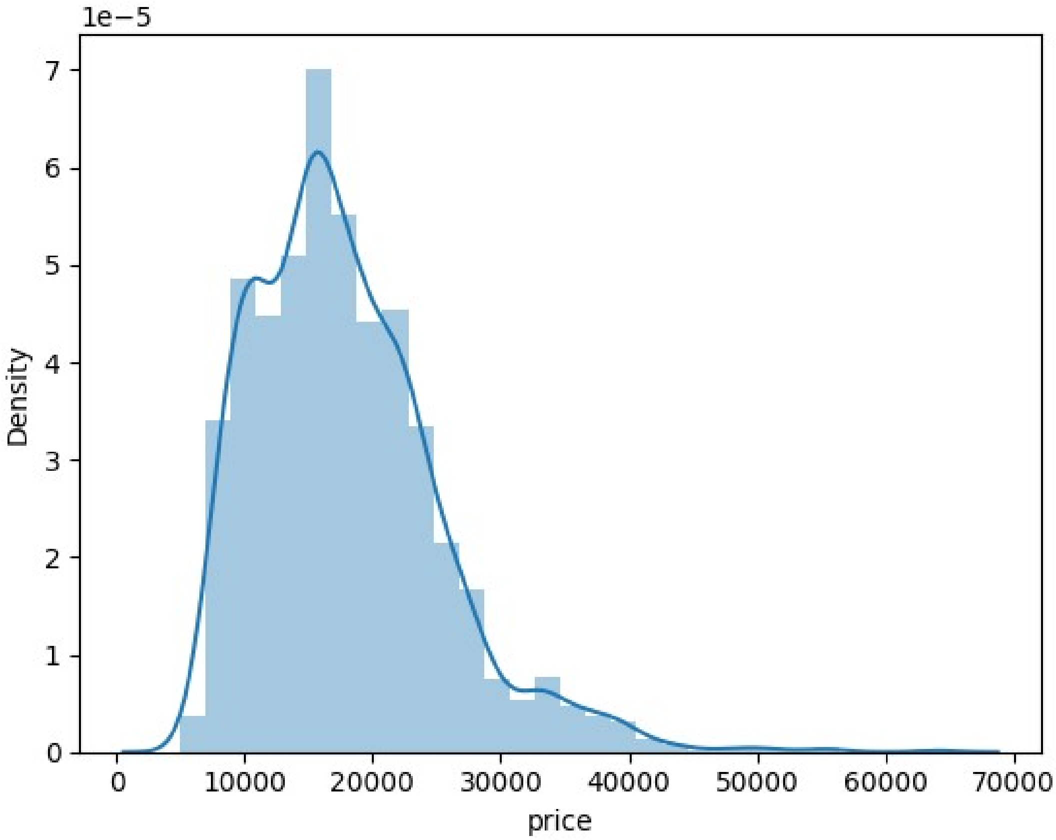
```
# The coefficients in a dataframe
cdf = pd.DataFrame(model_multi.coef_,X_multi.columns,columns=['Coef'])
cdf
```

	Coef
km	-1535.268368
hp_kW	4391.514606
Displacement_cc	-685.228336
Weight_kg	912.094101
Previous_Owners	-13.227102
cons_comb	-938.892372
Gears	1684.476421
age	-2115.331738

```
#should be normal
residuals = y_test_m
sns.distplot(residuals, bins=30)
```

```
<ipython-input-66-3eb82e56d15b>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(residuals, bins=30)
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
<Axes: xlabel='price', ylabel='Density'>
```



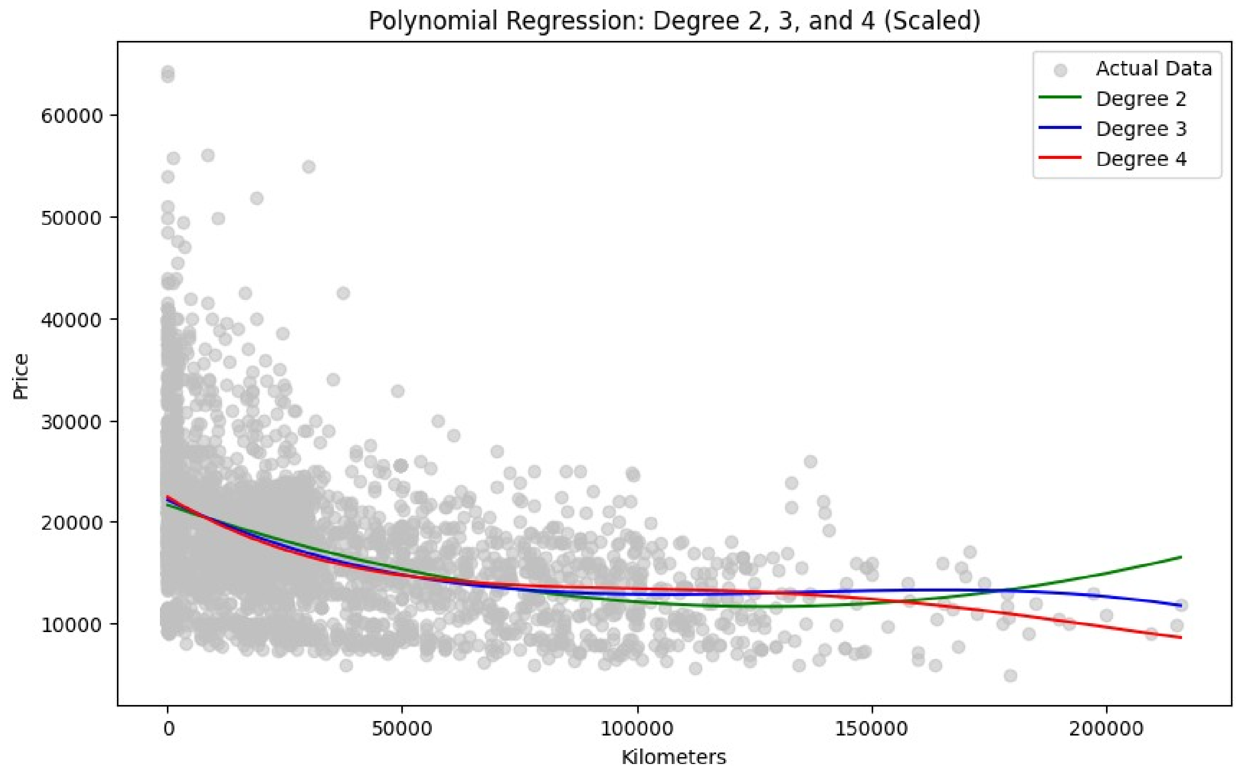

```
comparison_df = pd.DataFrame({
    "Model": ["Simple Linear Regression", "Multiple Linear
Regression"],
    "R² Score": [r2, r2_m],
    "MSE": [mse, mse_m]
})

comparison_df
```

	Model	R² Score	MSE
0	Simple Linear Regression	0.512232	2.632363e+07
1	Multiple Linear Regression	0.757867	1.306735e+07

polynomial regression

```
# Define X and y
y = cleaned_df['price']
X = cleaned_df[['km']]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42)

# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

degrees = [2, 3, 4]
colors = ['green', 'blue', 'red']
mse_values = []

plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='Silver', label='Actual Data',
alpha=0.6)

# Loop over each degree
for i, degree in enumerate(degrees):
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train_scaled)
    X_test_poly = poly.transform(X_test_scaled)

    model = LinearRegression()
    model.fit(X_train_poly, y_train)
    y_pred = model.predict(X_test_poly)

    mse = mean_squared_error(y_test, y_pred)
    mse_values.append((degree, mse))

    # Sorting for a smoother line plot
    sort_idx = X_test.values.flatten().argsort()
    X_sorted = X_test.values.flatten()[sort_idx]
    y_sorted = y_pred[sort_idx]

    plt.plot(X_sorted, y_sorted, color=colors[i], label=f'Degree
{degree}')

plt.xlabel('Kilometers')
plt.ylabel('Price')
plt.title('Polynomial Regression: Degree 2, 3, and 4 (Scaled)')
plt.legend()
plt.show()

# Print MSE values
for degree, mse in mse_values:
    print(f"Degree {degree}: MSE = {mse:.2f}")
```




```
Degree 2: MSE = 44508803.45
Degree 3: MSE = 44240410.67
Degree 4: MSE = 44193713.71
```

logistic regression

load the data

```
data =
pd.read_csv("/kaggle/input/breast-cancer-wisconsin-data/data.csv")

data.head()

/usr/local/lib/python3.10/dist-packages/pandas/io/formats/
format.py:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:14
59: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals >
0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:14
59: RuntimeWarning: invalid value encountered in greater
```

```
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals >
0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:14
58: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:14
59: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals >
0)).any()
/usr/local/lib/python3.10/dist-packages/pandas/io/formats/format.py:14
59: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals >
0)).any()

         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302         M        17.99         10.38          122.80
1001.0
1    842517         M        20.57         17.77          132.90
1326.0
2  84300903         M        19.69         21.25          130.00
1203.0
3  84348301         M        11.42         20.38           77.58
386.1
4  84358402         M        20.29         14.34          135.10
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   ...  texture_worst  perimeter_worst  area_worst
smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622

1  ...          23.41           158.80      1956.0            0.1238

2  ...          25.53           152.50      1709.0            0.1444

3  ...          26.50            98.87       567.7            0.2098
```

```
4  ...          16.67           152.20     1575.0            0.1374


   compactness_worst  concavity_worst  concave points_worst
symmetry_worst  \
0             0.6656           0.7119                0.2654
0.4601
1             0.1866           0.2416                0.1860
0.2750
2             0.4245           0.4504                0.2430
0.3613
3             0.8663           0.6869                0.2575
0.6638
4             0.2050           0.4000                0.1625
0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

```
data.info()
```

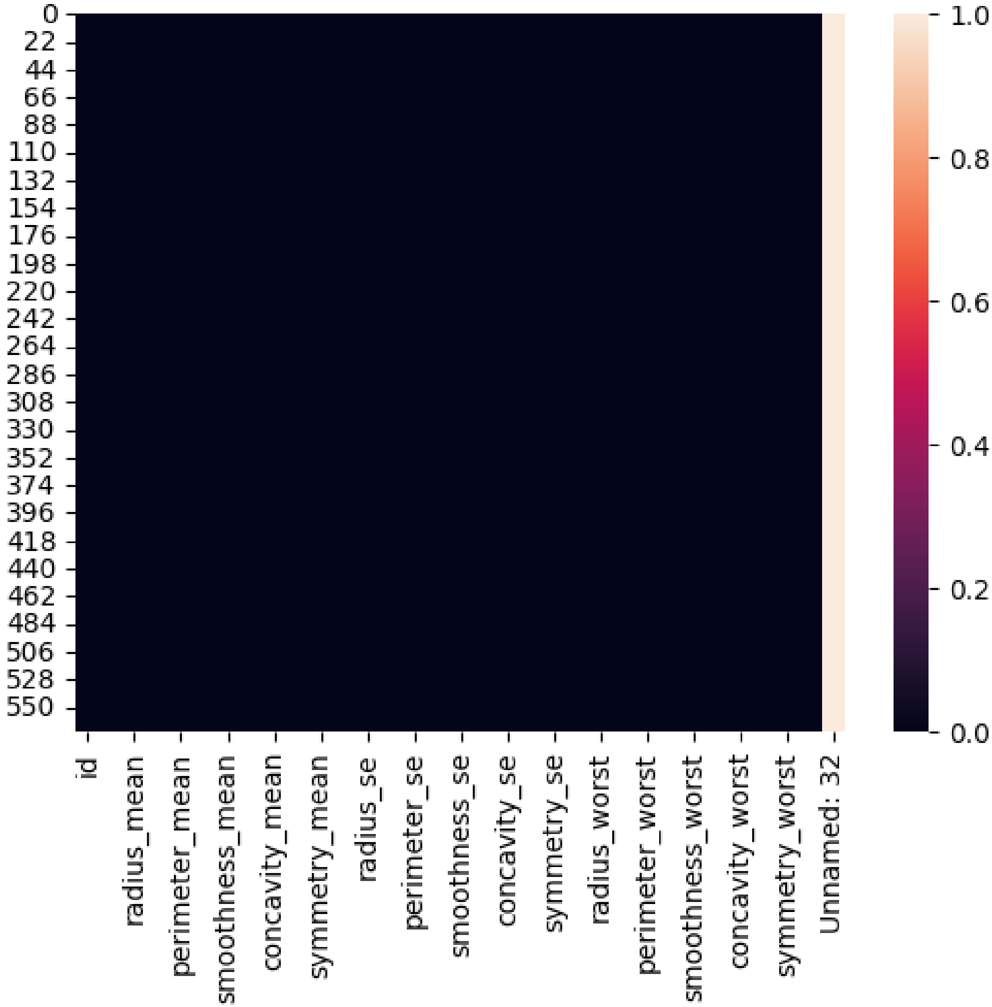
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
```

```
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

data cleaning

```
sns.heatmap(data.isnull())
```

```
<Axes: >
```




```
data.drop(["Unnamed: 32", "id"],axis = 1, inplace = True)
data.head()

  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
```

```
0.10520
4          0.10030          0.13280          0.1980
0.10430

   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33           184.60
1         0.1812  ...         24.99          23.41           158.80
2         0.2069  ...         23.57          25.53           152.50
3         0.2597  ...         14.91          26.50            98.87
4         0.1809  ...         22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 31 columns]
```

```python
data.diagnosis = [1 if value == "M" else 0 for value in
data.diagnosis]
data.head()
```

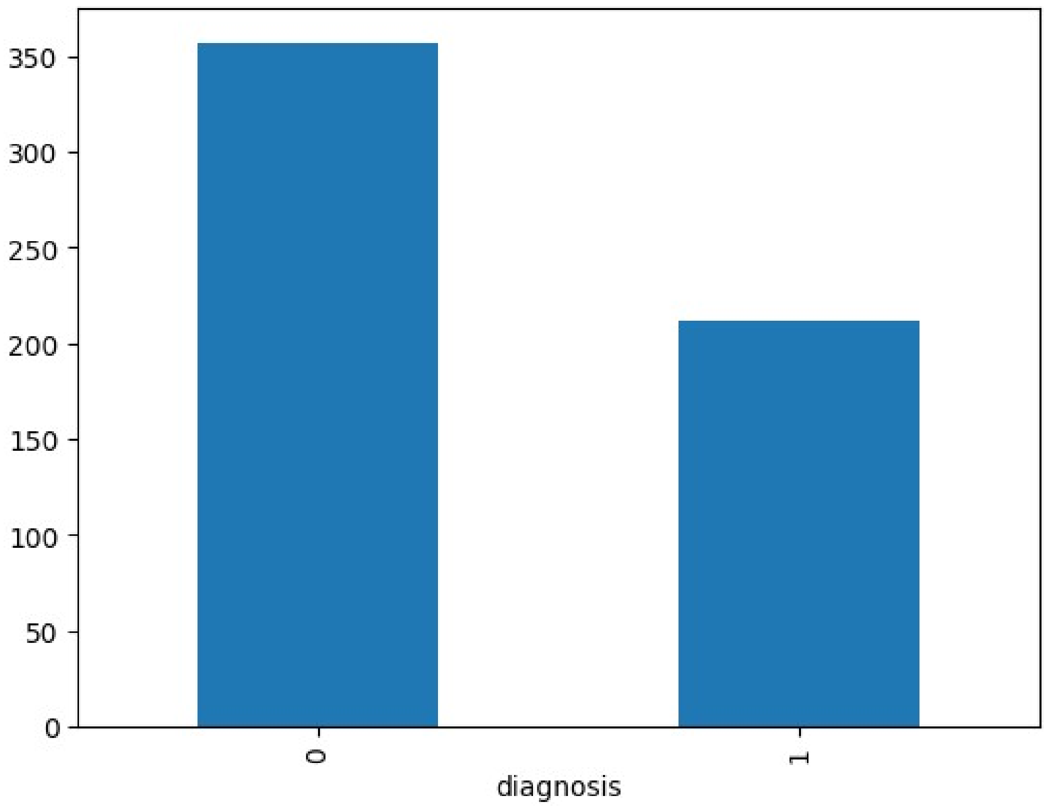
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
3          1        11.42         20.38           77.58      386.1
4          1        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840          0.27760          0.3001
0.14710
1          0.08474          0.07864          0.0869
0.07017
2          0.10960          0.15990          0.1974
0.12790
3          0.14250          0.28390          0.2414
0.10520
4          0.10030          0.13280          0.1980
0.10430
```

```
   symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0         0.2419  ...         25.38          17.33           184.60   
1         0.1812  ...         24.99          23.41           158.80   
2         0.2069  ...         23.57          25.53           152.50   
3         0.2597  ...         14.91          26.50            98.87   
4         0.1809  ...         22.54          16.67           152.20   

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119   
1      1956.0            0.1238             0.1866           0.2416   
2      1709.0            0.1444             0.4245           0.4504   
3       567.7            0.2098             0.8663           0.6869   
4      1575.0            0.1374             0.2050           0.4000   

   concave points_worst  symmetry_worst  fractal_dimension_worst  
0                0.2654          0.4601                  0.11890  
1                0.1860          0.2750                  0.08902  
2                0.2430          0.3613                  0.08758  
3                0.2575          0.6638                  0.17300  
4                0.1625          0.2364                  0.07678  

[5 rows x 31 columns]
```

data visualization

```
data["diagnosis"] = data['diagnosis'].astype("category", copy = False)
data["diagnosis"].value_counts().plot(kind = "bar")
```

```
<Axes: xlabel='diagnosis'>
```




```
y = data["diagnosis"]
X = data.drop(["diagnosis"], axis = 1)
y
```

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: category
Categories (2, int64): [0, 1]
```

scaling

```
scaler = StandardScaler()
#fit
```

```
X_scaled = scaler.fit_transform(X)
X_scaled
```

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_scaled, y,
test_size=0.30,random_state = 42)
```

train

```
from sklearn.linear_model import LogisticRegression
#Instantiate
lr = LogisticRegression()
#Train
lr.fit(X_train,y_train)
#Predict
y_pred = lr.predict(X_test)
print(f"Logistic Regression-Training set score: {lr.score(X_train,
y_train):.2f}")
print(f"Logistic Regression-Test set score: {lr.score(X_test,
y_test):.2f}")
```

```
Logistic Regression-Training set score: 0.99
Logistic Regression-Test set score: 0.98
```

Confusion matrix

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred)
print(f"Accuracy: {accuracy: 2f}")
```

```
Accuracy:  0.982456
```

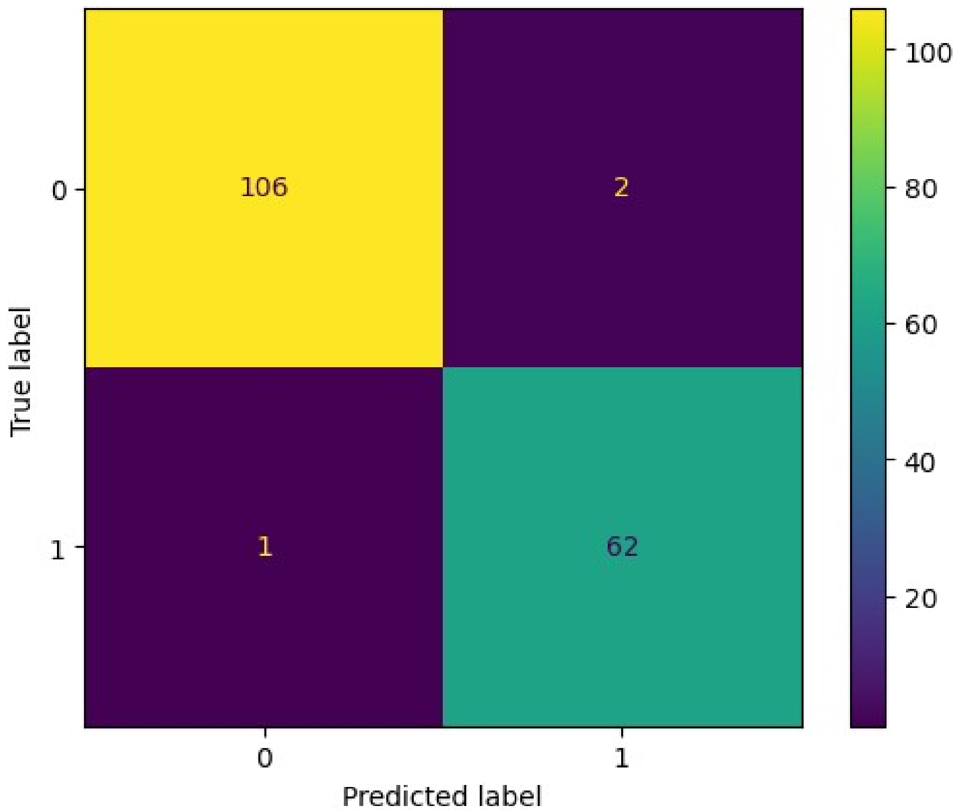
```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       107
           1       0.98      0.97      0.98        64

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

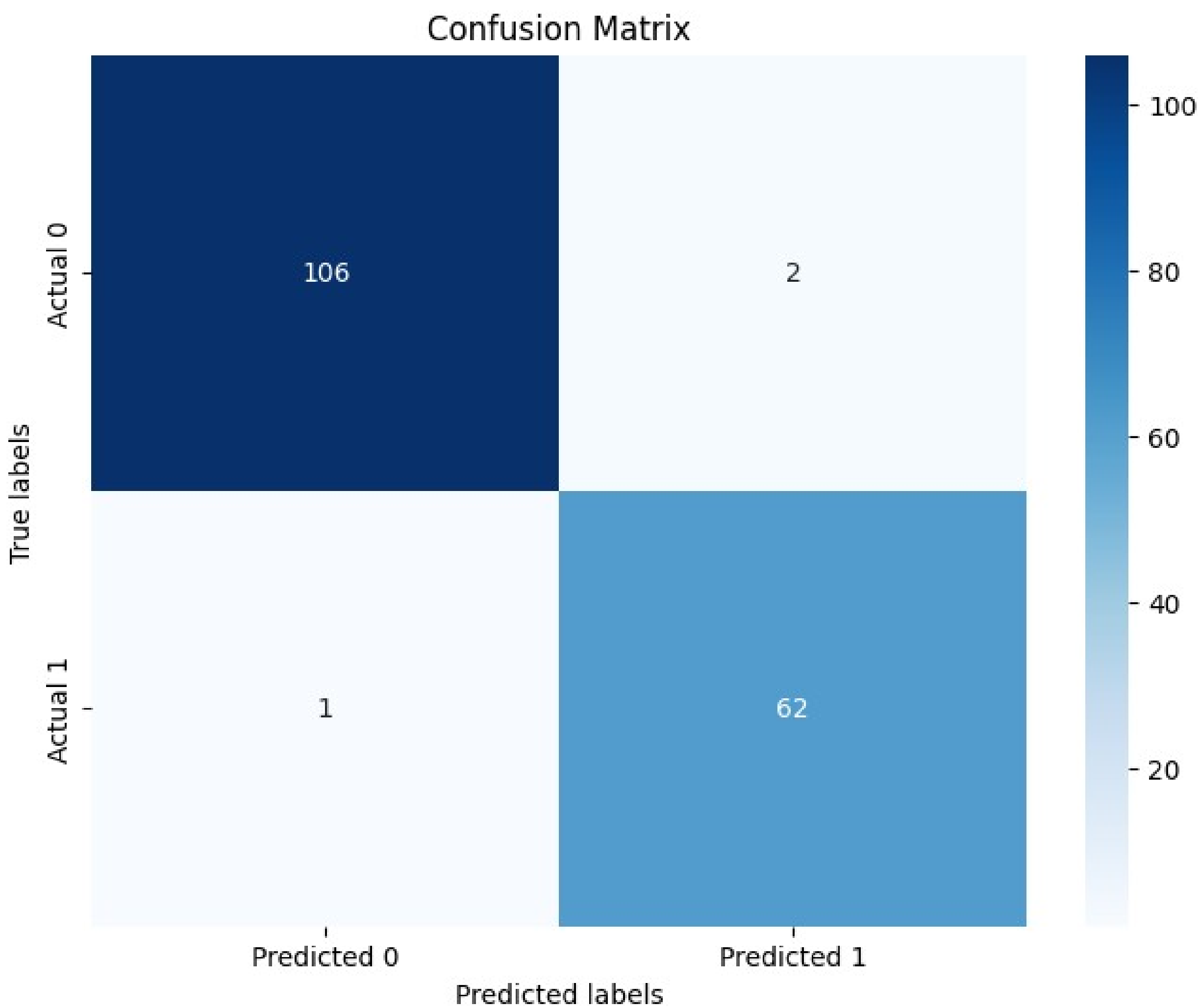
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

```
array([[106,   2],
       [  1,  62]])
```

```
from sklearn.metrics import ConfusionMatrixDisplay

disp =
ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test,
y_pred))
disp.plot()
plt.show()
```




```
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues',
xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0',
'Actual 1'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test,y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.98
```

```
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       107
           1       0.98      0.97      0.98        64

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```