```
'''
Iris flower has three species; setosa, versicolor, and virginica, which differs according to their
measurements. Now assume that you have the measurements of the iris flowers according to
their species, and here your task is to train a machine learning model that can learn from the
measurements of the iris species and classify them
Although the Scikit-learn library provides a dataset for iris flower classification, you can also
download the same dataset from here for the task of iris flower classification with Machine
Learning.
'''
```

```
'\nIris flower has three species; setosa, versicolor, and virginica, which differs according to their\nmeasurements. Now assume tha
t you have the measurements of the iris flowers according to\ntheir species, and here your task is to train a machine learning mode
l that can learn from the\nmeasurements of the iris species and classify them\nAlthough the Scikit-learn library provides a dataset
for iris flower classification, you can also\ndownload the same dataset from here for the task of iris flower classification with M
achine\nLearning.\n'
```

```
import pandas as pd
import numpy as np
import seaborn as sns
```

```
df=sns.load_dataset('iris')
```

```
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df['species'].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

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

	0
sepal_length	0
sepal_width	0
petal_length	0
petal_width	0
species	0

dtype: int64

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

```
np.int64(1)
```

```
df.shape
```

```
(150, 5)
```

```
df.drop_duplicates(inplace=True)
```

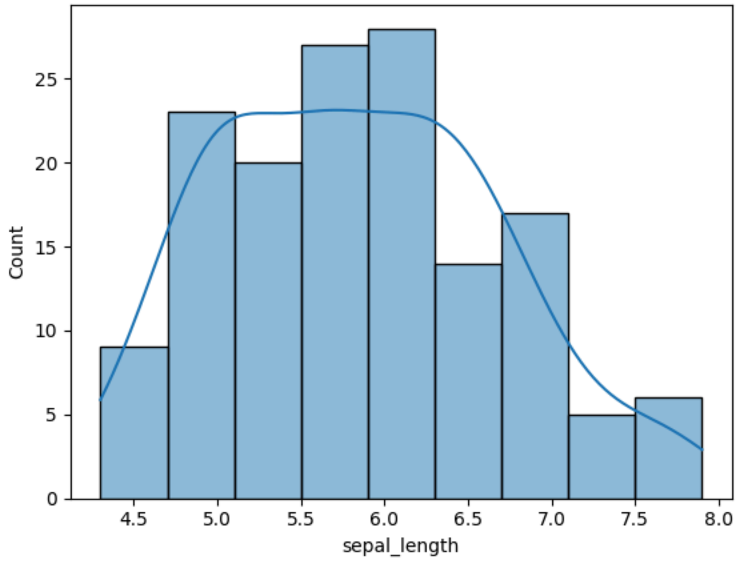
```
df.shape
```

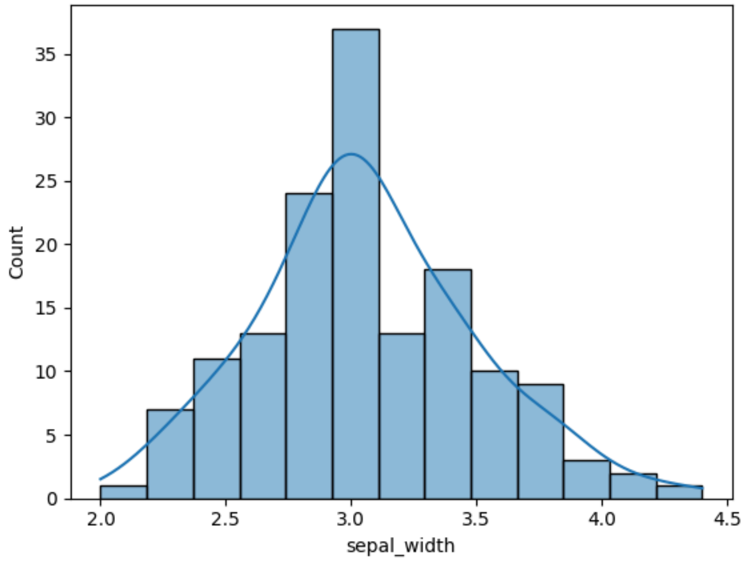
```
(149, 5)
```

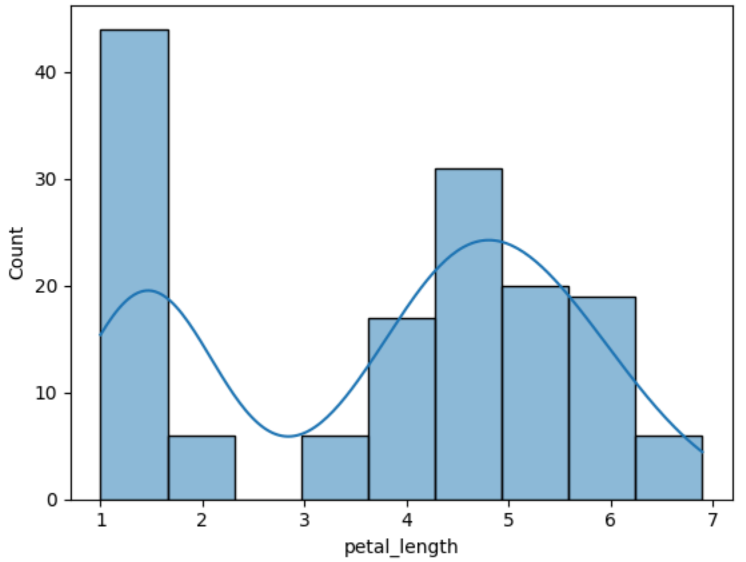
```
import matplotlib.pyplot as plt
```

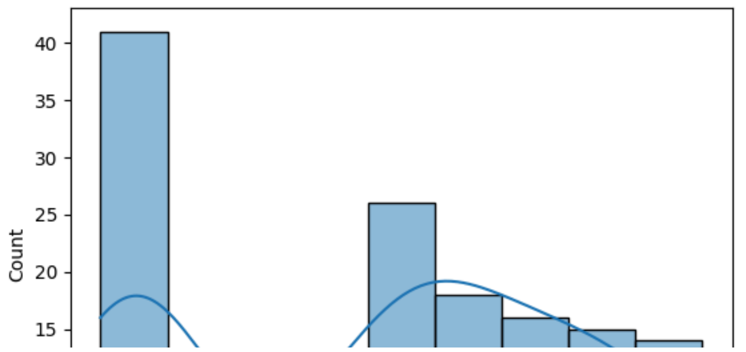
```
for i in df.columns:
    if df[i].dtypes=='object':
```

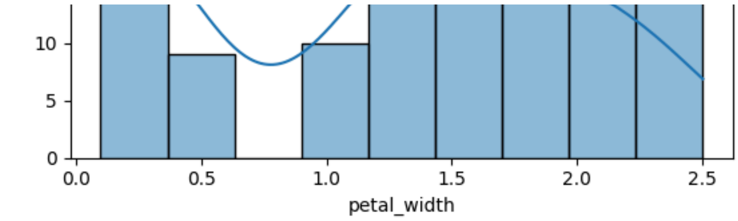
```
    continue
  sns.histplot(data=df,x=i,kde=True)
  plt.show()
```


Count
0
5
10
15
20
25
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
sepal_length


Count
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5
sepal_width


Count
0
10
20
30
40
1
2
3
4
5
6
7
petal_length


Count
15
20
25
30
35
40




```
df.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

```
df['petal_length']=np.log(df['petal_length'])
```

```
df['petal_width']=np.sqrt(df['petal_width'])
```

```
df['sepal_length']=np.log(df['sepal_length'])
```

```
np.mean(np.log(df['petal_width']))
```

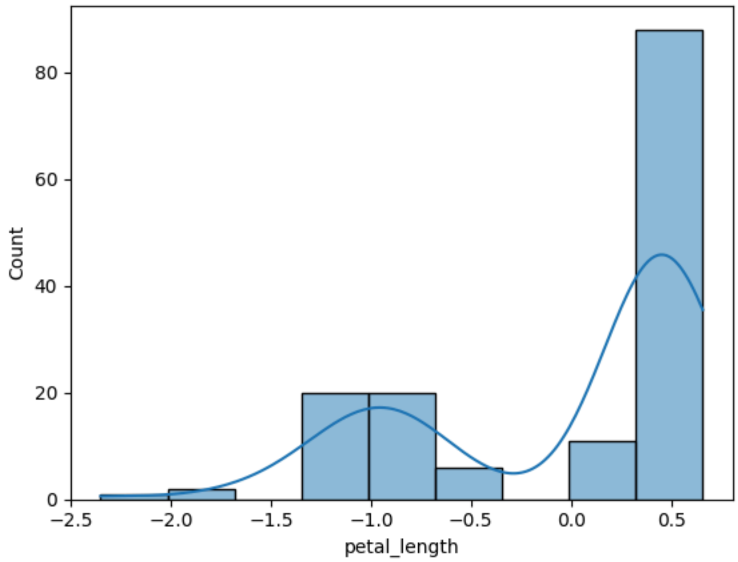
```
np.float64(-0.08889346431188545)
```

```
np.mean(np.sqrt(df['petal_width']))
```

```
np.float64(0.983830888077141)
```

```
sns.histplot(data=df,x=np.log(df['petal_length']),kde=True)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/pandas/core/arraylike.py:399: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



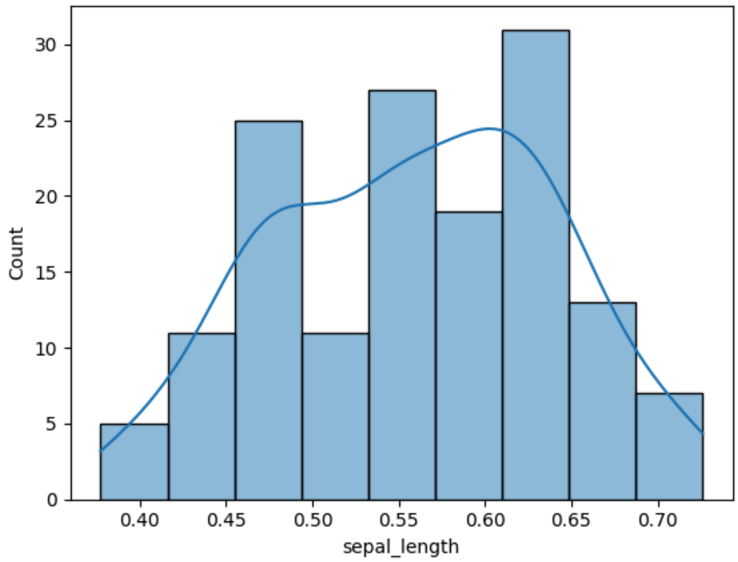

```
np.mean(np.sqrt(df['petal_length']))
```

```
np.float64(1.0333140332764557)
```

```
np.mean(np.log(df['petal_length']))
```

```
/usr/local/lib/python3.11/dist-packages/pandas/core/arraylike.py:399: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
np.float64(-inf)
```

```
sns.histplot(data=df,x=np.log(df['sepal_length']),kde=True)
plt.show()
```




```
np.mean(np.sqrt(df['sepal_length']))
```

```
np.float64(1.323831781280919)
```

```
np.mean(np.log(df['sepal_width']))
```

```
np.float64(1.1082055251621128)
```

```
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	1.629241	3.5	0.336472	0.447214	setosa
1	1.589235	3.0	0.336472	0.447214	setosa
2	1.547563	3.2	0.262364	0.447214	setosa
3	1.526056	3.1	0.405465	0.447214	setosa
4	1.609438	3.6	0.336472	0.447214	setosa

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.shape
```

```
(149, 5)
```

Start coding or generate with AI.

```
df['species'].value_counts()
```

species	count
setosa	50
versicolor	50
virginica	49

dtype: int64

MODEL BUILDING

```
X=df.drop('species',axis=1)
```

```
y=df['species']
```

```
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
```

```
df['species']=lb.fit_transform(df['species'])
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,Y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
from sklearn.model_selection import GridSearchCV
```

```
para={
    'C':[0.01,0.1,1],
    'solver':['liblinear','saga','newton-cg'],
    'penalty':['l1','l2']
}
```

```
from sklearn.linear_model import LogisticRegression
lg=LogisticRegression(C=1,penalty='l1',solver='saga')
lg.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means t
  warnings.warn(
```

```
LogisticRegression
LogisticRegression(C=1, penalty='l1', solver='saga')
```

```
gd=GridSearchCV(estimator=lg,param_grid=para,cv=5,scoring='accuracy')
gd.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528: FitFailedWarning:
15 fits failed out of a total of 90.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
15 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py", line 1193, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py", line 63, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or None penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are r
 0.66376812 0.6807971         nan 0.68043478 0.86594203 0.86594203
 0.89166667 0.95797101        nan 0.925      0.91630435 0.9076087 ]
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means
  warnings.warn(
```

▸ **GridSearchCV**

▸ **best_estimator_: LogisticRegression**

▸ LogisticRegression

```
gd.best_score_
```

```
np.float64(0.9579710144927537)
```

```
y_pred=lg.predict(X_test)
```

```
from sklearn.metrics import *
```

```
accuracy_score(Y_test,y_pred)
```

```
1.0
```

```
y_prob=lg.predict_proba(X_test)
```

```
auc=roc_auc_score(Y_test,y_prob,multi_class='ovr')
```

```
auc
```

```
np.float64(1.0)
```

```
confusion_matrix(Y_test,y_pred)
```

```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  0, 11]])
```

```
print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

UNEMPLOYMENT IN INDIA AT THE TIME OF CORONA

```
df1=pd.read_csv(r'/content/Unemployment in India.csv')
```

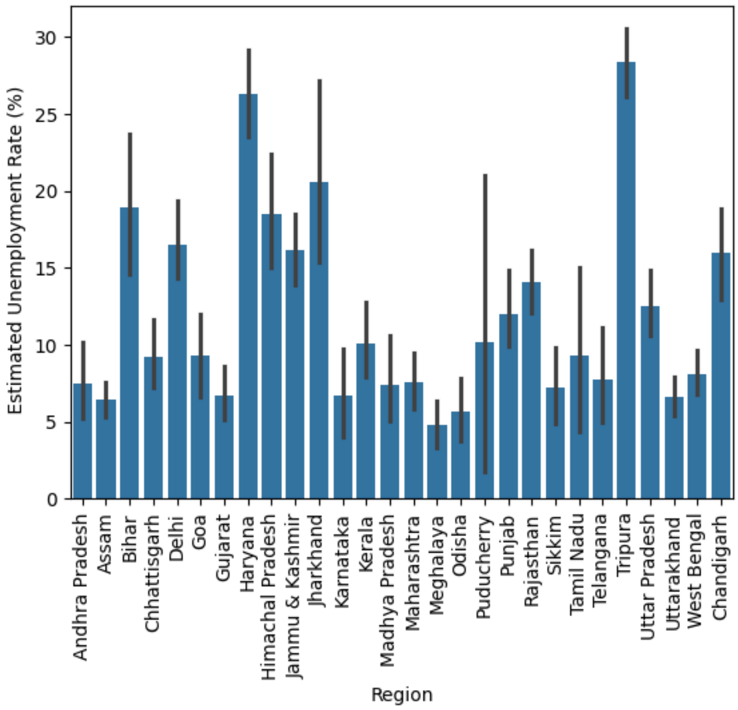
```
df1.head()
```

	Region	Date	Frequency	Estimated Unemployment Rate (%)	Estimated Employed	Estimated Labour Participation Rate (%)	Area
0	Andhra Pradesh	31-05-2019	Monthly	3.65	11999139.0	43.24	Rural
1	Andhra Pradesh	30-06-2019	Monthly	3.05	11755881.0	42.05	Rural
2	Andhra Pradesh	31-07-2019	Monthly	3.75	12086707.0	43.50	Rural

Next steps: Generate code with df1 | View recommended plots | New interactive sheet

EXPLORATORY DATA ANALYSIS

```
sns.barplot(data=df1, x='Region', y=' Estimated Unemployment Rate (%)')
plt.xticks(rotation=90)
plt.show()
```



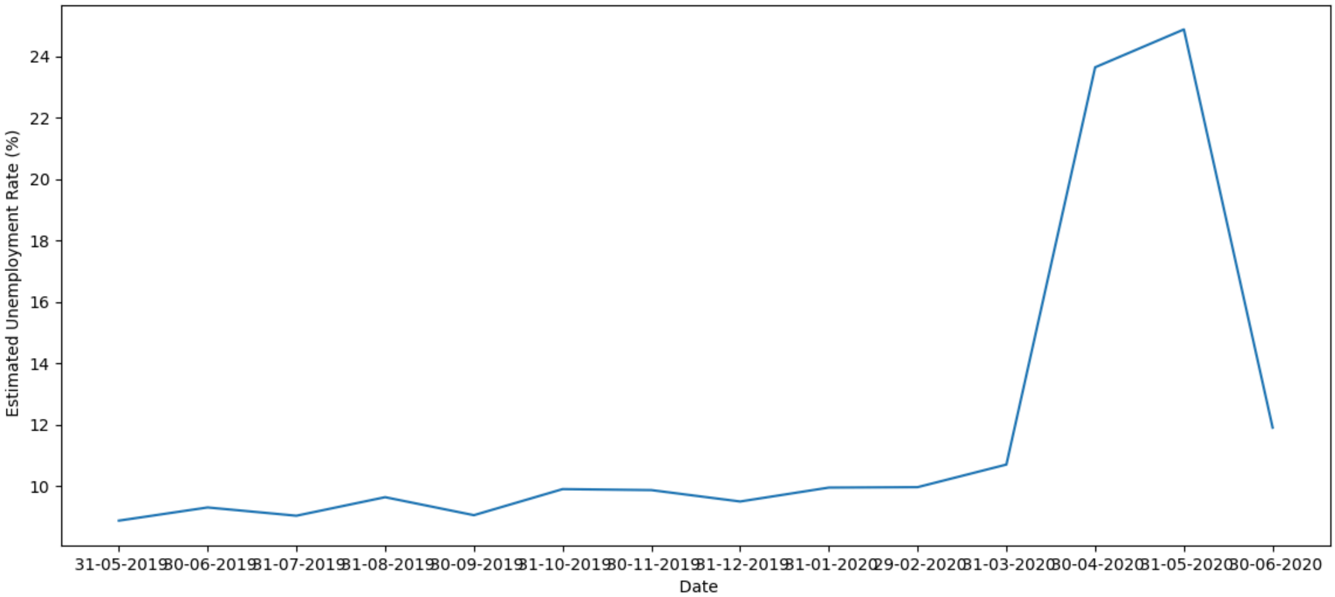

```
df1.head()
```

	Region	Date	Frequency	Estimated Unemployment Rate (%)	Estimated Employed	Estimated Labour Participation Rate (%)	Area
0	Andhra Pradesh	31-05-2019	Monthly	3.65	11999139.0	43.24	Rural
1	Andhra Pradesh	30-06-2019	Monthly	3.05	11755881.0	42.05	Rural
2	Andhra Pradesh	31-07-2019	Monthly	3.75	12086707.0	43.50	Rural

Next steps: Generate code with df1 View recommended plots New interactive sheet

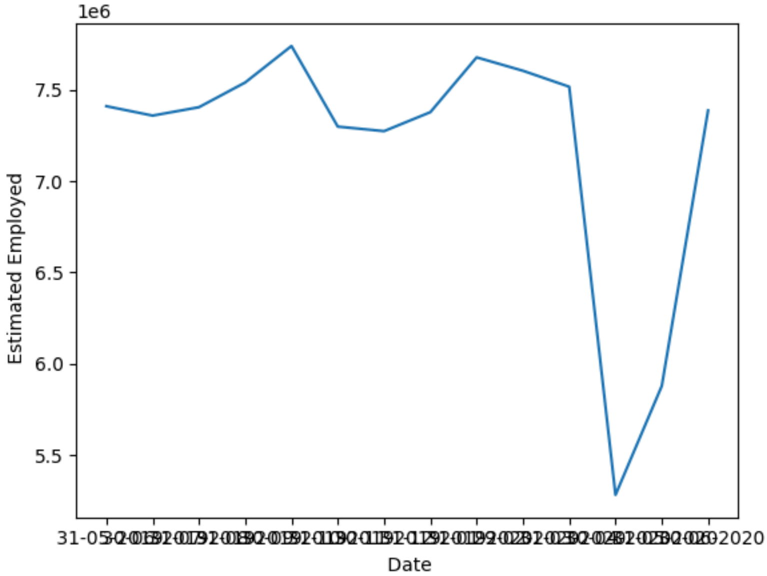
```
fig, ax = plt.subplots(figsize=(14, 6))
sns.lineplot(data=df1,x=' Date',y=' Estimated Unemployment Rate (%)',ci=None)
plt.show()
```

```
<ipython-input-111-ca29d06e5dbc>:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=df1,x=' Date',y=' Estimated Unemployment Rate (%)',ci=None)
```



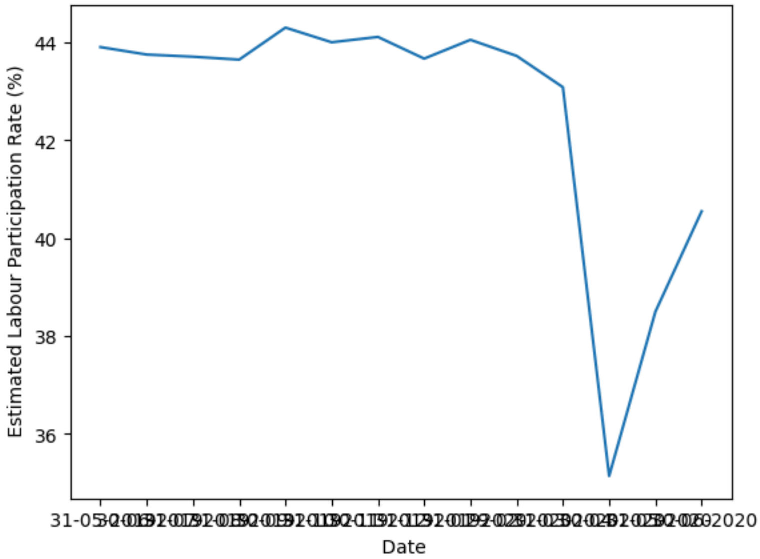

```
sns.lineplot(data=df1,x=' Date',y=' Estimated Employed',ci=None)
plt.show()
```

```
<ipython-input-112-10d3bc17e3bc>:1: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=df1,x=' Date',y=' Estimated Employed',ci=None)
```




```
plt.show()
sns.lineplot(data=df1,x=' Date',y=' Estimated Labour Participation Rate (%)',ci=None)
```

```
<ipython-input-113-1227a9dac27d>:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=df1,x=' Date',y=' Estimated Labour Participation Rate (%)',ci=None)
<Axes: xlabel=' Date', ylabel=' Estimated Labour Participation Rate (%)'>
```




```
df1[' Frequency'].unique()
```

```
array([' Monthly', nan, 'Monthly'], dtype=object)
```

```
df1.isnull().sum().sum()
```

```
np.int64(196)
```

```
df1.shape
```

```
(768, 7)
```

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

```
np.int64(0)
```

```
df1.dropna(inplace=True)
```

```
df1
```

	Region	Date	Frequency	Estimated Unemployment Rate (%)	Estimated Employed	Estimated Labour Participation Rate (%)	Area
0	Andhra Pradesh	31-05-2019	Monthly	3.65	11999139.0	43.24	Rural
1	Andhra Pradesh	30-06-2019	Monthly	3.05	11755881.0	42.05	Rural
2	Andhra Pradesh	31-07-2019	Monthly	3.75	12086707.0	43.50	Rural
3	Andhra Pradesh	31-08-2019	Monthly	3.32	12285693.0	43.97	Rural
4	Andhra Pradesh	30-09-2019	Monthly	5.17	12256762.0	44.68	Rural
...	...	...	...	...	...	...	...
749	West Bengal	29-02-2020	Monthly	7.55	10871168.0	44.09	Urban
750	West Bengal	31-03-2020	Monthly	6.67	10806105.0	43.34	Urban

Next steps: Generate code with df1 | View recommended plots | New interactive sheet

THIS TO BE CONTINUED AFTER SOME DAYS THE ABOVE

```
df1.head()
```

	Region	Date	Frequency	Estimated Unemployment Rate (%)	Estimated Employed	Estimated Labour Participation Rate (%)	Area
0	Andhra Pradesh	31-05-2019	Monthly	3.65	11999139.0	43.24	Rural
1	Andhra Pradesh	30-06-2019	Monthly	3.05	11755881.0	42.05	Rural
2	Andhra Pradesh	31-07-2019	Monthly	3.75	12086707.0	43.50	Rural

Next steps: Generate code with df1 View recommended plots New interactive sheet

```
df1.columns
```

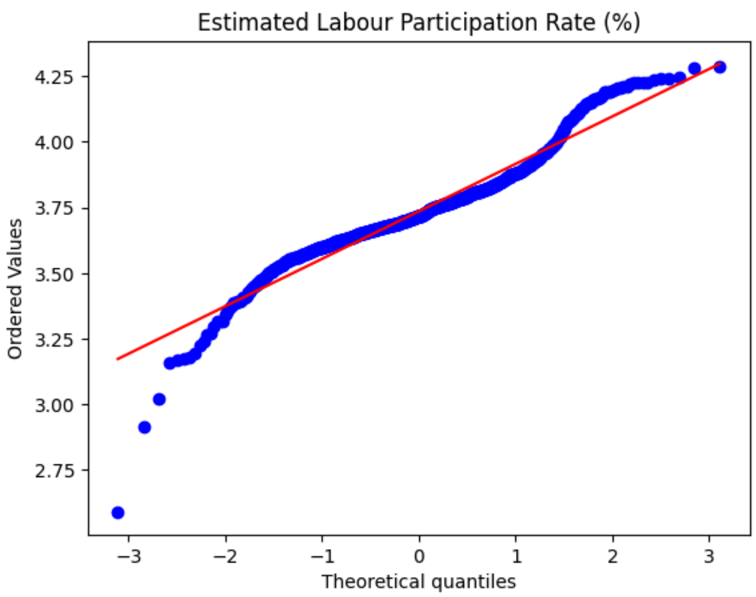
```
Index(['Region', ' Date', ' Frequency', ' Estimated Unemployment Rate (%)',
       ' Estimated Employed', ' Estimated Labour Participation Rate (%)',
       'Area'],
      dtype='object')
```

```
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
```

```
df1[' Estimated Employed']=s.fit_transform(df1[' Estimated Employed'].values.reshape(-1,1))
```

```
import scipy.stats as stats
```

```
stats.probplot(np.log(df1[' Estimated Labour Participation Rate (%)']), dist="norm", plot=plt)
plt.title('Estimated Labour Participation Rate (%)')
plt.show()
```




```
df1.isnull().sum()
```

	0
Region	0
Date	0
Frequency	0
Estimated Unemployment Rate (%)	0
Estimated Employed	0
Estimated Labour Participation Rate (%)	0
Area	0

dtype: int64

PREPROCESSING

```
df1.dropna(inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder
le1=LabelEncoder()
```

```
for i in df1.columns:
  if df1[i].dtypes=='object':
    df1[i]=le1.fit_transform(df1[i])
```

```
x1=df1.drop([' Estimated Employed','Area'],axis=1)
```

```
x1.isnull().sum()
```

	0
Region	0
Date	0
Frequency	0
Estimated Unemployment Rate (%)	0
Estimated Labour Participation Rate (%)	0

dtype: int64

```
y1=df1[' Estimated Employed']
```

```
y1.isnull().sum()
```

```
np.int64(0)
```

MODEL BUILDIING

```
X1_train,X1_test,y1_train,y1_test=train_test_split(x1,y1,test_size=0.2,random_state=43)
```

```
para1={
    'n_estimators':[10,20,30,40],
    'max_depth':[4,6,7],
    'min_samples_split':[30,40],
    'min_samples_leaf':[4,5,6],
    'criterion':['squared_error','absolute_error']
}
```

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
```

```
g=GridSearchCV(estimator=rf,param_grid=para1,cv=5,n_jobs=-1,scoring='neg_mean_squared_error')
g.fit(X1_train,y1_train)
```

```
GridSearchCV
best_estimator_: RandomForestRegressor
RandomForestRegressor
RandomForestRegressor(max_depth=7, min_samples_leaf=4, min_samples_split=30,
                      n_estimators=20)
```

```
g.best_score_
```

```
np.float64(-0.3933487557143115)
```

```
rf=RandomForestRegressor(max_depth=8, min_samples_leaf=5, min_samples_split=20,
                      n_estimators=18)
```

```
rf.fit(X1_train,y1_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=8, min_samples_leaf=5, min_samples_split=20,
                      n_estimators=18)
```

```
mean_squared_error(rf.predict(X1_test),y1_test)
```

```
0.6019545686377283
```

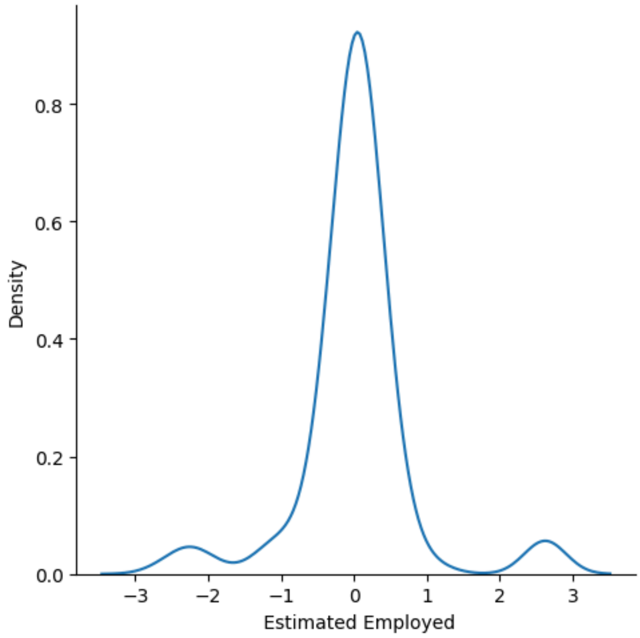
```
x1.columns
```

```
Index(['Region', ' Date', ' Frequency', ' Estimated Unemployment Rate (%)',
       ' Estimated Labour Participation Rate (%)'],
      dtype='object')
```

```
r2_score(rf.predict(X1_test),y1_test)
```

```
0.157928096623694
```

```
sns.displot(rf.predict(X1_test)-y1_test,kind='kde')
plt.show()
```



CAR PRICE PRIDICTION

```
df3=pd.read_csv(r'/content/car data.csv')
```

```
df3.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

Next steps: Generate code with df3 View recommended plots New interactive sheet

EXPLORATORY DATA ANALYSIS

```
df3['Fuel_Type'].unique()
```

```
array(['Petrol', 'Diesel', 'CNG'], dtype=object)
```

```
df3['Selling_type'].unique()
```

```
array(['Dealer', 'Individual'], dtype=object)
```

```
df3['Transmission'].unique()
```

```
array(['Manual', 'Automatic'], dtype=object)
```

```
df3['Owner'].unique()
```

```
array([0, 1, 3])
```

```
df3.shape
```

```
(301, 9)
```

```
df3.isnull().sum()
```

	0
Car_Name	0
Year	0
Selling_Price	0
Present_Price	0
Driven_kms	0
Fuel_Type	0
Selling_type	0
Transmission	0
Owner	0

dtype: int64

```
df3.duplicated().sum()
```

```
np.int64(2)
```

```
df3[df3.duplicated()]
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
17	ertiga	2016	7.75	10.79	43000	Diesel	Dealer	Manual	0
93	fortuner	2015	23.00	30.61	40000	Diesel	Dealer	Automatic	0

```
df3.drop(93,axis=0,inplace=True)
```

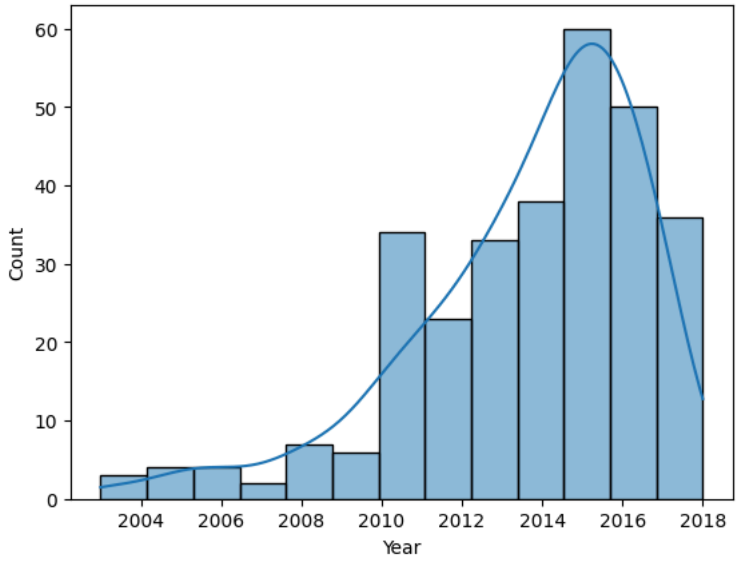
```
df3.info()
```

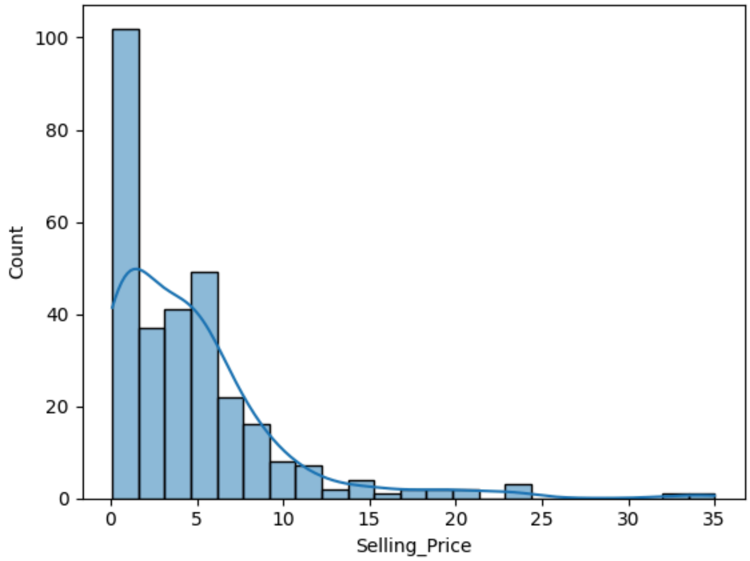
```
<class 'pandas.core.frame.DataFrame'>
Index: 300 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       300 non-null    object 
 1   Year           300 non-null    int64  
 2   Selling_Price  300 non-null    float64
 3   Present_Price  300 non-null    float64
 4   Driven_kms     300 non-null    int64  
 5   Fuel_Type      300 non-null    object 
 6   Selling_type   300 non-null    object 
 7   Transmission   300 non-null    object 
 8   Owner          300 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 23.4+ KB
```

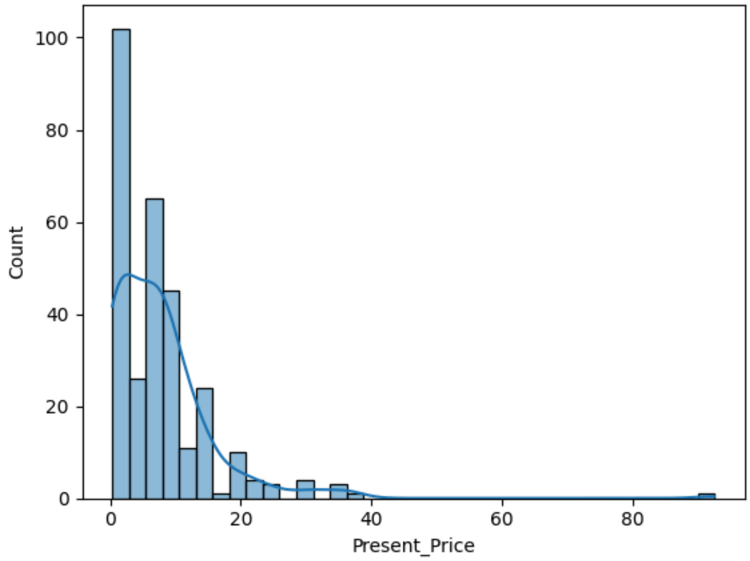
```
import numpy as np
```

```
for i in df3.columns:
  if df3[i].dtypes=='object':
    continue
```

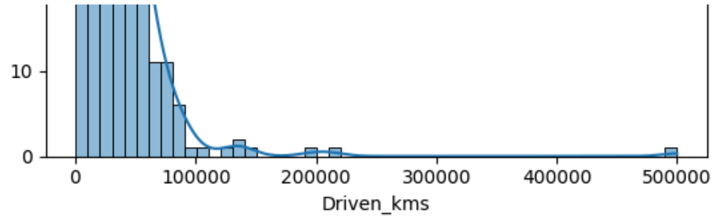
```
sns.histplot(data=df3,x=(i),kde=True)
plt.show()
```

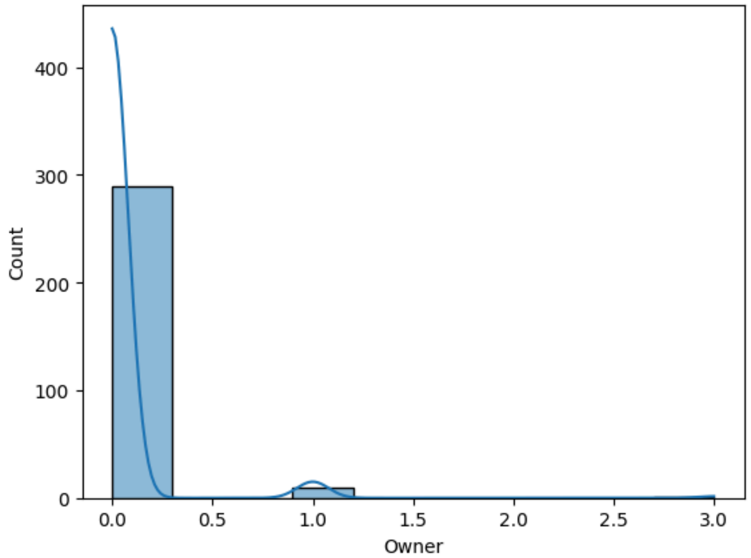

Count
Year
2004
2006
2008
2010
2012
2014
2016
2018


Count
Selling_Price


Count
Present_Price


Count






```
np.mean(df3['Selling_Price'])
```

```
np.float64(4.6001666666666665)
```

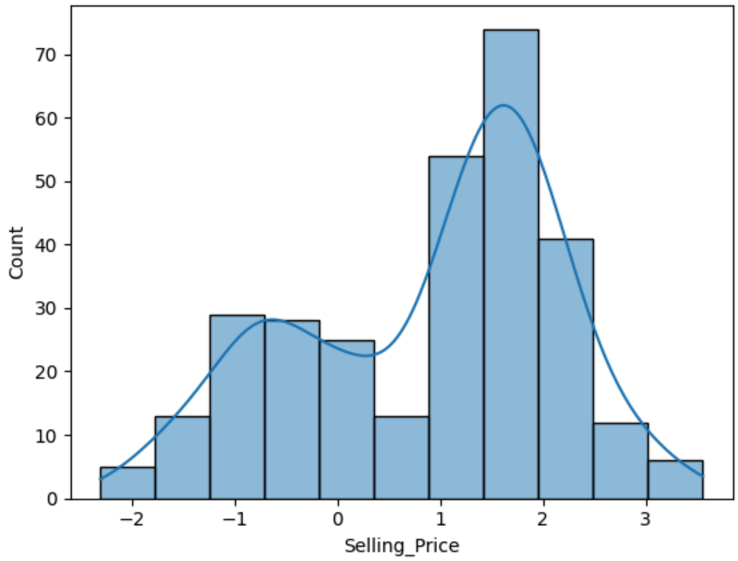
```
np.mean(np.sqrt(df3['Selling_Price']))
```

```
np.float64(1.8761384695271879)
```

```
np.mean(np.log(df3['Selling_Price']))
```

```
np.float64(0.9039446587834338)
```

```
sns.histplot(data=df3,x=np.log(df3['Selling_Price']),kde=True)
plt.show()
```




```
df3['Selling_Price']=np.log(df3['Selling_Price'])
```

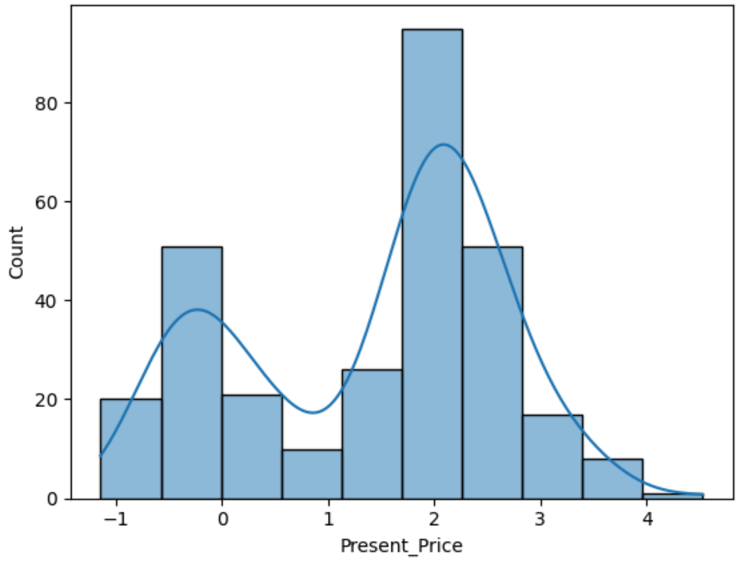
```
np.mean(np.sqrt(df3['Present_Price']))
```

```
np.float64(2.4119916890364816)
```

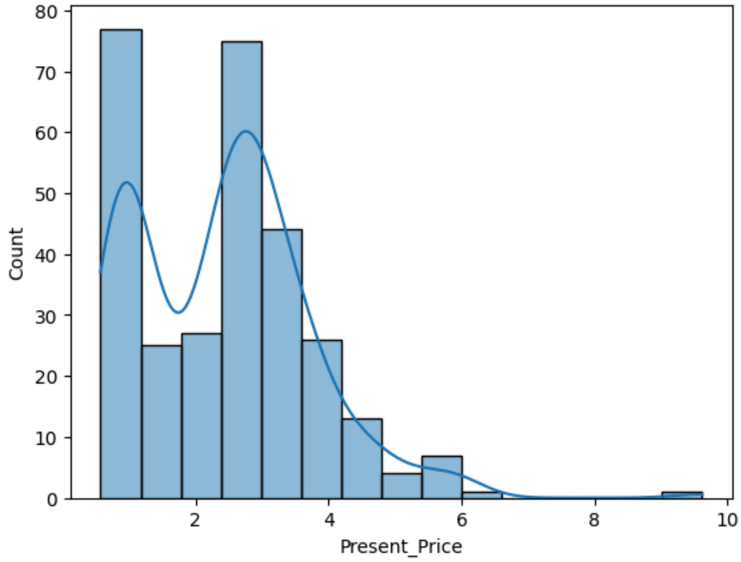
```
np.mean(np.log(df3['Present_Price']))
```

```
np.float64(1.4265831790876917)
```

```
sns.histplot(data=df3,x=np.log(df3['Present_Price']),kde=True)
plt.show()
```




```
sns.histplot(data=df3,x=np.sqrt(df3['Present_Price']),kde=True)
plt.show()
```




```
df3
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	1.208960	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	1.558145	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	1.981001	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	1.047319	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	1.526056	6.87	42450	Diesel	Dealer	Manual	0
...	...	...	...	...	...	...	...	...	...
296	city	2016	2.251292	11.60	33988	Diesel	Dealer	Manual	0
297	brio	2015	1.386294	5.90	60000	Petrol	Dealer	Manual	0
298	city	2009	1.208960	11.00	87934	Petrol	Dealer	Manual	0
299	city	2017	2.442347	12.50	9000	Diesel	Dealer	Manual	0
300	brio	2016	1.667707	5.90	5464	Petrol	Dealer	Manual	0

300 rows × 9 columns

Next steps: Generate code with df3 View recommended plots New interactive sheet

VISUALIZATION

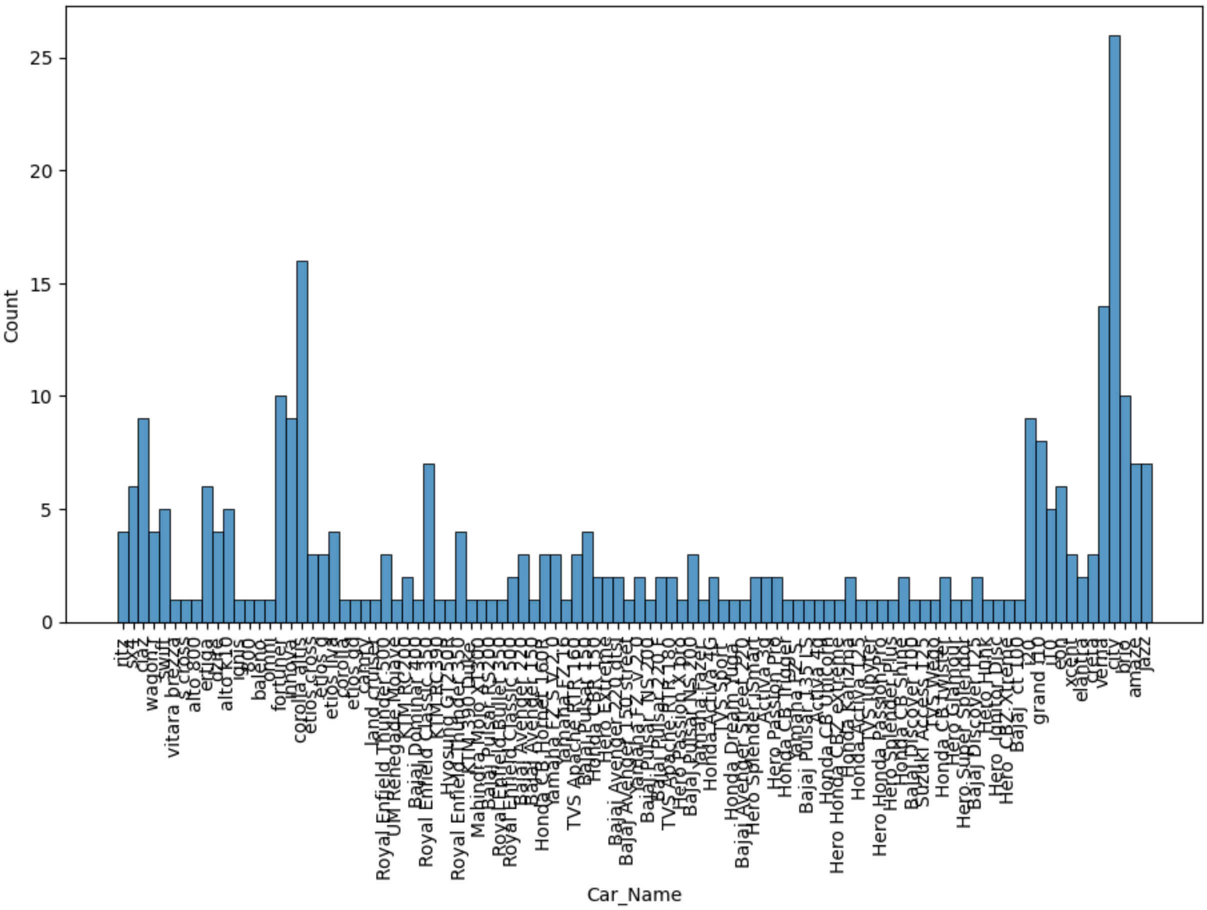
```
'''
What is the average selling price of used cars?

Which car models are the most frequently sold?

How does the fuel type affect the selling price?

Are older cars (based on Year) priced significantly lower?

What is the distribution of Driven_kms — are most cars lightly used?

How many cars are sold by Dealers vs. Individuals?

What's the proportion of manual vs. automatic transmission?
'''
```

```
'\nWhat is the average selling price of used cars?\n\nWhich car models are the most frequently sold?\n\nHow does the fuel type affe
ct the selling price?\n\nAre older cars (based on Year) priced significantly lower?\n\nWhat is the distribution of Driven_kms — are
most cars lightly used?\n\nHow many cars are sold by Dealers vs. Individuals?\n\nWhat's the proportion of manual vs. automatic tran
smission?\n'
```

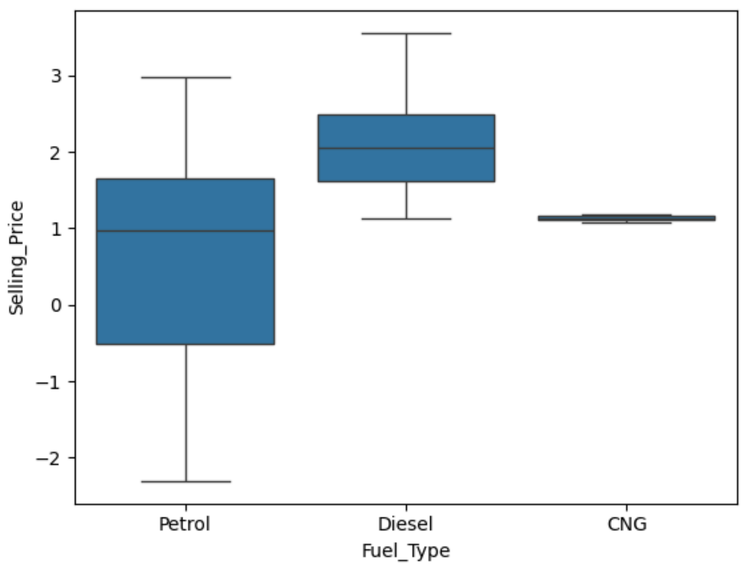
```
df3['Year'].unique()
```

```
array([2014, 2013, 2017, 2011, 2018, 2015, 2016, 2009, 2010, 2012, 2003,
       2008, 2006, 2005, 2004, 2007])
```

```
plt.figure(figsize=(11,6))
sns.histplot(data=df3,x=df3['Car_Name'])
plt.xticks(rotation=90)
plt.show()
```




```
sns.boxplot(data=df3,x='Fuel_Type',y='Selling_Price')
plt.show()
```




```
df31=df3.groupby('Year')['Selling_Price'].mean().reset_index()
```

```
df31
```

	Year	Selling_Price
0	2003	-0.119446
1	2004	0.405465
2	2005	0.476920
3	2006	-0.148892
4	2007	-1.864851
5	2008	-0.815614
6	2009	0.938180
7	2010	0.682096
8	2011	0.507475
9	2012	0.773808
10	2013	0.702343
11	2014	1.105755
12	2015	1.293036
13	2016	1.072335
14	2017	1.063544
15	2018	2.224624

Next steps: Generate code with df31 View recommended plots New interactive sheet

```
sns.lineplot(data=df31,x='Year',y='Selling_Price')
plt.show()
```


```
df3['Car_Name'].value_counts()
```

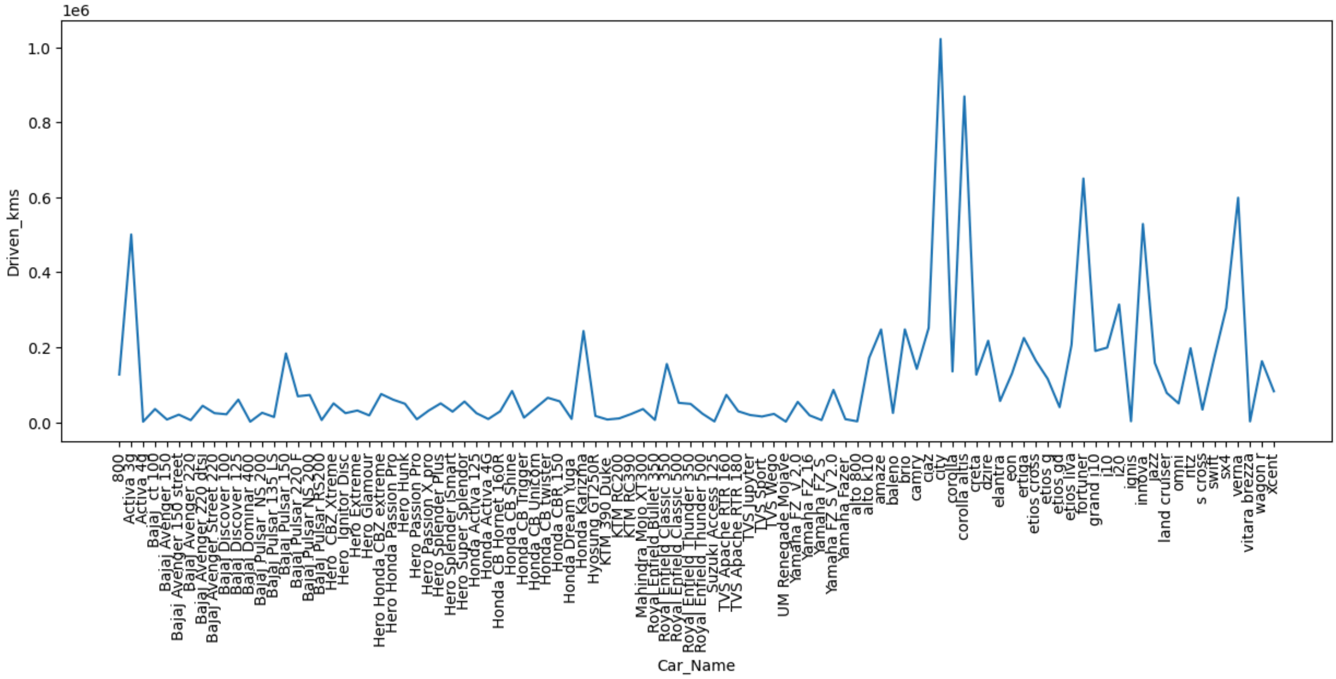
Car_Name	count
city	26
corolla altis	16
verna	14
brio	10
fortuner	10
...	...
Honda Activa 125	1
Hero Hunk	1
Hero Ignitor Disc	1
Hero CBZ Xtreme	1
Bajaj ct 100	1

98 rows × 1 columns

dtype: int64

```
df32=df3.groupby('Car_Name')['Driven_kms'].sum().reset_index()
```

```
plt.figure(figsize=(15,5))
sns.lineplot(data=df32,x='Car_Name',y='Driven_kms')
plt.xticks(rotation=90)
plt.show()
```




```
df3.head(2)
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	1.208960	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	1.558145	9.54	43000	Diesel	Dealer	Manual	0

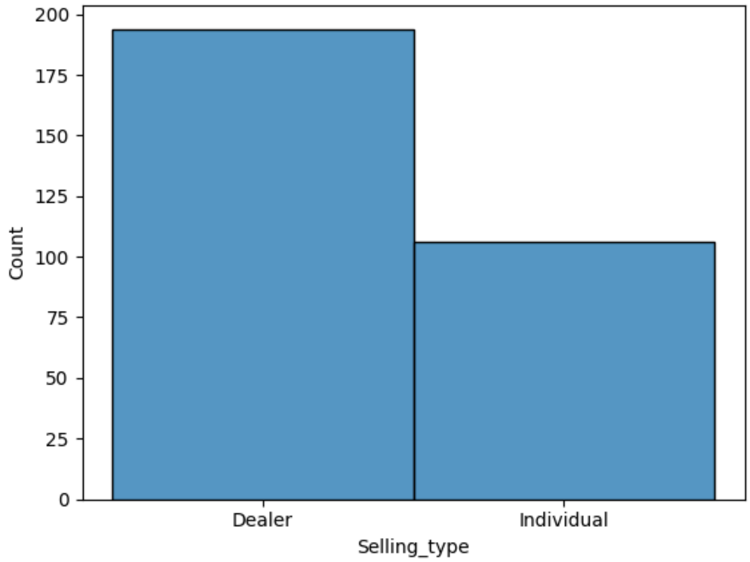
Next steps: Generate code with df3 View recommended plots New interactive sheet

```
df3.groupby('Selling_type')['Car_Name']
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x785fe4ffe250>
```

```
k=df3['Selling_type'].value_counts()
```

```
sns.histplot(data=df3, x='Selling_type')
plt.show()
```




```
df3['Transmission'].value_counts()
```

Transmission	count
Manual	261
Automatic	39

dtype: int64

```
df3['Current_Year']=2025-df3['Year']
```

```
df3.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner	Current_Year
0	ritz	2014	1.208960	5.59	27000	Petrol	Dealer	Manual	0	11
1	sx4	2013	1.558145	9.54	43000	Diesel	Dealer	Manual	0	12
2	ciaz	2017	1.981001	9.85	6900	Petrol	Dealer	Manual	0	8
3	wagon r	2011	1.047319	4.15	5200	Petrol	Dealer	Manual	0	14
4	swift	2014	1.526056	6.87	42450	Diesel	Dealer	Manual	0	11

Next steps: Generate code with df3 View recommended plots New interactive sheet

```
df3=df3.sort_values(by='Year')
```

```
df3.head(2)
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner	Current_Year
39	sx4	2003	0.810930	7.98	62000	Petrol	Dealer	Manual	0	22
37	800	2003	-1.049822	2.28	127000	Petrol	Individual	Manual	0	22

Next steps: Generate code with df3 View recommended plots New interactive sheet

```
from sklearn.preprocessing import LabelBinarizer
```