Car price prediction

In this notebook we are going to do a regression project to predict selling price of a car based on some features that are provided in dataset.

Data Dictionary

- **Car_name** -----> The name of the car.
- **Year** -----> Year of car production.
- **Selling_price** -----> The price that car was dealed with: target value
- **Present_price** -----> Current car's price .
- **Kms_Driven** -----> The distance traveled by the car in kilometer.
- **Fuel_Type** -----> Fuel type that car uses.
- **Seller_type** -----> Car has sold be a dealer or a individual.
- **Transmission** -----> Type of gearbox that car has.
- **Owner** -----> The number of previous owner.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Defining the path
path = 'E:/EDU/Programming/Python/AI/car_prediction_data.csv'
```

In [3]:
```
dataset = pd.read_csv(path)
dataset.head()
```

Out[3]:

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

Data preprocessing and Exploratory data analysis.

In [4]:
```
# see the unique values
print(dataset['Fuel_Type'].unique())
print(dataset['Seller_Type'].unique())
print(dataset['Transmission'].unique())
print(dataset['Owner'].unique())
```

```
['Petrol' 'Diesel' 'CNG']
['Dealer' 'Individual']
['Manual' 'Automatic']
[0 1 3]
```

In [5]:
```
# finding null values
dataset.isnull().sum()
```

Out[5]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

In [6]:
```
# Car name is not nessesary as it has no effect on price of the car
dataset = dataset.drop('Car_Name', axis = 1)
```

In [7]:
```
dataset.head()
```

Out[7]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [11]:
```
# insted of year of the car production it is better to have the age of it. u
import datetime
current_time = datetime.datetime.now()
current_year = current_time.year
dataset['Car_age'] = current_year - dataset['Year']
```

In [12]:
```
dataset.head()
```

Out[12]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	C
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	

In [13]:
```
# know we do not need Year column
dataset = dataset.drop('Year', axis = 1)
dataset.head()
```

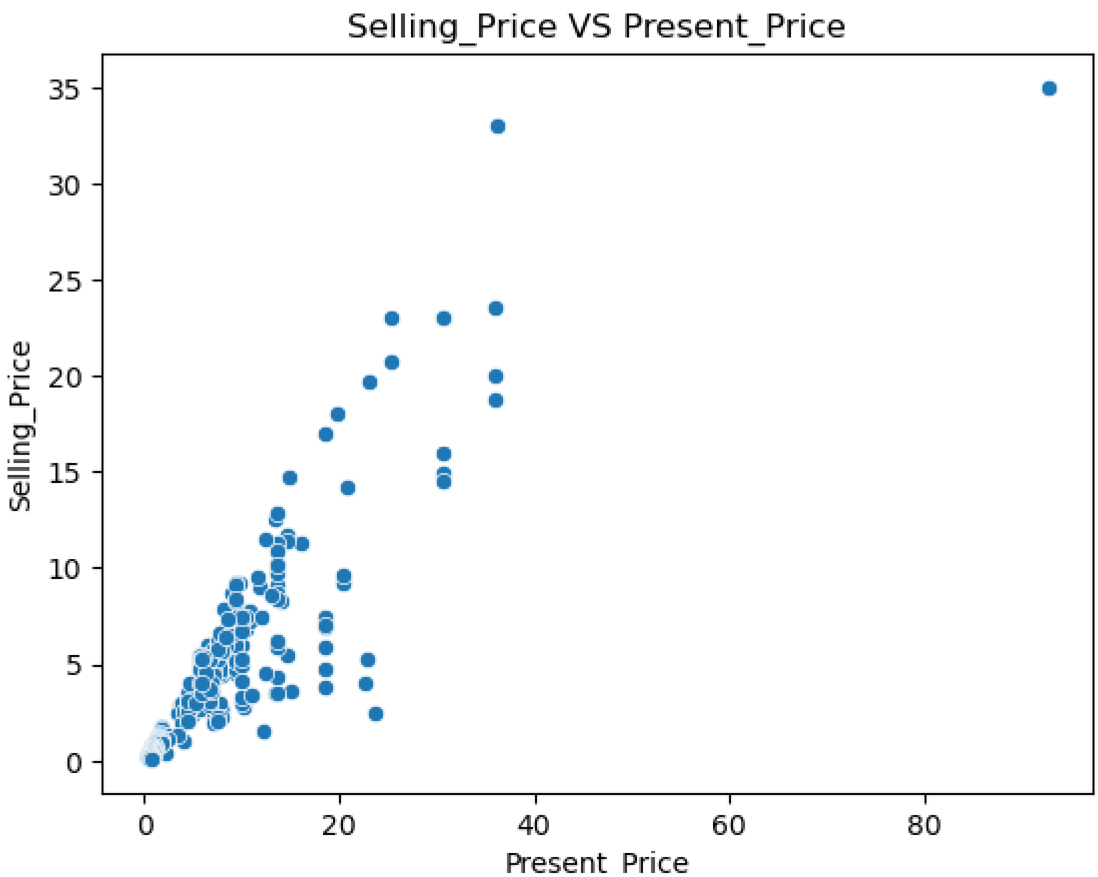
Out[13]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Car_age
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	9
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	10
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	6
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	12
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	9

Let's see the relation between indipendent variables and dependent one which is 'Selling_Price'

Selling_Price VS Present_Price

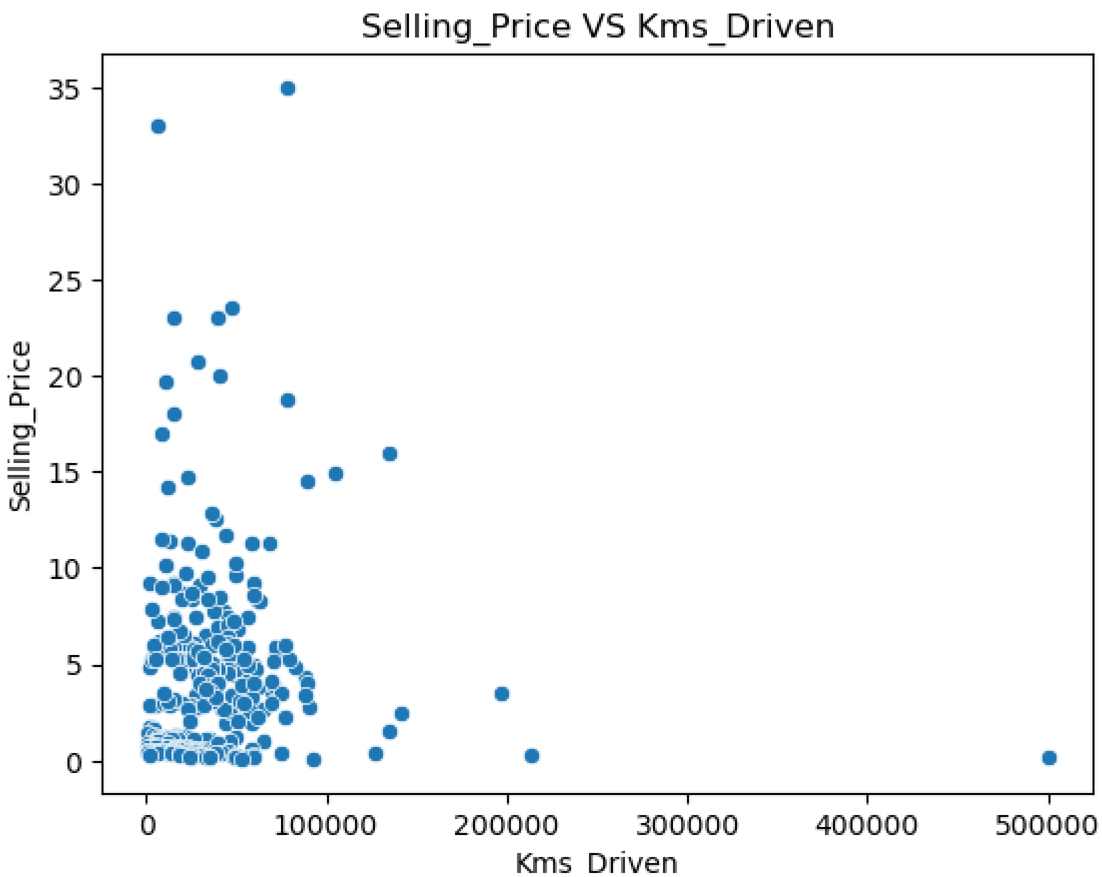
In [14]:
```
sns.scatterplot(x = 'Present_Price', y = 'Selling_Price', data = dataset).se
```



The scatter plot displays the rise in selling price in relation to the current price. Nonetheless, there are certain cars whose price is comparatively lower than others with a similar current price. Hence, there must be additional factors influencing the selling price.

Selling_Price VS Kms_Driven

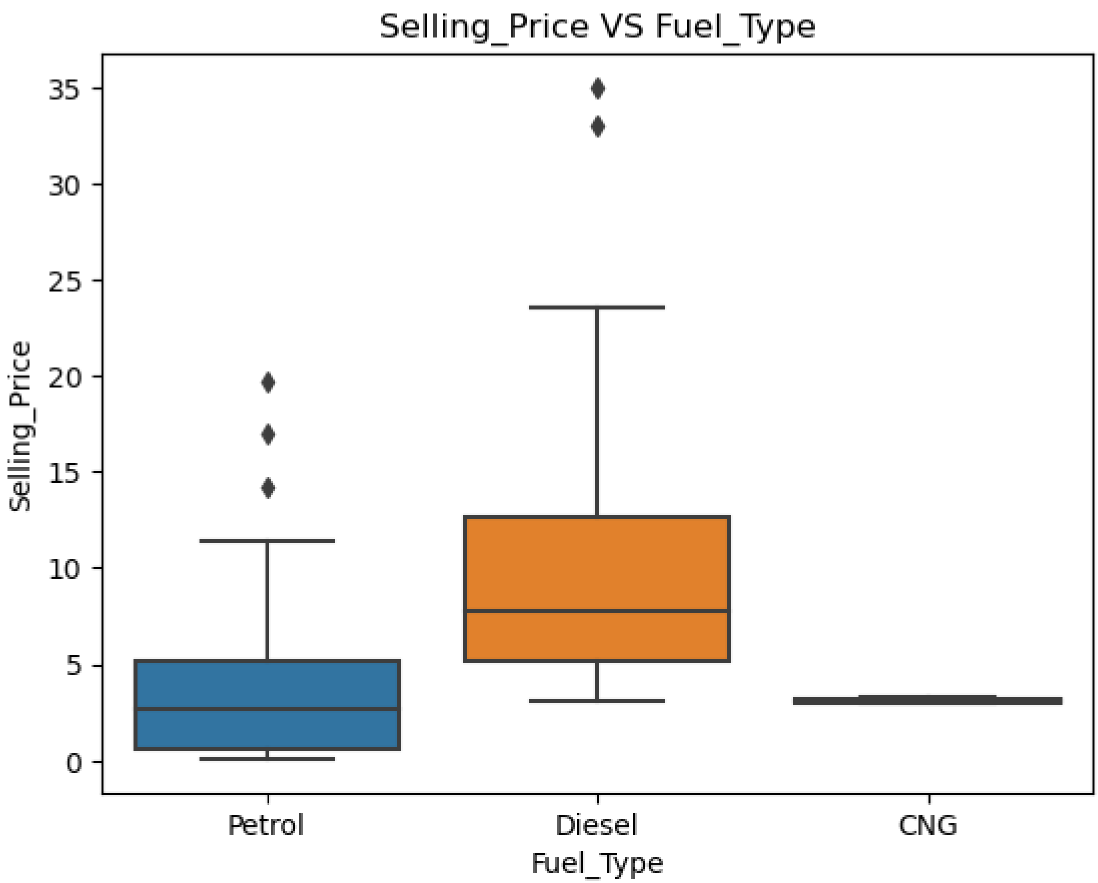
In [15]:
```
sns.scatterplot(x = 'Kms_Driven', y = 'Selling_Price', data = dataset).set_t
```



There appears to be no specific correlation between the `selling price` and `kms_Driven` , as observed.

Selling_Price VS Fuel_Type

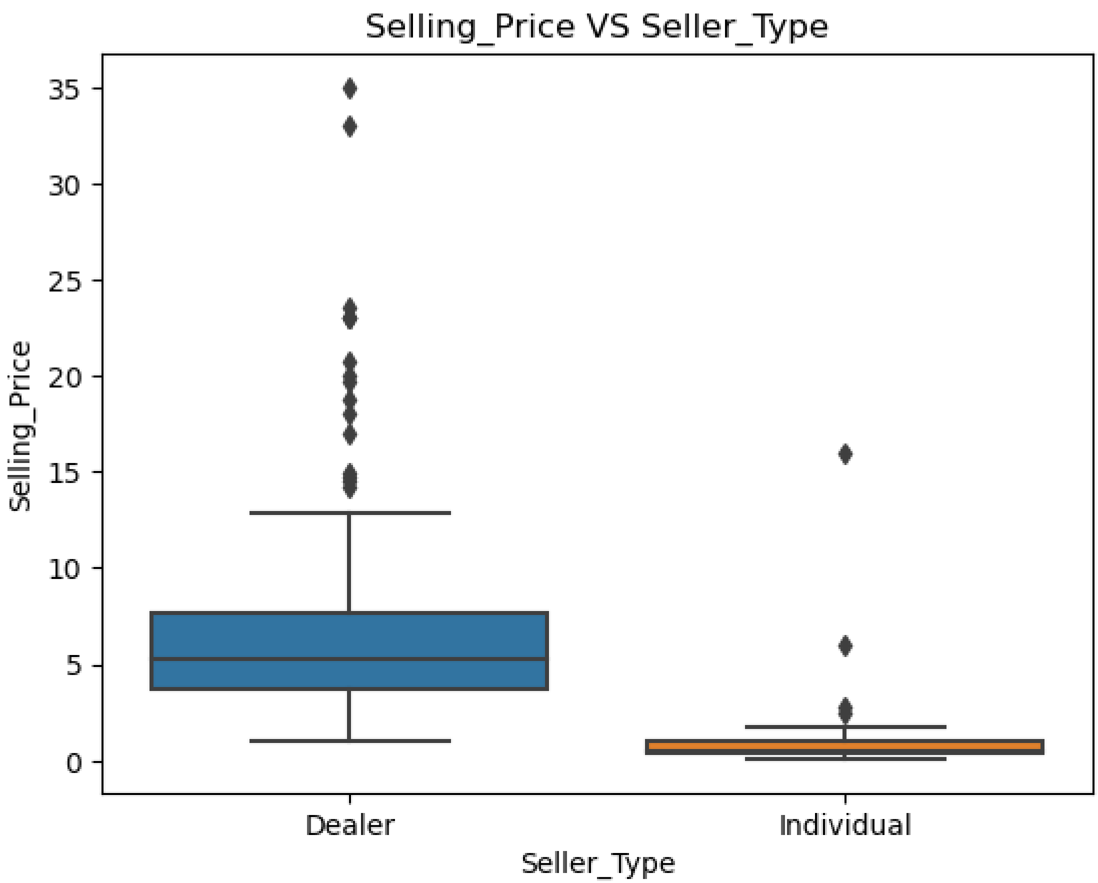
In [16]:
```
sns.boxplot(x = 'Fuel_Type', y = 'Selling_Price', data = dataset).set_title(
```



The plot indicates that cars fueled by `diesel` tend to have higher selling prices. It is important to take into account the limited number of car samples with CNG fuel type in this dataset.

Selling_Price VS Seller_Type

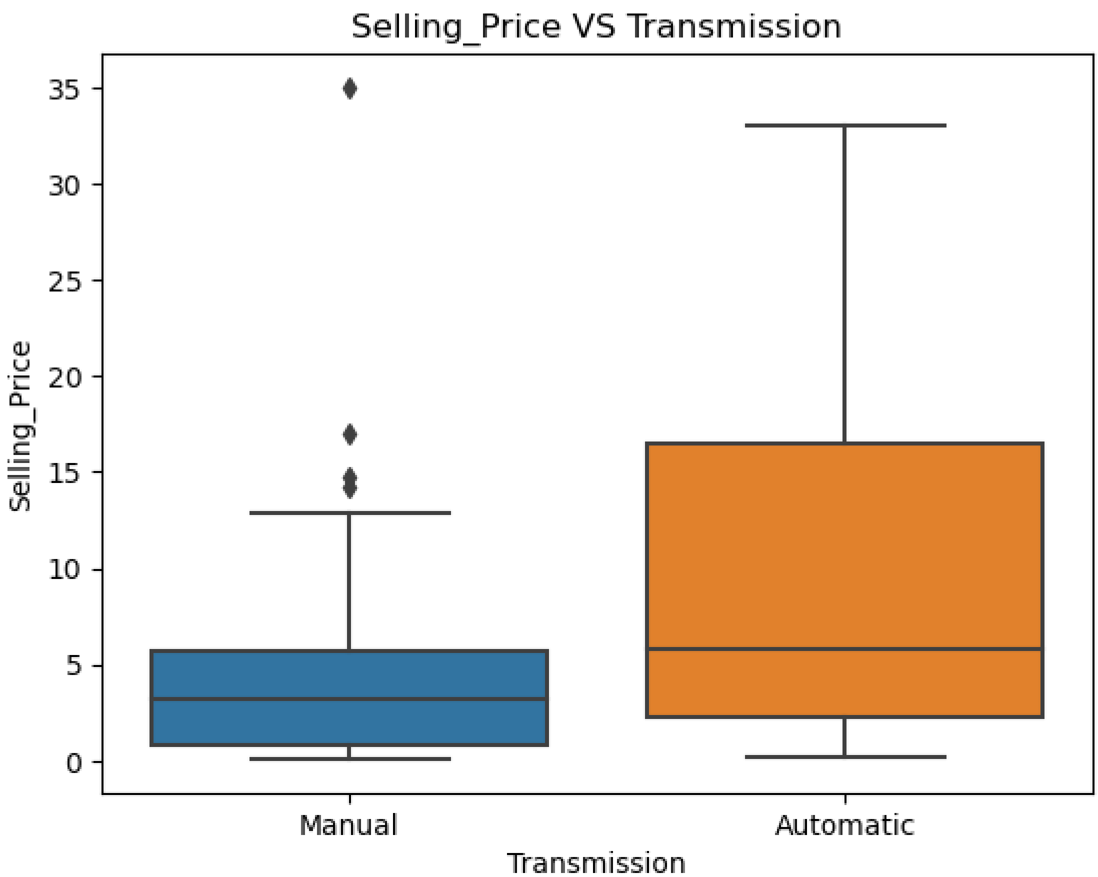
In [17]:
```
sns.boxplot(x = 'Seller_Type', y = 'Selling_Price', data = dataset).set_titl
```



We can observe the impact of the seller type on the selling price, which suggests that cars sold by a "Dealer" generally fetch higher selling prices.

Selling_Price VS Transmission

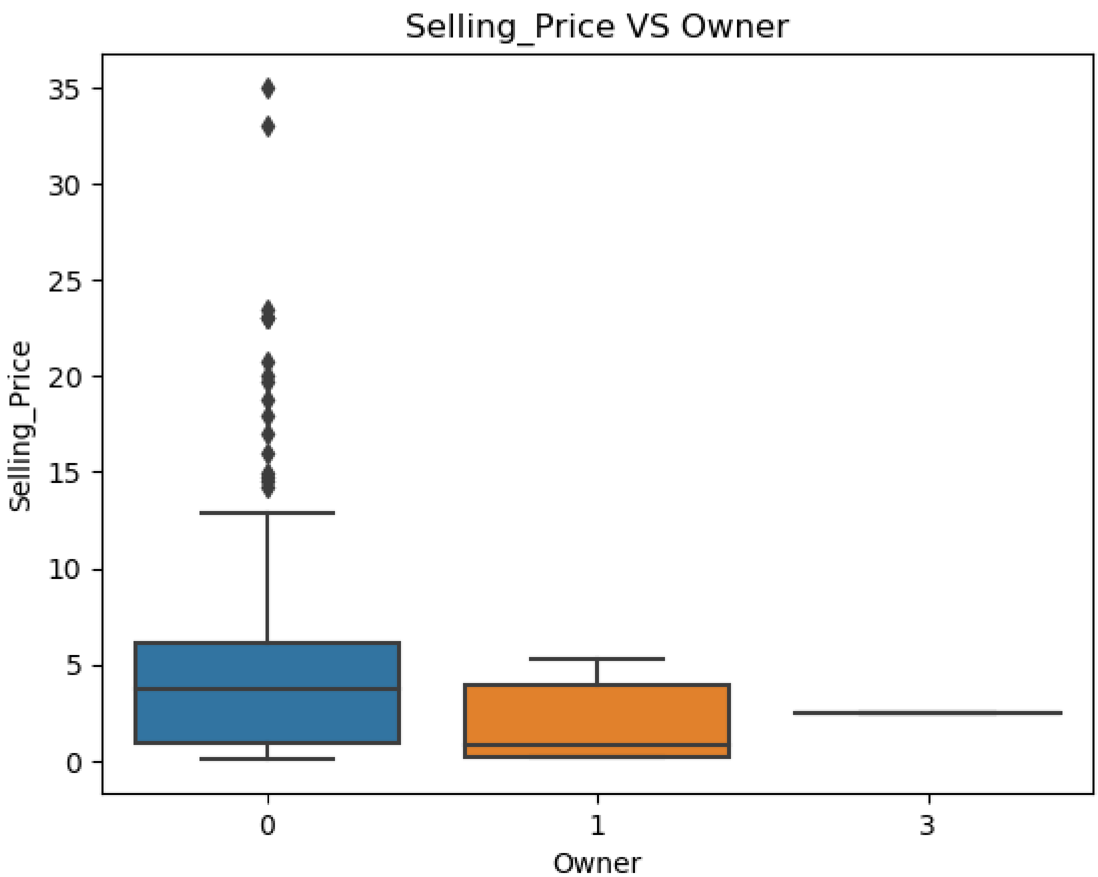
In [18]:
```
sns.boxplot(x = 'Transmission', y = 'Selling_Price', data = dataset).set_tit
```



The presented plot demonstrates the influence of the transmission type on the selling price. It is evident that cars equipped with automatic transmission tend to be more expensive compared to those with manual transmission. Nevertheless, there are instances where some cars with manual transmission command higher selling prices.

Selling_Price VS Owner

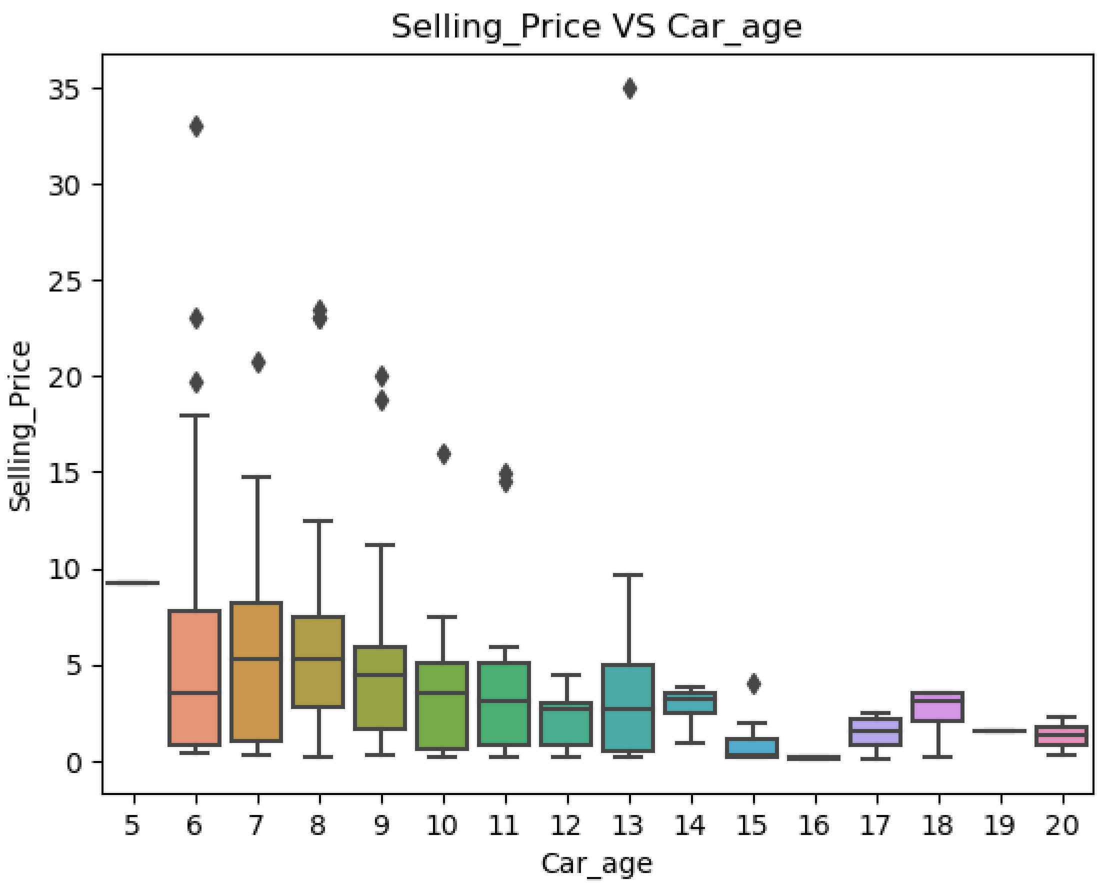
In [19]:
```
sns.boxplot(x = 'Owner', y = 'Selling_Price', data = dataset).set_title('Sel
```



The graph illustrates that the selling price typically declines as the number of previous owners increases.

Selling_Price VS Car_age

In [20]:
```
sns.boxplot(x = 'Car_age',  y = 'Selling_Price', data = dataset).set_title('
```



The last graph demonstrates that the selling price frequently decreases as the age of the car increases.

In [21]:

```
dataset
```

Out[21]:

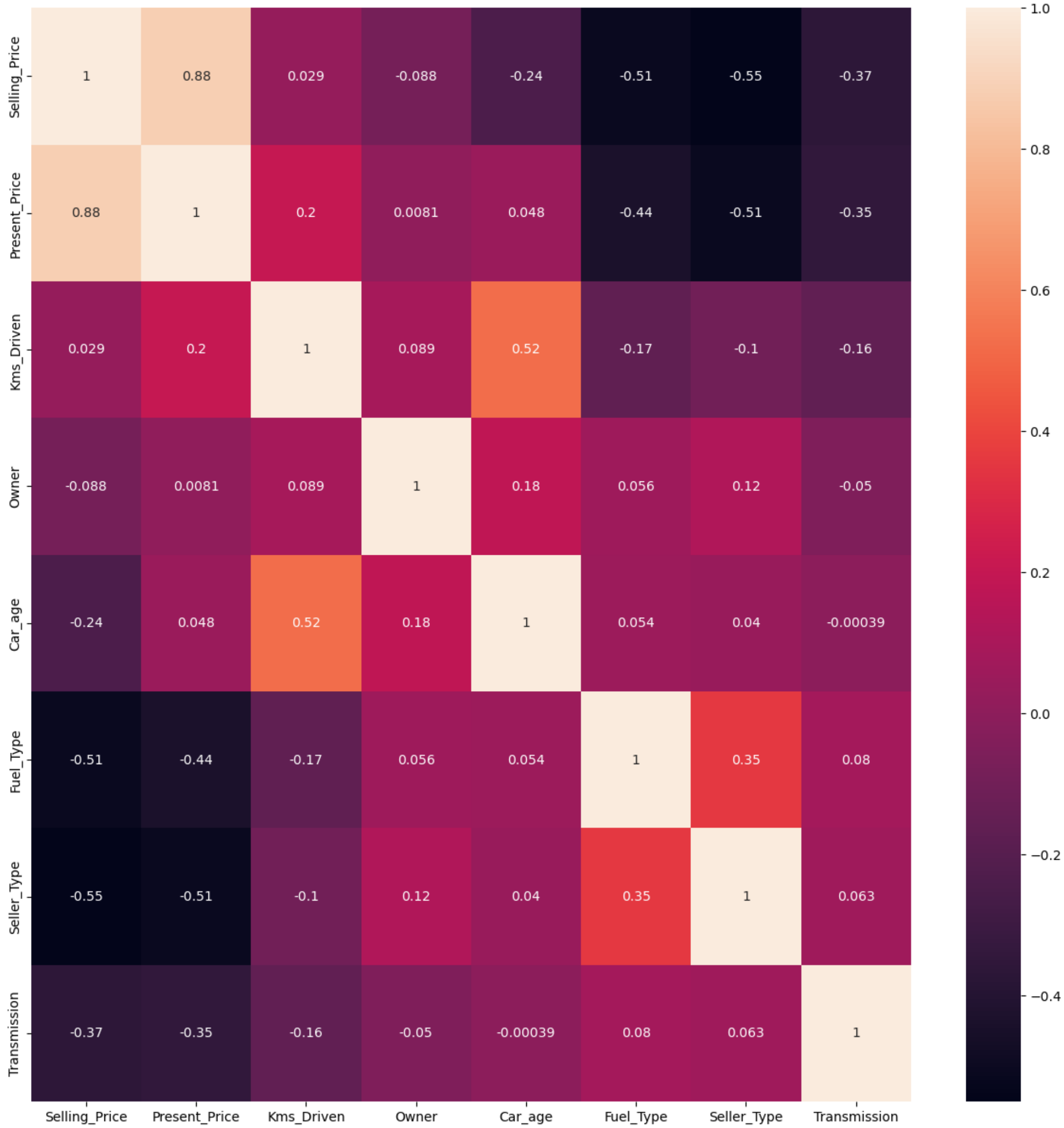
	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Car_a
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	
...	...	...	...	...	...	...	...	
296	9.50	11.60	33988	Diesel	Dealer	Manual	0	
297	4.00	5.90	60000	Petrol	Dealer	Manual	0	
298	3.35	11.00	87934	Petrol	Dealer	Manual	0	
299	11.50	12.50	9000	Diesel	Dealer	Manual	0	
300	5.30	5.90	5464	Petrol	Dealer	Manual	0	

301 rows × 8 columns

In [26]:

```
# creat a class to plot corrilation and do onehot_encoding for given dataset
class manual_functions():
    def __init__(self, dataset):

        self.dataset = dataset

    def corrilation(self):
        from sklearn.preprocessing import LabelEncoder
        global dataset
        cat_col = [c for i, c in enumerate(dataset.columns) if dataset.dtype
        if len(cat_col) > 0:
            new_dataset = dataset.copy()
            for feature in cat_col:
                le = LabelEncoder()
                label = le.fit_transform(new_dataset[feature])
                new_dataset.drop([feature], axis=1, inplace=True)
                new_dataset[feature] = label

            plt.figure(figsize = (15, 15))
            g= sns.heatmap(new_dataset.corr(),annot=True, )
        else:
            plt.figure(figsize = (15, 15))
            g= sns.heatmap(new_dataset.corr(),annot=True, )

    def onehot_encoding(self):
        global dataset
        cat_col = [c for i, c in enumerate(dataset.columns) if dataset.dtype
        for cat_features in cat_col:
            dataset = pd.get_dummies(dataset, columns = [cat_features])
        return dataset
```

In [27]:
```
1  manual_functions.corrilation(dataset)
```



In this corelation matrix heatmap, we can see that the present price has highest corrilation with selling price.

In [28]:
```
# onehot encoding
manual_functions.onehot_encoding(dataset)
```

Out[28]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Car_age	Fuel_Type_CNG	Fuel_Type_Diesel
0	3.35	5.59	27000	0	9	0	0
1	4.75	9.54	43000	0	10	0	1
2	7.25	9.85	6900	0	6	0	0
3	2.85	4.15	5200	0	12	0	0
4	4.60	6.87	42450	0	9	0	1
...	...	...	...	...	...	...	...
296	9.50	11.60	33988	0	7	0	1
297	4.00	5.90	60000	0	8	0	0
298	3.35	11.00	87934	0	14	0	0
299	11.50	12.50	9000	0	6	0	1
300	5.30	5.90	5464	0	7	0	0

301 rows × 12 columns

In [29]:
```
# here we need to detemine the y or target
target = dataset['Selling_Price']
X = dataset.drop('Selling_Price', axis = 1)

print(X.shape, target.shape)
```

```
(301, 11) (301,)
```

Feature Importance

Feature importance is a feature selection technique usually use for larg dataset. However our dataset is a small dataset we are goting to it anyway just to show the importance of features and are not going to implement it on out model.

In [30]:
```
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(X,target)
```

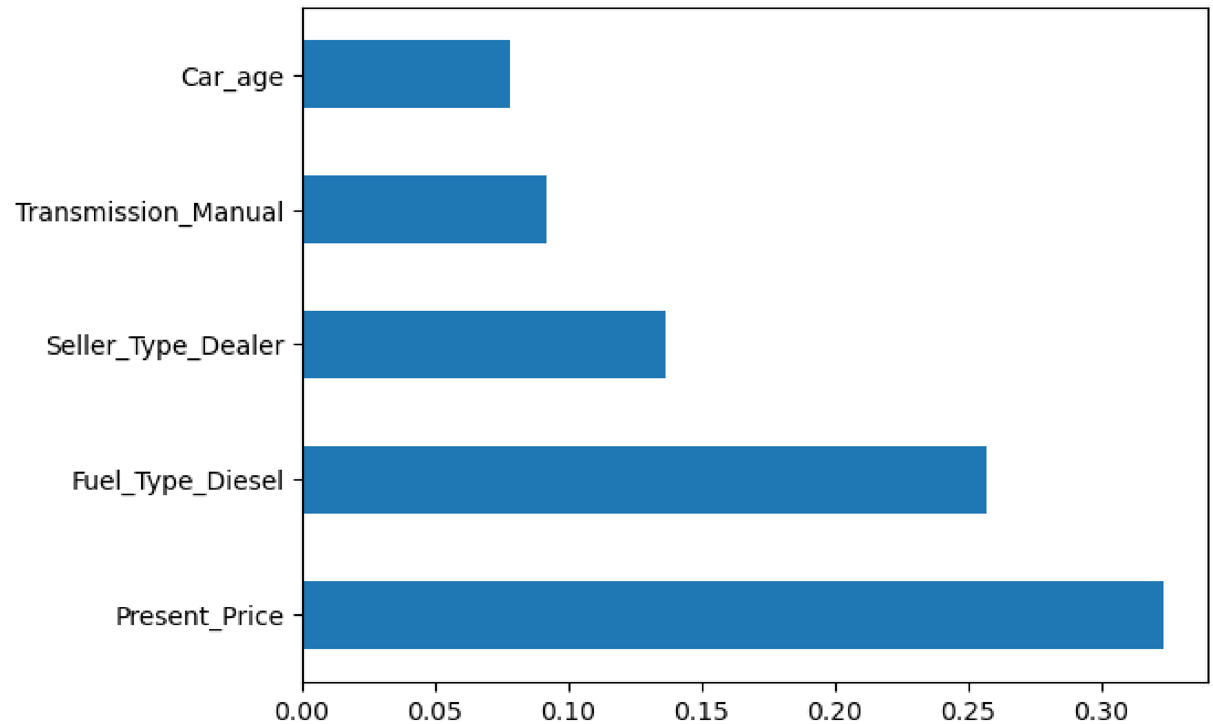
Out[30]:
```
ExtraTreesRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:
```
print(model.feature_importances_)
```

```
[3.23726240e-01 3.64512971e-02 4.70093518e-04 7.83290823e-02
 9.38298797e-05 2.56821706e-01 8.78417882e-03 1.36674032e-01
 5.44067201e-04 6.62484185e-02 9.18570550e-02]
```

In [32]:
```
feature_importance = pd.Series(model.feature_importances_, index = X.columns
feature_importance.nlargest(5).plot(kind = 'barh')
plt.show;
```



As we can see the most importannt feature to predict selling price is present price.

In [35]:
```
from sklearn.model_selection import train_test_split, cross_val_score
X_train, X_test, y_train, y_test = train_test_split(X, target, random_state=

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(225, 11) (76, 11) (225,) (76,)
```

Model Creation

In this nokebook we are going to compare four different regressor and then tune each one that have poor performance then see which one is going to perform better.

In [36]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [37]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import RandomForestRegressor
from catboost import CatBoostRegressor

from sklearn.model_selection import RandomizedSearchCV

# import metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
```

In [39]:
```
GBR_model = GradientBoostingRegressor(random_state= 42)
SGDR_model = SGDRegressor(random_state= 42)
RFR_model = RandomForestRegressor()
CBR_model = CatBoostRegressor()
```

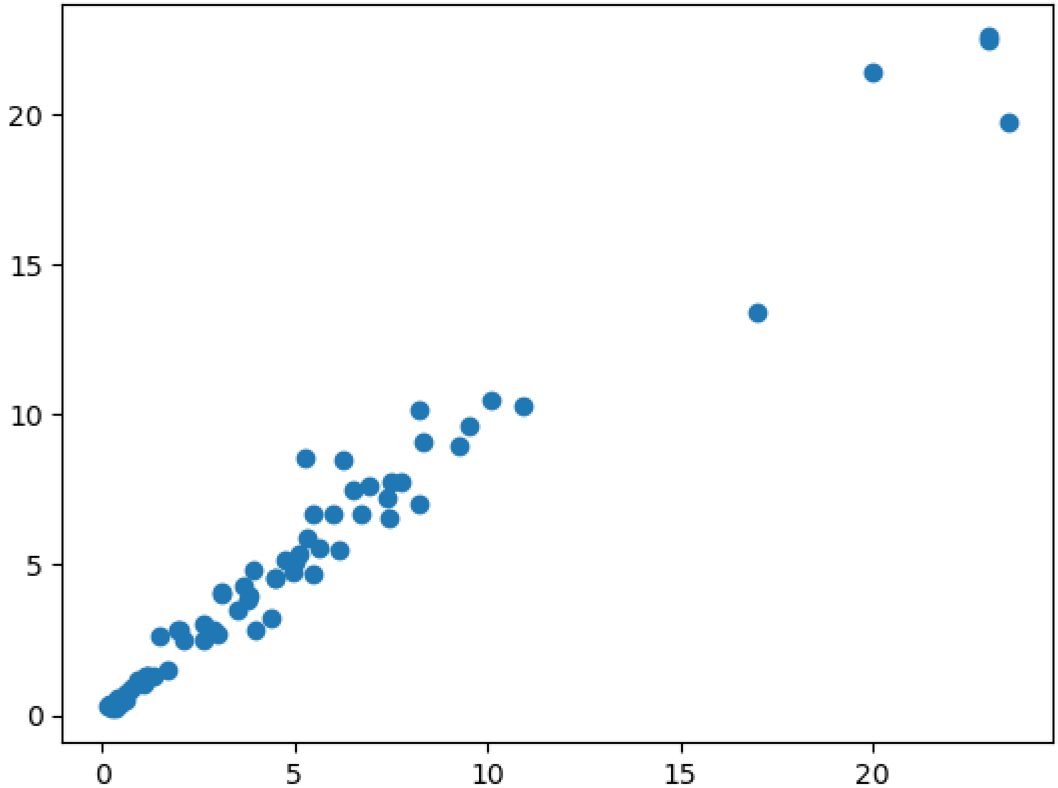
GBR_model

In [41]:
```
GBR_model = GradientBoostingRegressor(random_state= 42)
GBR_model.fit(X_train, y_train)
y_pred = GBR_model.predict(X_test)
```

In [42]:
```
# evaluate model
GBR_MAE = mean_absolute_error(y_test, y_pred)
GBR_MSE = mean_squared_error(y_test, y_pred)
GBR_R2 = r2_score(y_test, y_pred)

# printing the results
print(f'MAE:{GBR_MAE}, MSE: {GBR_MSE}, R2: {GBR_R2}')
```

```
MAE:0.5408562819175843, MSE: 0.8808967747078672, R2: 0.9679374302804042
```

In [43]:
```
plt.scatter(y_test, y_pred);
```



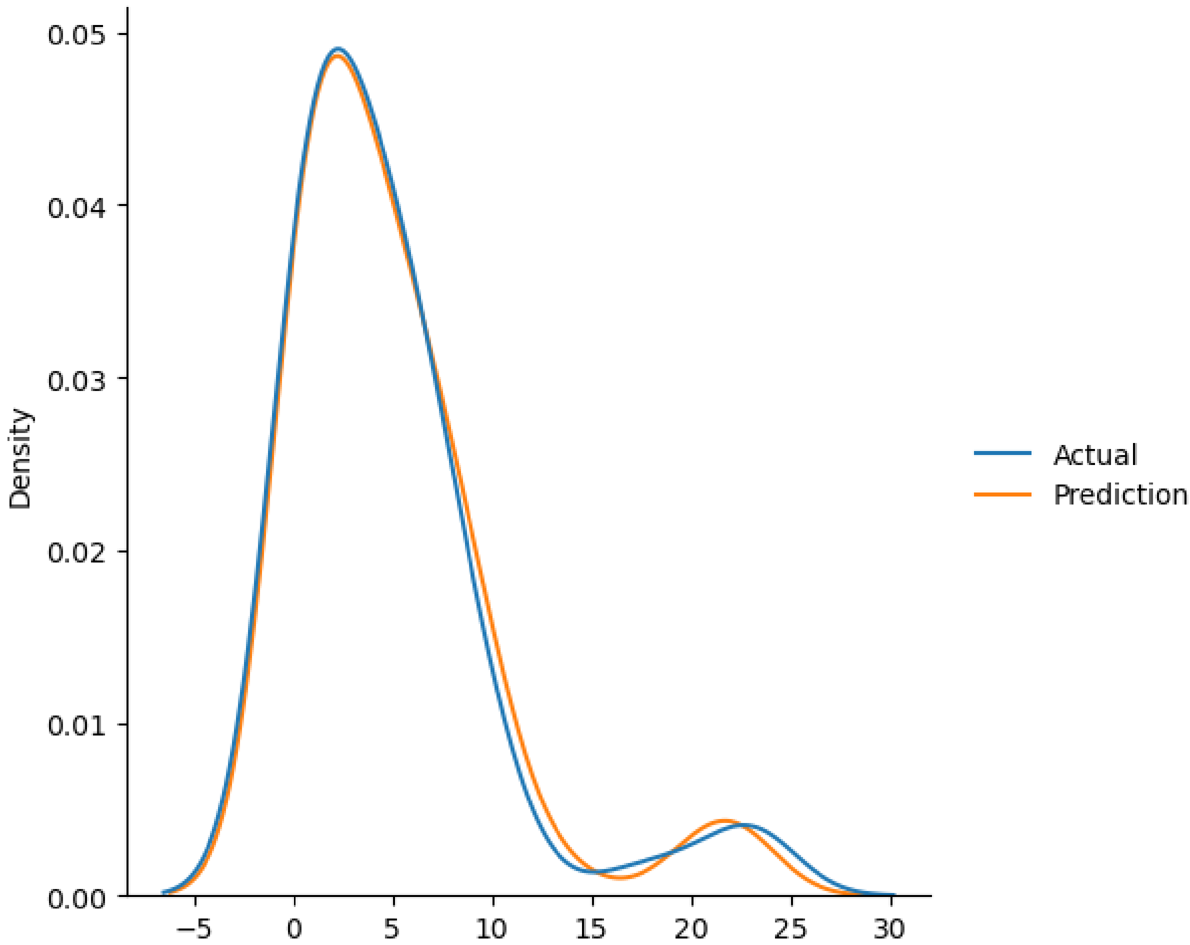
As this model works prety good with default hyperparameter we do not need to tune it

In [44]:
```
dft = pd.DataFrame({'Actual': y_test, 'Prediction': y_pred})
dft.reset_index(drop=True, inplace=True)
dft.head()
```

Out[44]:

	Actual	Prediction
0	0.35	0.424882
1	10.11	10.467838
2	4.95	4.772250
3	0.15	0.268056
4	6.95	7.657061

In [45]:
```
ax = sns.displot(data = (dft['Actual'], dft['Prediction']),kind = 'kde')
```



GBR model performed very good without tuning so we are not going to tune it.

SGDR_model

In [46]:
```
SGDR_model = SGDRegressor(random_state= 42)
SGDR_model.fit(X_train, y_train)
SGDR_y_pred = SGDR_model.predict(X_test)
```

In [47]:
```
SGDR_MAE = mean_absolute_error(y_test, SGDR_y_pred)
SGDR_MSE = mean_squared_error(y_test, SGDR_y_pred)
SGDR_R2 = r2_score(y_test, SGDR_y_pred)

# printing the results
print(f'MAE:{SGDR_MAE}, MSE: {SGDR_MSE}, R2: {SGDR_R2}')
```

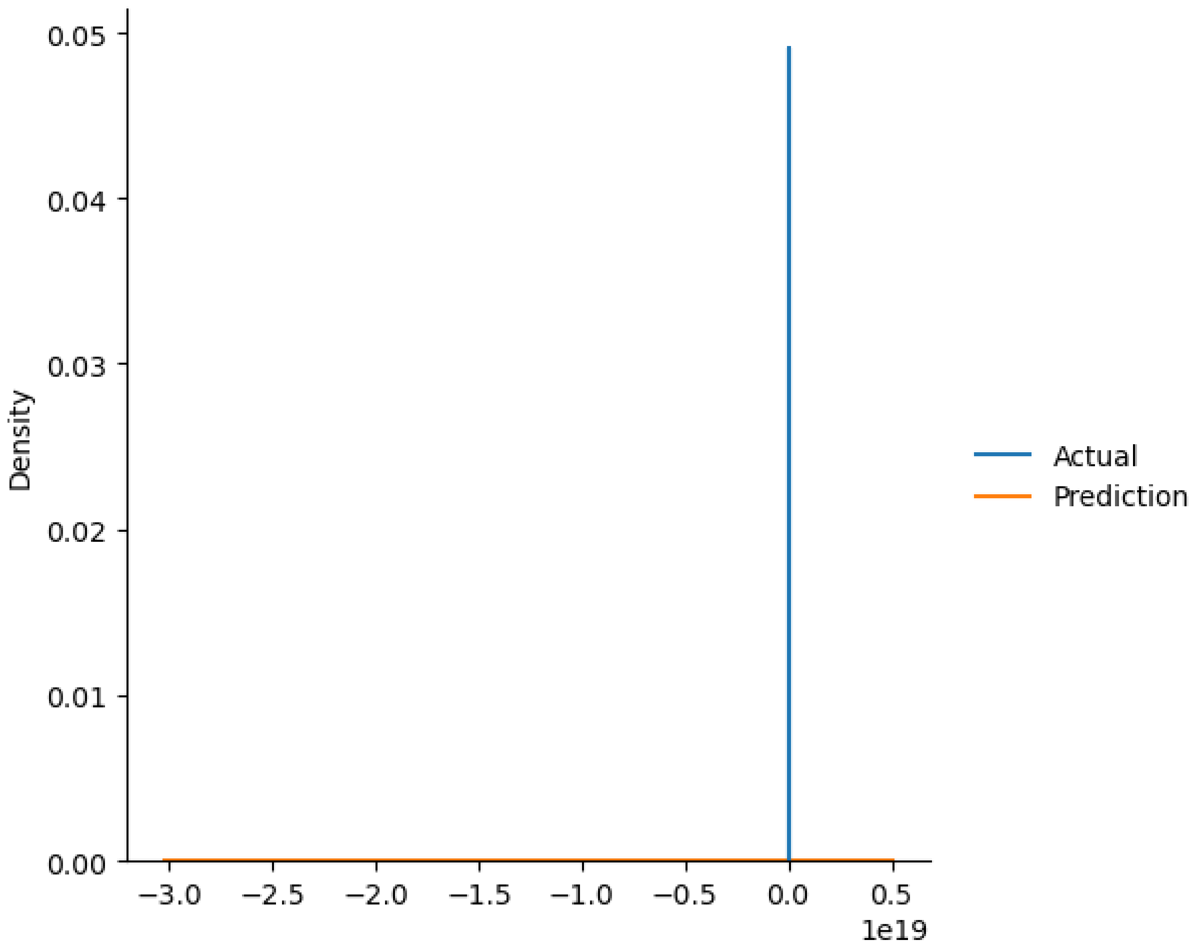
```
MAE:6.4245507559862e+18, MSE: 5.884683481743032e+37, R2: -2.1418863120904382e+3
6
```

In [48]:
```
SGDR_df = pd.DataFrame({'Actual': y_test, 'Prediction': SGDR_y_pred})
SGDR_df.head()
```

Out[48]:

	Actual	Prediction
177	0.35	-4.426537e+18
289	10.11	-2.025141e+18
228	4.95	-1.106634e+19
198	0.15	-6.455367e+18
60	6.95	-7.377746e+18

In [49]:
```
ax = sns.displot(data = (SGDR_df['Actual'], SGDR_df['Prediction']),kind = 'k
```



As we can see it cannot preddict well with default hyperparameter so let's tune it.

In [50]:

```
loss= ['squared_error','huber', 'epsilon_insensitive', 'squared_epsilon_inse
penalty = ['l2', 'l1']
alpha = [float(X) for X in np.linspace(start= 0.0001, stop= 0.01, num = 100
l1_ratio = [float(X) for X in np.linspace(start = 0.1, stop = 0.5, num = 100
fit_intercept = ['True, False']
max_iter = [int(X) for X in np.linspace(start = 900, stop = 1500, num = 100)
learning_rate = ['constant', 'optimal', 'invscaling', 'adaptive']
warm_start = ['False', 'True']
```

In [51]:

```
random_grid = {'loss': loss,
               'penalty': penalty,
               'alpha': alpha,
               'l1_ratio': l1_ratio,
               'max_iter': max_iter,
               'learning_rate': learning_rate,
               'warm_start': warm_start,}
```

In [52]:

```
Tuned_SGDR_model = RandomizedSearchCV(estimator= SGDR_model, param_distribut
```

In [53]:
```
Tuned_SGDR_model.fit(X_train, y_train)
```

Out[53]:
```
RandomizedSearchCV(cv=10, estimator=SGDRegressor(random_state=42),
                   param_distributions={'alpha': [0.0001, 0.0002,
                                                  0.00030000000000000003,
                                                  0.0004, 0.0005,
                                                  0.0006000000000000001,
                                                  0.0007000000000000001, 0.000
8,
                                                  0.0009000000000000001, 0.001,
                                                  0.0011, 0.0012000000000000000
1,
                                                  0.0013000000000000002,
                                                  0.0014000000000000002, 0.001
5,
                                                  0.0016, 0.0017000000000000000
1,
                                                  0.00180000000000...
                                                          'adaptive'],
                                        'loss': ['squared_error', 'huber',
                                                 'epsilon_insensitive',
                                                 'squared_epsilon_insensitiv
e'],
                                        'max_iter': [900, 906, 912, 918, 924,
                                                     930, 936, 942, 948, 954,
                                                     960, 966, 972, 978, 984,
                                                     990, 996, 1003, 1009, 101
5,
                                                     1021, 1027, 1033, 1039,
                                                     1045, 1051, 1057, 1063,
                                                     1069, 1075, ...],
                                        'penalty': ['l2', 'l1'],
                                        'warm_start': ['False', 'True']},
                   random_state=42, scoring='neg_mean_absolute_error')
```

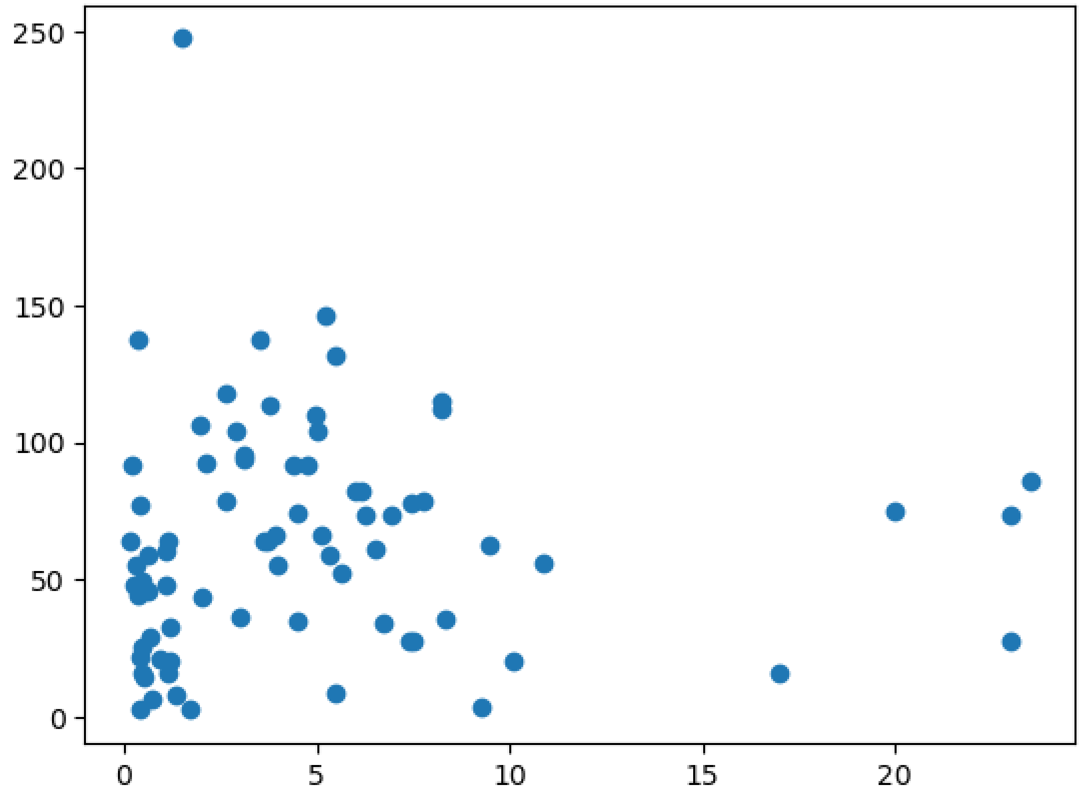
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [54]:
```
Tuned_SGDR_model.best_params_
```

Out[54]:
```
{'warm_start': 'False',
 'penalty': 'l1',
 'max_iter': 1324,
 'loss': 'huber',
 'learning_rate': 'adaptive',
 'l1_ratio': 0.4515151515151515154,
 'alpha': 0.0089}
```

In [55]:
```
T_SGDR_y_pred = Tuned_SGDR_model.predict(X_test)
```

In [57]:
```
plt.scatter(y_test, T_SGDR_y_pred);
```



In [58]:
```
# evaluate model
T_SGDR_MAE = mean_absolute_error(y_test, T_SGDR_y_pred)
T_SGDR_MSE = mean_squared_error(y_test, T_SGDR_y_pred)
T_SGDR_R2 = r2_score(y_test, T_SGDR_y_pred)

# printing the results
print(f'MAE:{T_SGDR_MAE}, MSE: {T_SGDR_MSE}, R2: {T_SGDR_R2}')
```

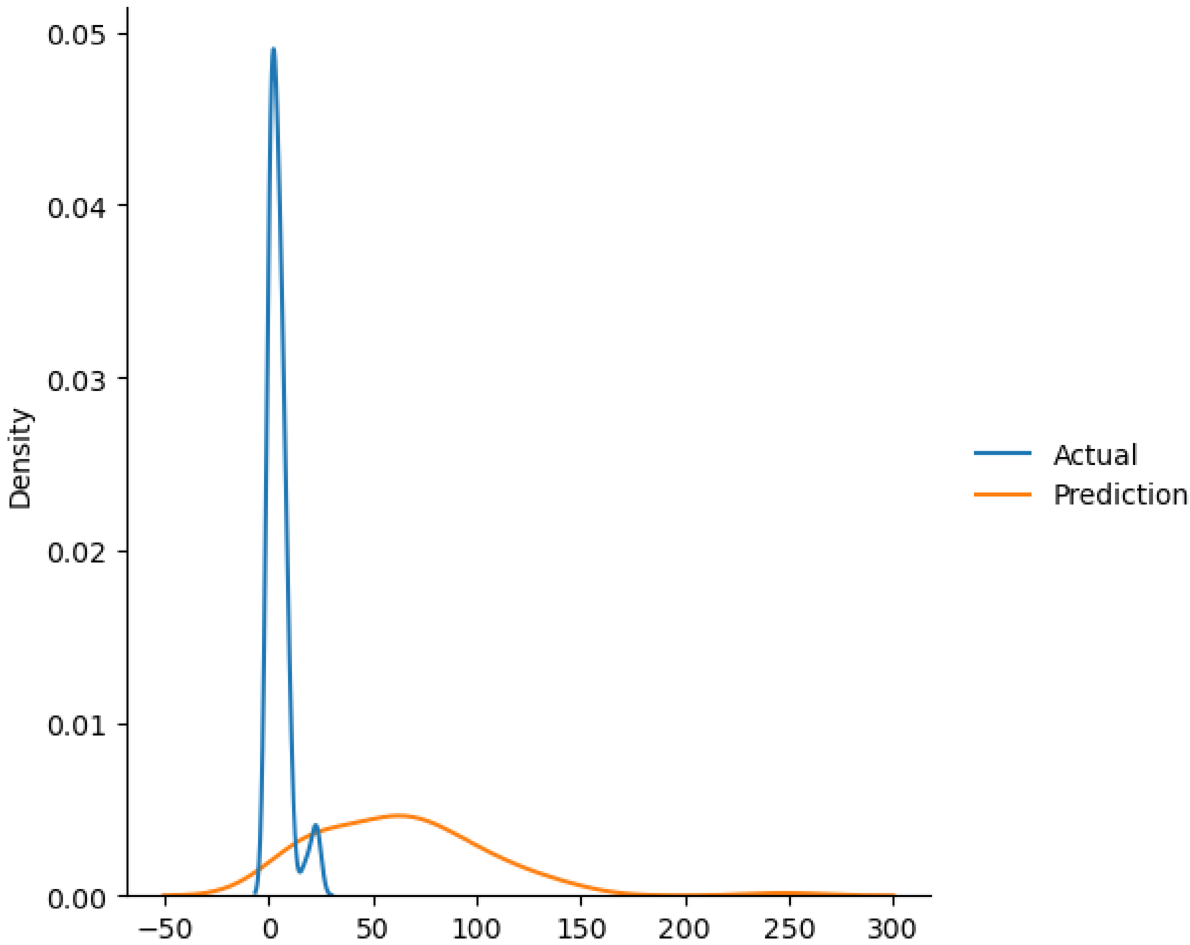
```
MAE:59.07001780969827, MSE: 5217.386128116324, R2: -188.9005777859246
```

In [59]:
```
T_SGDR_df = pd.DataFrame({'Actual': y_test, 'Prediction': T_SGDR_y_pred})
T_SGDR_df.head()
```

Out[59]:

	Actual	Prediction
177	0.35	43.892625
289	10.11	20.069339
228	4.95	109.763463
198	0.15	64.019826
60	6.95	73.170383

In [60]:
```
ax = sns.displot(data = (T_SGDR_df['Actual'], T_SGDR_df['Prediction']),kind
```

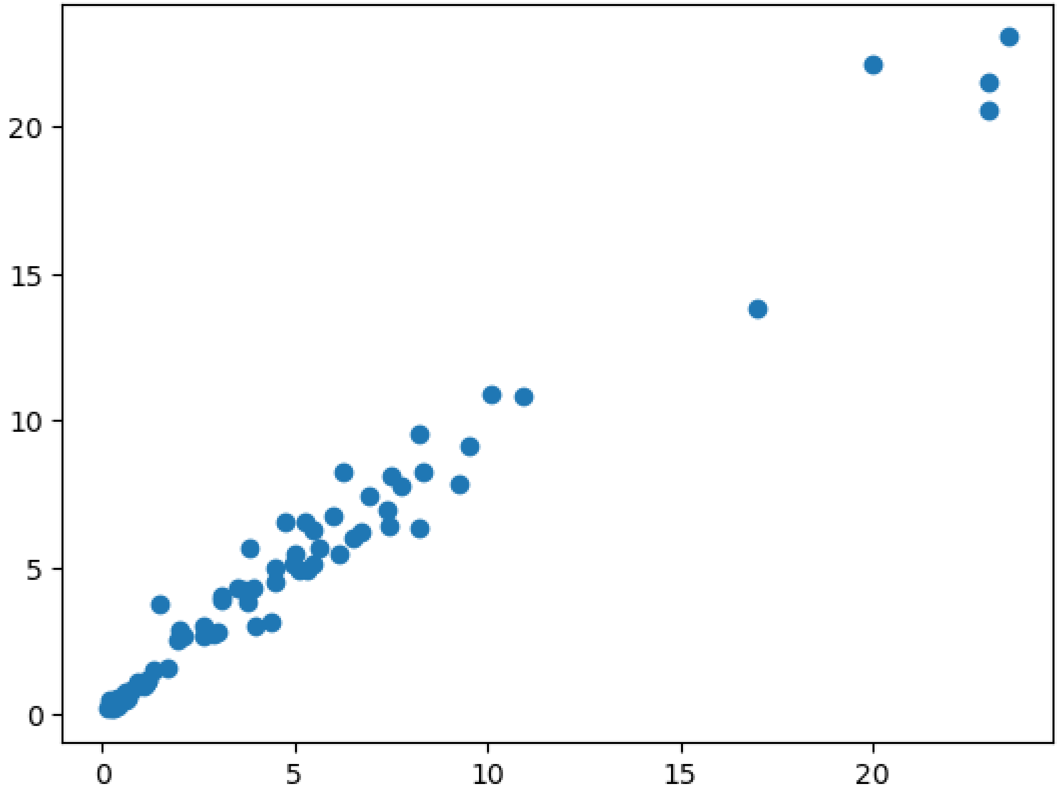


Obviously, the model has not a good performance. Howerver, it is essential to note a substantial difference between the SGDR model after the implementation of tuning techniques as compared to its pre-tuning state.`

RandomForestRegressor

In [61]:
```
RFR_model = RandomForestRegressor()
RFR_model.fit(X_train, y_train)
RFR_y_pred = RFR_model.predict(X_test)
```

In [62]:
```
plt.scatter(y_test, RFR_y_pred);
```



In [63]:
```
RFR_MAE = mean_absolute_error(y_test, RFR_y_pred)
RFR_MSE = mean_squared_error(y_test, RFR_y_pred)
RFR_R2 = r2_score(y_test, RFR_y_pred)

print(f'MAE: {RFR_MAE} | MSE: {RFR_MSE} | R2: {RFR_R2}')
```

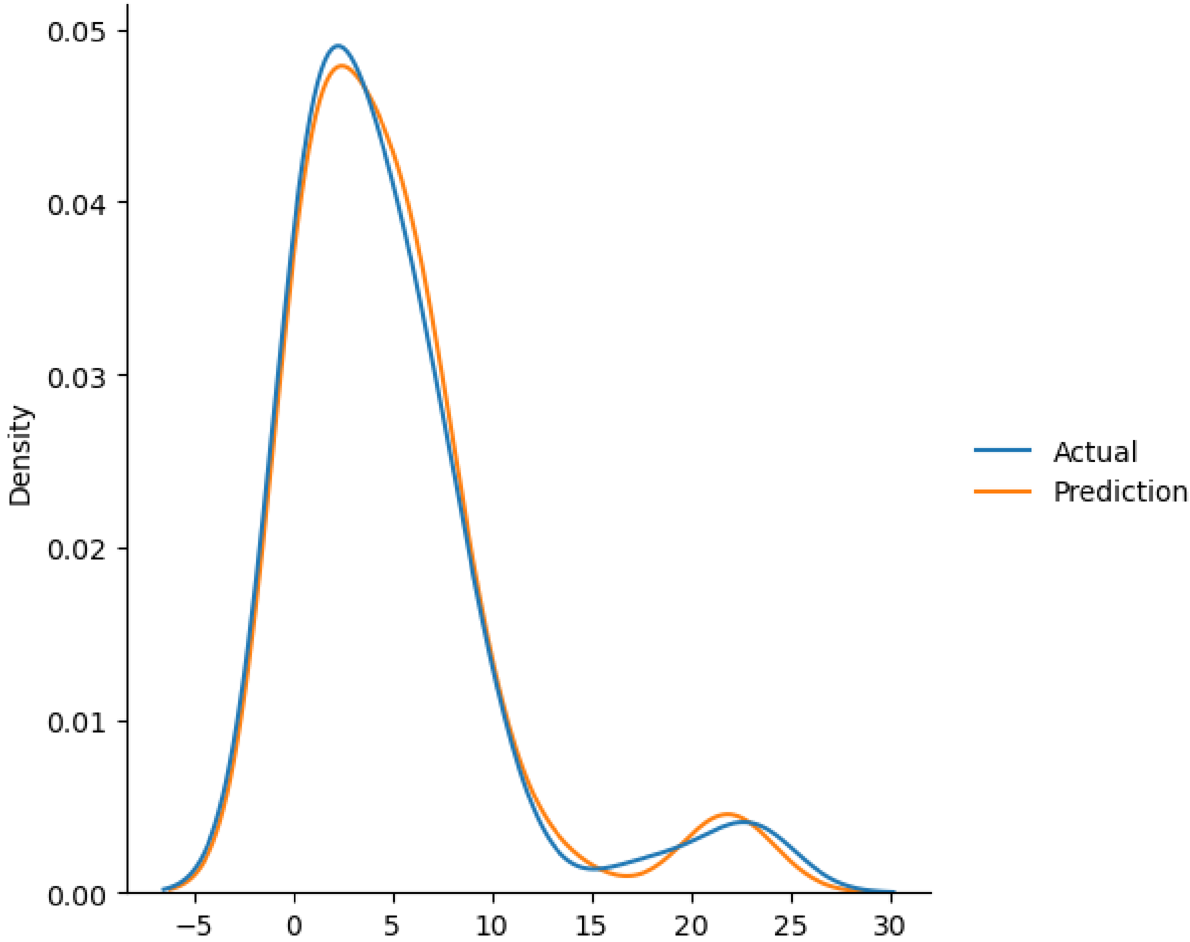
```
MAE: 0.5749657894736846 | MSE: 0.7935562621052629 | R2: 0.9711164194140571
```

In [64]:
```
RFR_df = pd.DataFrame({'Actual': y_test, 'Prediction': RFR_y_pred})
RFR_df.head()
```

Out[64]:

	Actual	Prediction
177	0.35	0.4381
289	10.11	10.8830
228	4.95	5.1575
198	0.15	0.1967
60	6.95	7.4735

In [65]:
```
ax = sns.displot(data = (RFR_df['Actual'], RFR_df['Prediction']), kind = 'kd
```



The random forest regressor exhibits a remarkable level of performance, rendering any further adjustments or tuning is unnecessary.

CatBoostRegressor

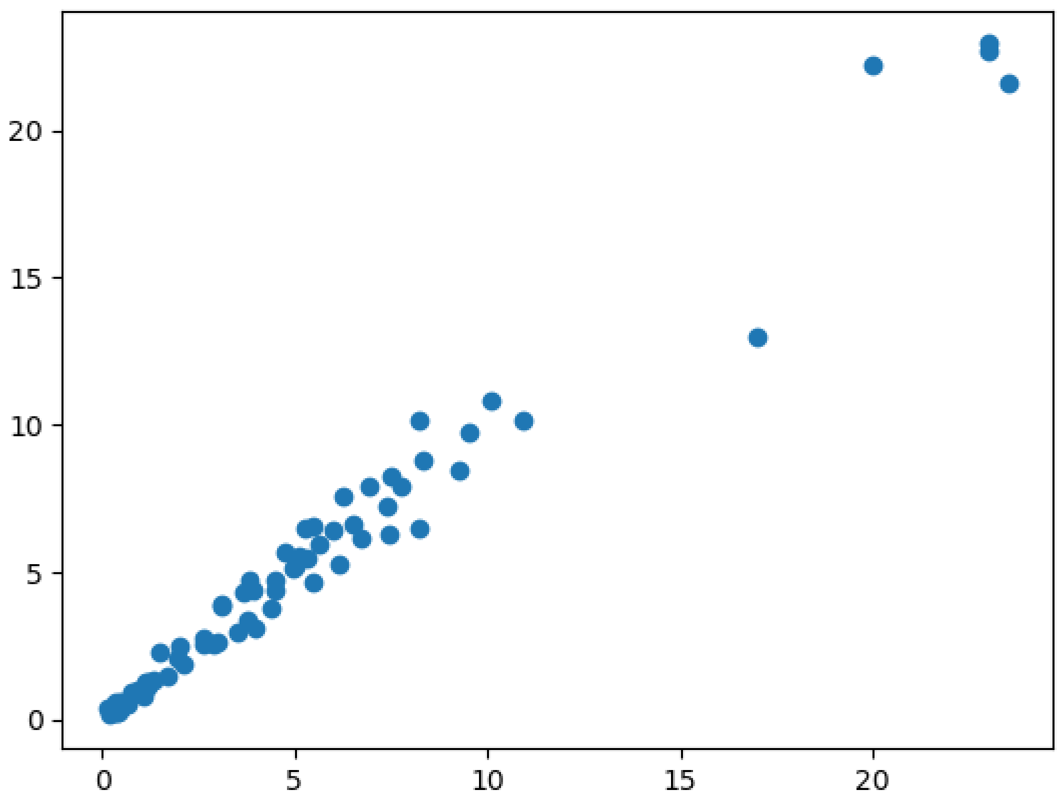
In [66]:

```
CBR_model = CatBoostRegressor()
CBR_model.fit(X_train, y_train)
CBR_y_pred = CBR_model.predict(X_test)
```

```
Learning rate set to 0.032346
0:	learn: 4.9267607	total: 141ms	remaining: 2m 21s
1:	learn: 4.8379801	total: 142ms	remaining: 1m 10s
2:	learn: 4.7517828	total: 143ms	remaining: 47.6s
3:	learn: 4.6718790	total: 144ms	remaining: 35.9s
4:	learn: 4.5993245	total: 145ms	remaining: 28.9s
5:	learn: 4.5244262	total: 146ms	remaining: 24.2s
6:	learn: 4.4477143	total: 147ms	remaining: 20.8s
7:	learn: 4.3766781	total: 148ms	remaining: 18.3s
8:	learn: 4.2933881	total: 149ms	remaining: 16.4s
9:	learn: 4.2372730	total: 149ms	remaining: 14.8s
10:	learn: 4.1637480	total: 150ms	remaining: 13.5s
11:	learn: 4.1025327	total: 151ms	remaining: 12.4s
12:	learn: 4.0415710	total: 152ms	remaining: 11.5s
13:	learn: 3.9772887	total: 153ms	remaining: 10.8s
14:	learn: 3.9159160	total: 154ms	remaining: 10.1s
15:	learn: 3.8678159	total: 156ms	remaining: 9.56s
16:	learn: 3.8074929	total: 157ms	remaining: 9.06s
17:	learn: 3.7543990	total: 158ms	remaining: 8.63s
```

In [67]:

```
plt.scatter(y_test, CBR_y_pred);
```



In [68]:
```
# evaluate model
CBR_MAE = mean_absolute_error(y_test, CBR_y_pred)
CBR_MSE = mean_squared_error(y_test, CBR_y_pred)
CBR_R2 = r2_score(y_test, CBR_y_pred)

# printing the results
print(f'MAE:{CBR_MAE}, MSE: {CBR_MSE}, R2: {CBR_R2}')
```

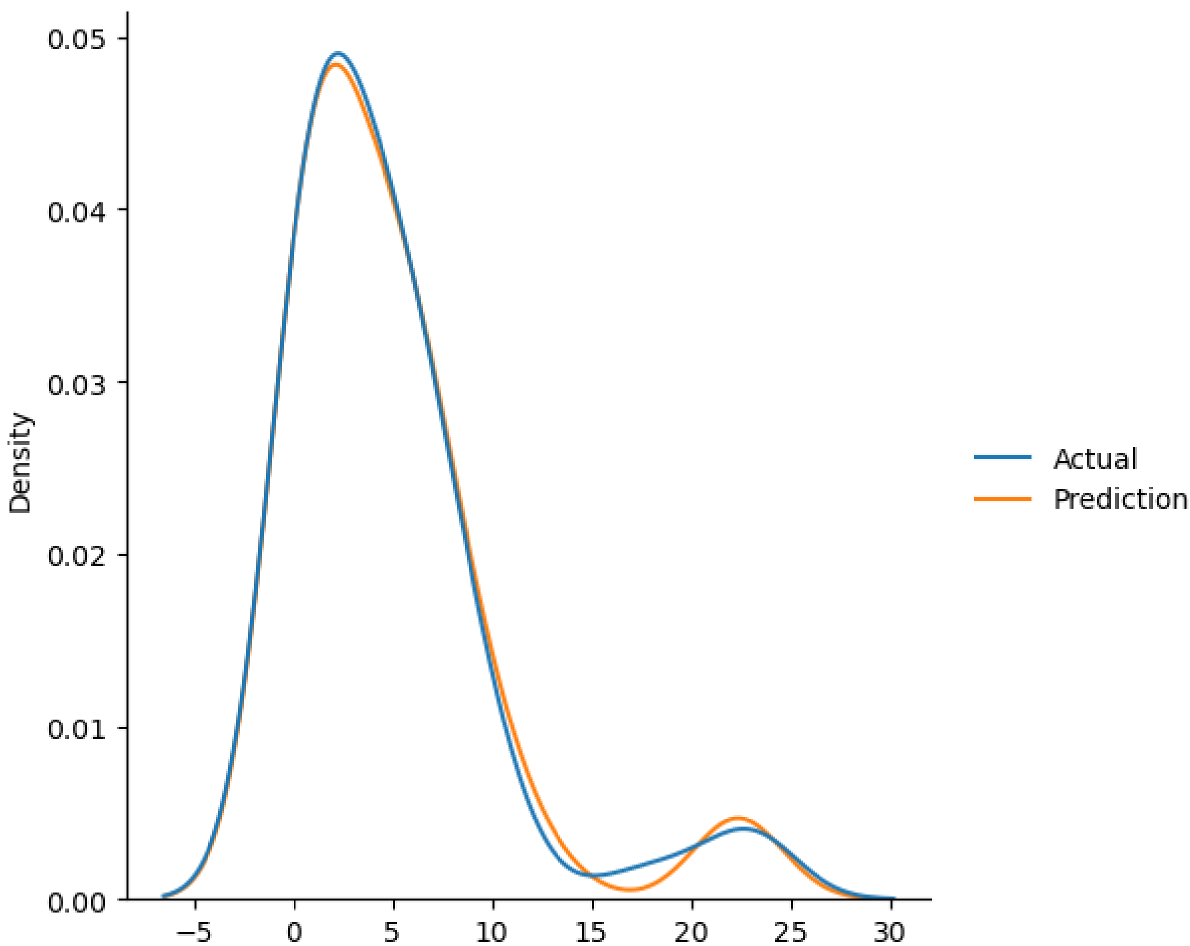
```
MAE:0.5097258370652804, MSE: 0.6627723092087091, R2: 0.9758766475456004
```

In [69]:
```
CBR_df = pd.DataFrame({'Actual': y_test, 'Prediction': CBR_y_pred})
CBR_df.head()
```

Out[69]:

	Actual	Prediction
177	0.35	0.592639
289	10.11	10.833225
228	4.95	5.120798
198	0.15	0.342871
60	6.95	7.926235

In [70]:
```
ax = sns.displot(data = (CBR_df['Actual'], CBR_df['Prediction']),kind = 'kde
```



Catboost regressor also is doing well so tuning is unnecessary

In [72]:
```
print(f'GBR_MAE:{round(GBR_MAE, 3)}, GBR_MSE: {round(GBR_MSE, 3)}, GBR_R2: {
print('-----------------------------------')
print(f'SGDR_MAE:{round(SGDR_MAE, 3)}, SGDR_MSE: {round(SGDR_MSE, 3)}, SGDR_
print('-----------------------------------')
print(f'Tuned_SGDR_MAE:{round(T_SGDR_MAE, 3)}, Tuned_SGDR_MSE: {round(T_SGDR_
print('-----------------------------------')
print(f'RFR_MAE: {round(RFR_MAE, 3)} | RFR_MSE: {round(RFR_MSE, 3)} | RFR_R2
print('-----------------------------------')
print(f'CBR_MAE:{round(CBR_MAE, 3)}, CBR_MSE: {round(CBR_MSE, 3)}, CBR_R2: {
```

```
GBR_MAE:0.541, GBR_MSE: 0.881, GBR_R2: 0.968
-----------------------------------
SGDR_MAE:6.424550755986201e+18, SGDR_MSE: 5.884683481743032e+37, SGDR_R2: -2.14
18863120904382e+36
-----------------------------------
Tuned_SGDR_MAE:59.07, Tuned_SGDR_MSE: 5217.386, Tuned_SGDR_R2: -188.901
-----------------------------------
RFR_MAE: 0.575 | RFR_MSE: 0.794 | RFR_R2: 0.971
-----------------------------------
CBR_MAE:0.51, CBR_MSE: 0.663, CBR_R2: 0.976
```

Conclusion

Based on our exploratory data analysis, it was determined that the selling price of cars is predominantly influenced by factors such as the current price, fuel type, and seller type. Specifically, a higher current price tends to correspond to a higher selling price. Moreover, vehicles with diesel fuel type tend to exhibit higher price tags, while cars sold by dealers tend to have inflated prices, potentially attributable to profit margins.

In the realm of machine learning models, we conducted a comprehensive comparison analysis involving four specific regressor models: Gradient Boosting Regressor, SGD Regressor, Random Forest Regressor, and CatBoost Regressor. Subsequent evaluation revealed that the CatBoost Regressor exhibited the highest accuracy score, garnering an impressive performance level of 97.6%.