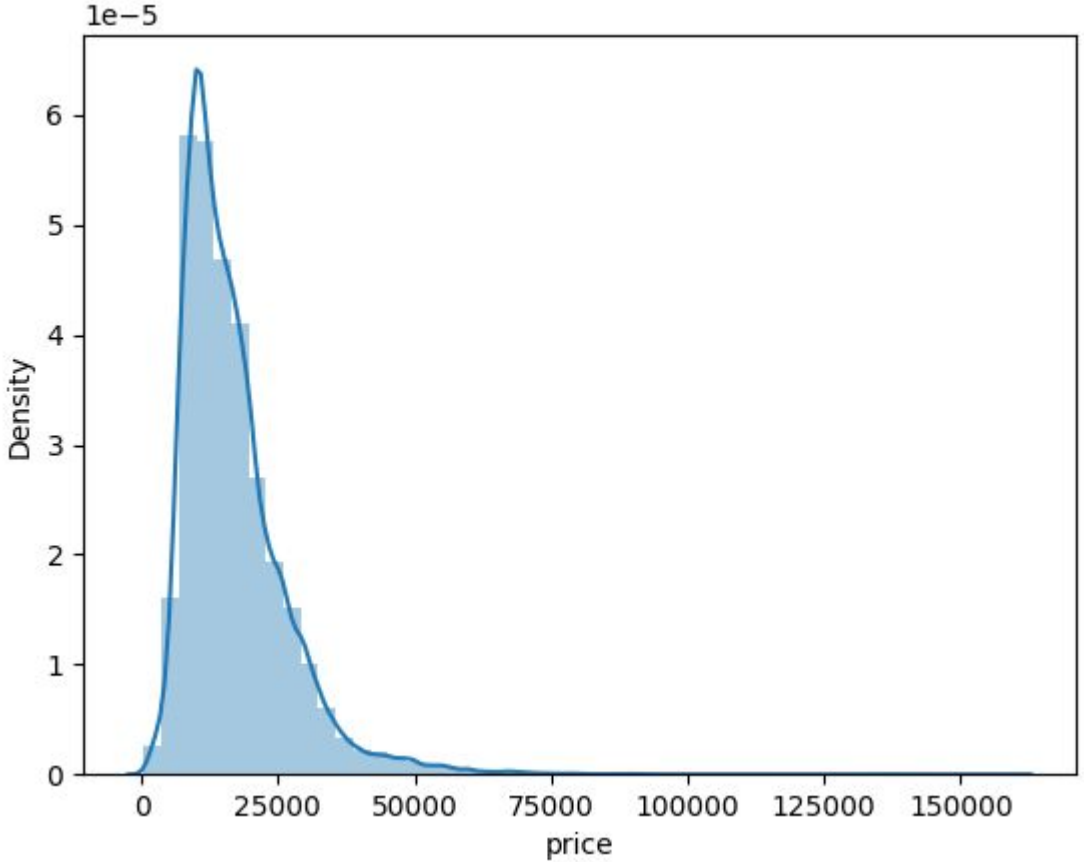
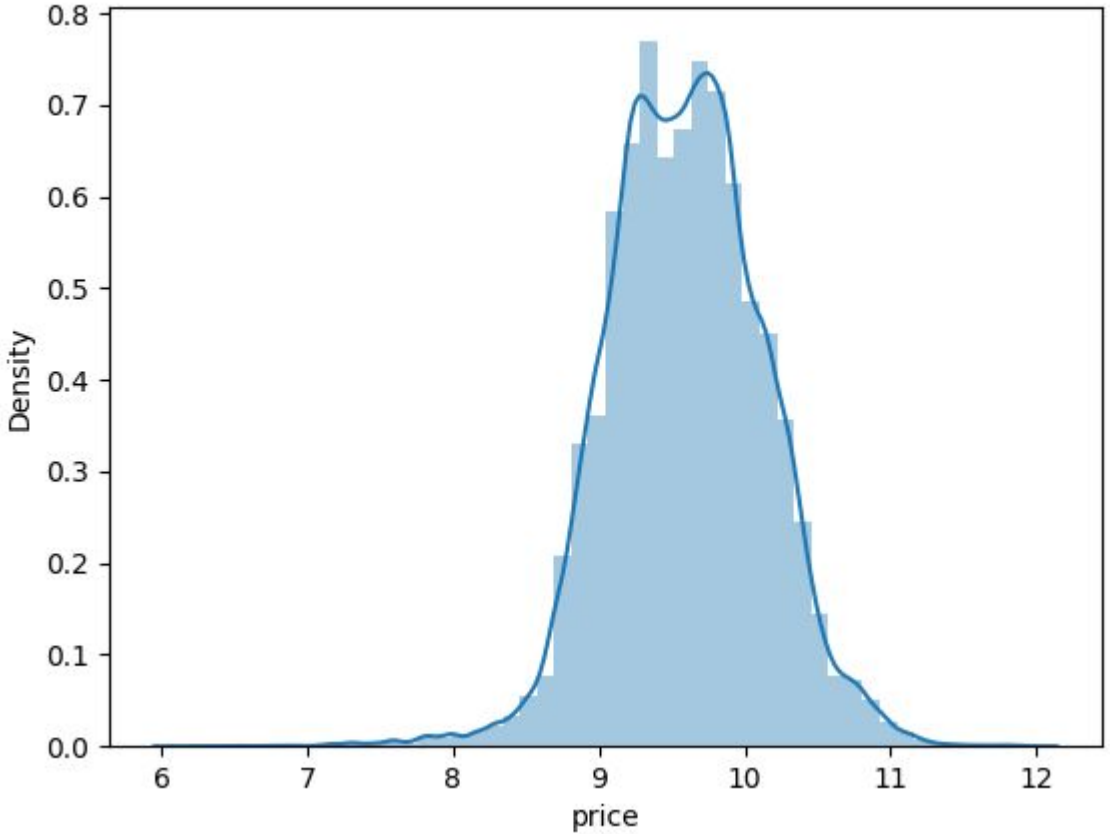
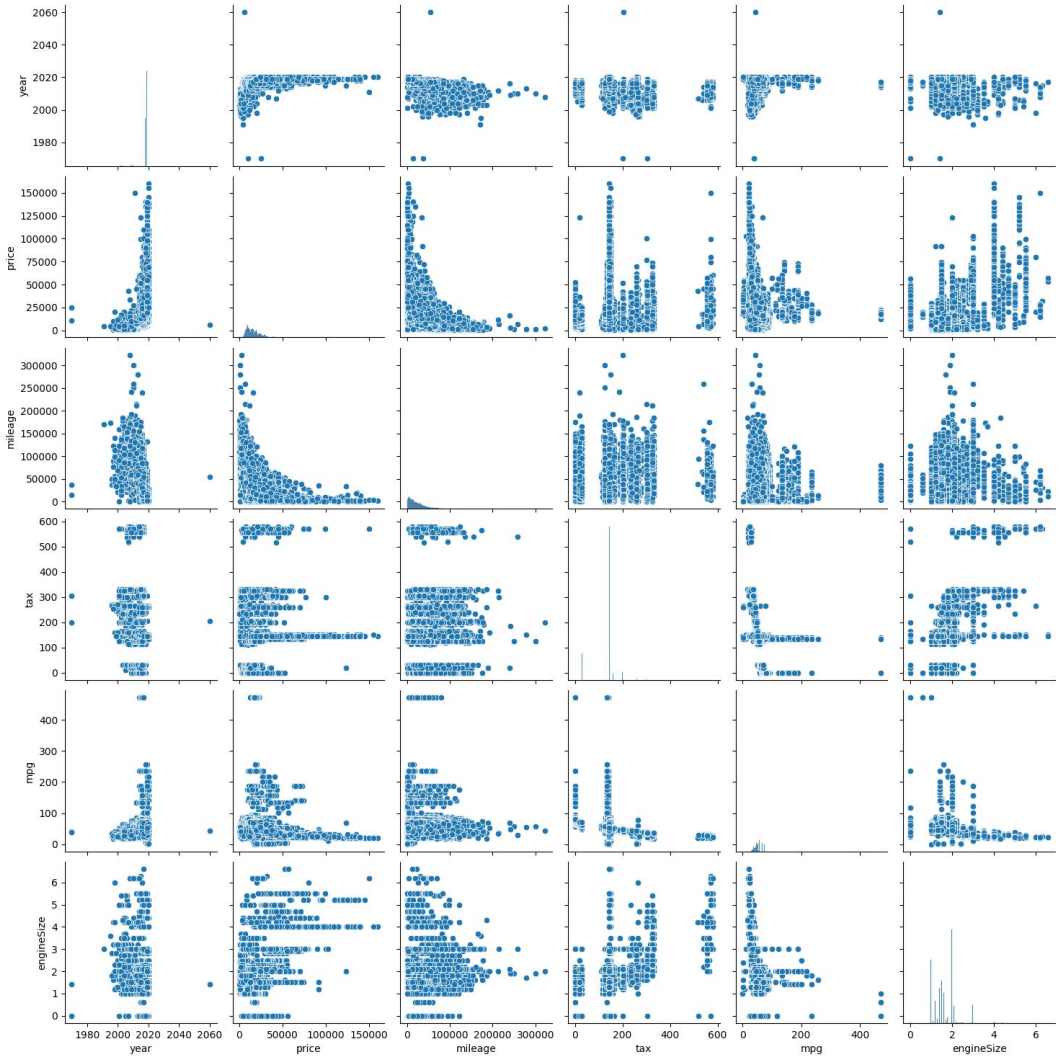
# Proyecto 3

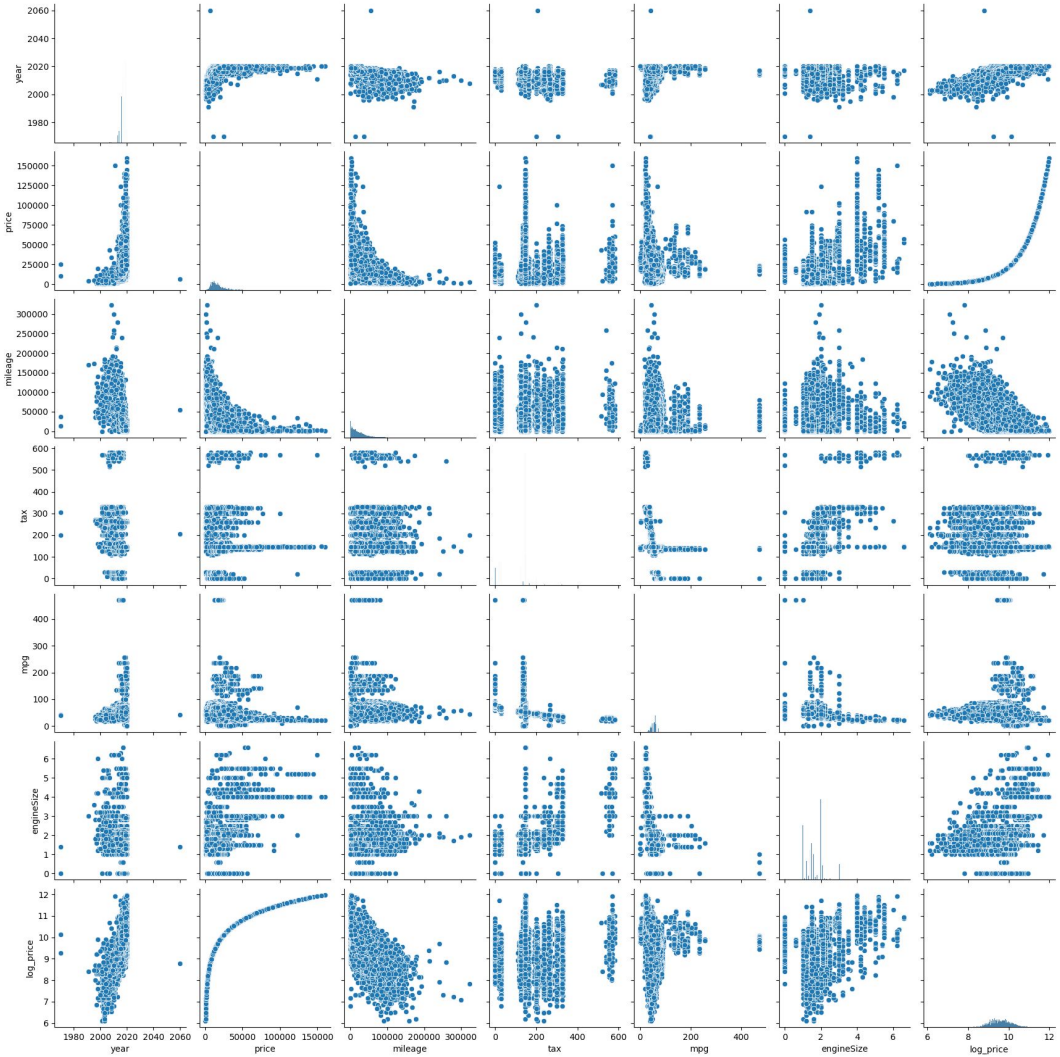
# Data

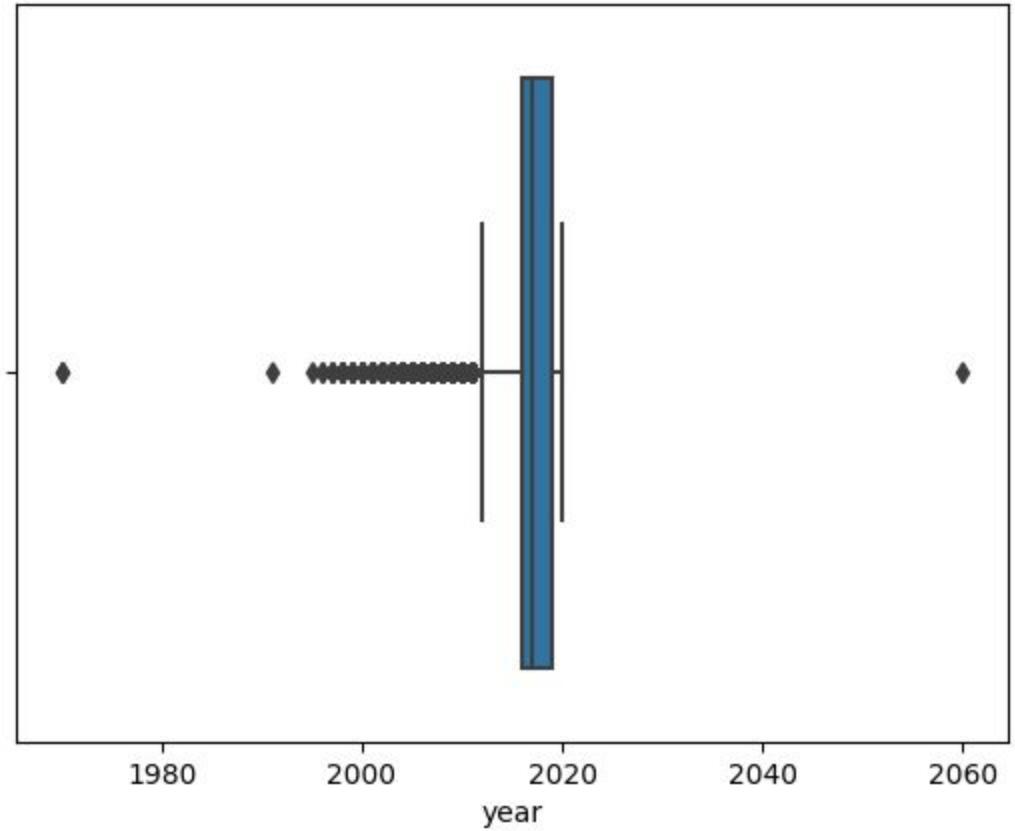
| | model | year | price | transmission | mileage | fuelType | tax | mpg | engineSize | make |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Corsa | 2018 | 7885 | Manual | 9876 | Petrol | 145.0 | 55.4 | 1.4 | vauxhall |
| 1 | Corsa | 2019 | 11995 | Manual | 2500 | Petrol | 145.0 | 54.3 | 1.4 | vauxhall |
| 2 | Corsa | 2017 | 9777 | Automatic | 9625 | Petrol | 145.0 | 47.9 | 1.4 | vauxhall |
| 3 | Corsa | 2016 | 8500 | Manual | 25796 | Petrol | 30.0 | 55.4 | 1.4 | vauxhall |
| 4 | Corsa | 2019 | 10000 | Manual | 3887 | Petrol | 145.0 | 43.5 | 1.4 | vauxhall |

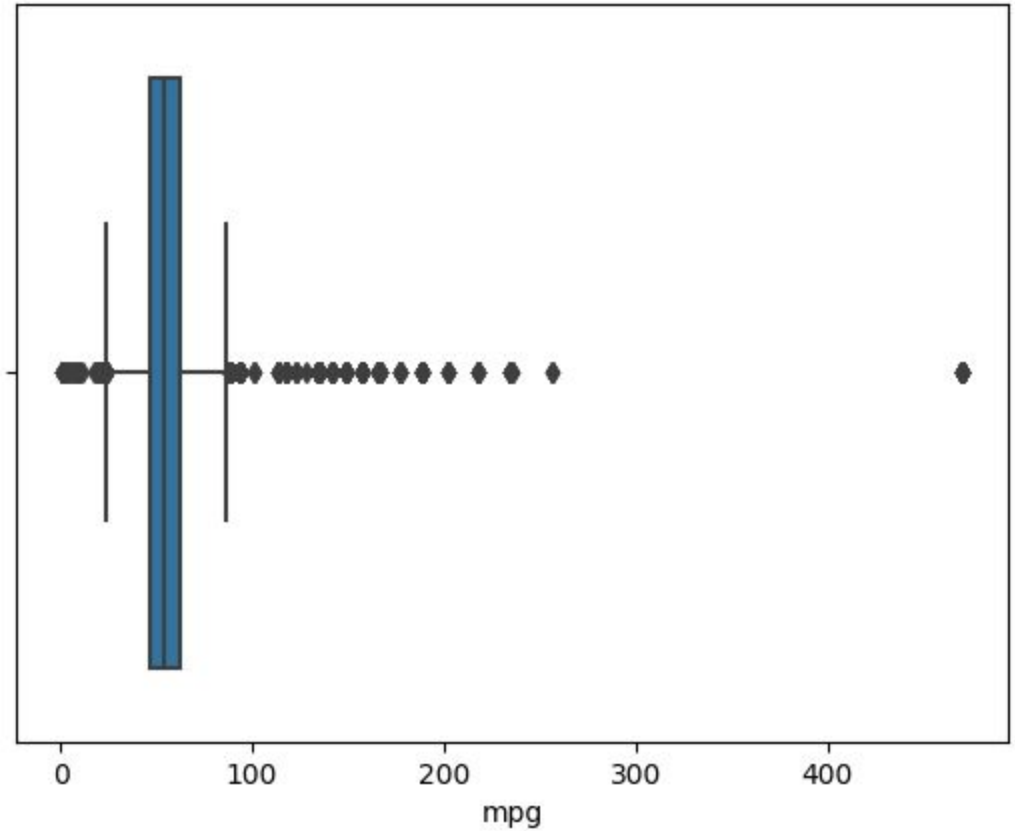

1e−5
6
5
4
3
2
1
0
Density
0
25000
50000
75000
100000
125000
150000
price


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Density
6
7
8
9
10
11
12
price


year
price
mileage
tax
mpg
engineSize
2060
2040
2020
2000
1980
150000
125000
100000
75000
50000
25000
0
300000
250000
200000
150000
100000
50000
600
500
400
300
200
100
0
400
300
200
100
0
6
5
4
3
2
1
0
1980 2000 2020 2040 2060
0 50000 100000 150000
0 100000 200000 300000
0 200 400 600
0 200 400
0 2 4 6


year
price
mileage
tax
mpg
engineSize
log_price


1980
2000
2020
2040
2060
year


0
100
200
300
400
mpg

# Missing values

```
nulls = raw_data[raw_data['tax'].isna()==True]
nulls.select_dtypes(exclude='number')['model'].unique()
```

14] ✓ 0.1s

```
array(['Focus', 'C Class'], dtype=object)
```

```
c_class = raw_data.query('model == "C Class"')
c_class.describe()
```

1] ✓ 0.9s

| | year | price | mileage | tax | mpg | engineSize |
|---|---|---|---|---|---|---|
| count | 7646.000000 | 7646.00000 | 7646.000000 | 3747.000000 | 3747.000000 | 7646.000000 |
| mean | 2017.367251 | 23684.84057 | 22279.462072 | 118.405391 | 56.883133 | 2.033992 |
| std | 2.117069 | 8864.65286 | 22481.301121 | 63.088864 | 16.371416 | 0.482721 |
| min | 1991.000000 | 1290.00000 | 1.000000 | 0.000000 | 23.000000 | 0.000000 |
| 25% | 2016.000000 | 17699.00000 | 6001.000000 | 125.000000 | 46.300000 | 2.000000 |
| 50% | 2018.000000 | 22995.00000 | 14584.500000 | 145.000000 | 57.700000 | 2.000000 |
| 75% | 2019.000000 | 28896.00000 | 32038.750000 | 145.000000 | 64.200000 | 2.100000 |
| max | 2020.000000 | 88995.00000 | 173000.000000 | 580.000000 | 217.300000 | 6.200000 |

```
focus = raw_data.query('model == "Focus"')
focus.describe()
```

8] ✓ 0.5s

| | year | price | mileage | tax | mpg | engineSize |
|---|---|---|---|---|---|---|
| count | 10042.000000 | 10042.000000 | 10042.000000 | 4588.000000 | 4588.000000 | 10042.000000 |
| mean | 2017.060347 | 13400.999004 | 23268.902709 | 111.156277 | 60.089385 | 1.362418 |
| std | 2.138223 | 4638.690036 | 20557.013307 | 63.634199 | 10.733369 | 0.398380 |
| min | 2002.000000 | 495.000000 | 1.000000 | 0.000000 | 26.300000 | 0.000000 |
| 25% | 2016.000000 | 10295.500000 | 9381.000000 | 20.000000 | 55.400000 | 1.000000 |
| 50% | 2017.000000 | 13000.000000 | 16548.000000 | 145.000000 | 60.100000 | 1.500000 |
| 75% | 2019.000000 | 16799.000000 | 30689.250000 | 145.000000 | 67.300000 | 1.600000 |
| max | 2020.000000 | 54995.000000 | 177644.000000 | 330.000000 | 83.100000 | 2.500000 |

```
class ByCategoryImputer(BaseEstimator, TransformerMixin):
    """
    Imputes empty values based on category of observation.

    Parameters
    ----------
    category_col: str
        column on which to find categories by which to perform imputation
    target_cols: list[str]
        list of columns which contain empty values to be imputed
    strategy: str
        imputation strategy (mean, median, mode)

    Returns
    ----------
    X: array-like
        array with imputed values in the target column
    """
```

# Pipeline Model

```
numeric_pipeline = Pipeline(steps=[
    ('scale', StandardScaler())
])

categorical_pipeline = Pipeline(steps=[
    ('one-hot', OneHotEncoder(handle_unknown='ignore',sparse=False))
])
```

```
full_processor = ColumnTransformer(
        transformers=[('number', numeric_pipeline, numerical_features),
                      ('category', categorical_pipeline, categorical_features)]
)
```

```
ridge_pipeline = Pipeline(steps=[
    ('impute', ByCategoryImputer( category_col='model',target_cols=['mpg'],strategy='mean')),
    ('preprocess', full_processor),
    ('model', Ridge())
])

param_grid = {
    'model__alpha': [1e-3, 1e-2, 1e-1, 1e0]
}
```

```
grid_search = GridSearchCV(
    ridge_pipeline,
    param_grid,
    cv=5,
    scoring='r2'
    )
```

# Results

```
grid_search.best_score_
✓ 0.2s
0.9372948590120899
```

```
grid_search.best_params_
✓ 0.3s
{'model__alpha': 0.1}
```

```
final_pipeline.score(X_test,y_test)
```

✓ 0.1s

```
0.9395486968379899
```