In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects
import matplotlib.pylab as pl
import matplotlib.gridspec as gridspec
import seaborn as sns
import warnings
%matplotlib inline
sns.set(style='ticks', font_scale=1.2)

warnings.filterwarnings('ignore')
```

In [2]:
```python
def conti_var_summary(x):
    '''
    UDF for getting customised summary for continuous variables
    '''
    # freq and missings
    n_total = x.shape[0]
    n_miss = x.isna().sum()
    perc_miss = n_miss * 100 / n_total

    # outliers - iqr
    q1 = x.quantile(0.25)
    q3 = x.quantile(0.75)
    iqr = q3 - q1
    lc_iqr = q1 - 1.5 * iqr
    uc_iqr = q3 + 1.5 * iqr

    return pd.Series([
        x.dtype,
        x.nunique(), n_total,
        x.count(), n_miss, perc_miss,
        x.sum(),
        x.mean(),
        x.std(),
        x.var(), lc_iqr, uc_iqr,
        x.min(),
        x.quantile(0.01),
        x.quantile(0.05),
        x.quantile(0.10),
        x.quantile(0.25),
        x.quantile(0.5),
        x.quantile(0.75),
        x.quantile(0.90),
        x.quantile(0.95),
        x.quantile(0.99),
        x.max()
    ],
        index=[
        'dtype', 'cardinality', 'n_tot', 'n', 'nmiss',
        'perc_miss', 'sum', 'mean', 'std', 'var', 'lc_iqr',
        'uc_iqr', 'min', 'p1', 'p5', 'p10', 'p25', 'p50',
                         'p75', 'p90', 'p95', 'p99', 'max'
    ])
```

In [3]:
```python
def cat_var_summary(x):
    '''
    UDF for getting customised summary for categorical variables
    '''
    Mode = x.value_counts().sort_values(ascending=False)[0:1].reset_index()
    return pd.Series([
        x.count(),
        x.nunique(),
        x.isnull().sum(), Mode.iloc[0, 0], Mode.iloc[0, 1],
        round(Mode.iloc[0, 1] * 100 / x.count(), 2)
    ],
                    index=['N', 'CARDINALITY', 'NMISS', 'MODE', 'FREQ', 'PERCENT'])
```

In [4]:
```python
bikes=pd.read_csv("bikes.csv")
```

In [5]:
```python
bikes
```

Out[5]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price
0	Bajaj Avenger Cruise 220 2017	2017	17000 Km	first owner	hyderabad	\n\n 35 kmpl	19 bhp	63500
1	Royal Enfield Classic 350cc 2016	2016	50000 Km	first owner	hyderabad	\n\n 35 kmpl	19.80 bhp	115000
2	Hyosung GT250R 2012	2012	14795 Km	first owner	hyderabad	\n\n 30 kmpl	28 bhp	300000
3	Bajaj Dominar 400 ABS 2017	2017	Mileage 28 Kms	first owner	pondicherry	\n\n 28 Kms	34.50 bhp	100000
4	Jawa Perak 330cc 2020	2020	2000 Km	first owner	bangalore	\n\n	30 bhp	197500
...	...	...	...	...	...	...	...	...
7852	Yamaha YZF-R15 150cc 2011	2011	7000 Km	first owner	agra	\n\n 42 kmpl	16 bhp	55000
7853	Bajaj Discover 100cc 2015	2015	Mileage 80 Kmpl	first owner	delhi	\n\n 80 Kmpl	7.7	28000
7854	Bajaj Pulsar 180cc 2016	2016	6407 Km	first owner	bangalore	\n\n 65 kmpl	17 bhp	61740
7855	Bajaj V15 150cc 2016	2016	7524 Km	first owner	bangalore	\n\n 57 kmpl	11.80 bhp	49000
7856	Bajaj Pulsar 220cc 2016	2016	15000 Km	first owner	chennai	\n\n 38 kmpl	21 bhp	65000

7857 rows × 8 columns

In [6]:
```
bikes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7857 entries, 0 to 7856
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   model_name  7857 non-null   object
 1   model_year  7857 non-null   int64 
 2   kms_driven  7857 non-null   object
 3   owner       7857 non-null   object
 4   location    7838 non-null   object
 5   mileage     7846 non-null   object
 6   power       7826 non-null   object
 7   price       7857 non-null   int64 
dtypes: int64(2), object(6)
memory usage: 491.2+ KB
```

In [7]:
```
bikes.dtypes
```

Out[7]:
```
model_name    object
model_year     int64
kms_driven    object
owner         object
location      object
mileage       object
power         object
price          int64
dtype: object
```

In [8]:
```
bikes.head()
```

Out[8]:

	model_name	model_year	kms_driven	owner	location	mileage	power	price
0	Bajaj Avenger Cruise 220 2017	2017	17000 Km	first owner	hyderabad	\n\n 35 kmpl	19 bhp	63500
1	Royal Enfield Classic 350cc 2016	2016	50000 Km	first owner	hyderabad	\n\n 35 kmpl	19.80 bhp	115000
2	Hyosung GT250R 2012	2012	14795 Km	first owner	hyderabad	\n\n 30 kmpl	28 bhp	300000
3	Bajaj Dominar 400 ABS 2017	2017	Mileage 28 Kms	first owner	pondicherry	\n\n 28 Kms	34.50 bhp	100000
4	Jawa Perak 330cc 2020	2020	2000 Km	first owner	bangalore	\n\n	30 bhp	197500

In [9]:
```
bikes.shape
```

Out[9]: (7857, 8)

In [10]:
```
###Dropping Duplicates
bikes.drop_duplicates(inplace=True)
bikes.shape
```

Out[10]: (7857, 8)

In [11]:
```
##Continuous variables :
bikes.select_dtypes(['int64','float64']).apply(conti_var_summary)
```

Out[11]:

	model_year	price
dtype	int64	int64
cardinality	36	1627
n_tot	7857	7857
n	7857	7857
nmiss	0	0
perc_miss	0.0	0.0
sum	15834744	839059534
mean	2015.367698	106791.336897
std	4.001443	138926.124628
var	16.011548	19300468104.087734
lc_iqr	2008.0	-82500.0
uc_iqr	2024.0	249500.0
min	1950	0
p1	2003.0	10500.0
p5	2009.0	18500.0
p10	2011.0	25000.0
p25	2014.0	42000.0
p50	2016.0	75000.0
p75	2018.0	125000.0
p90	2019.0	180000.0
p95	2020.0	270000.0
p99	2021.0	778950.0
max	2021	3000000

In [14]:

```
##Categorical variables :
bikes.select_dtypes(['object']).apply(cat_var_summary)
```

Out[14]:

	model_name	kms_driven	owner	location	mileage	power
N	7857	7857	7857	7838	7846	7826
CARDINALITY	1724	1801	4	561	117	272
NMISS	0	0	0	19	11	31
MODE	Royal Enfield Classic 350cc 2017	Mileage 65 Kmpl	first owner	delhi	\n\n 35 kmpl	19.80 bhp
FREQ	78	436	6817	1438	1071	922
PERCENT	0.99	5.55	86.76	18.35	13.65	11.78

In [21]:

```
import pandas as pd
import numpy as np

# Example cleaning process
bikes['price'] = pd.to_numeric(bikes['price'], errors='coerce')
bikes['mileage'] = bikes['mileage'].str.extract('(\d+\.?\d*)').astype(float)

# Remove rows with price <= 0
bikes.drop(bikes[bikes['price'] <= 0].index, inplace=True)

# Replace 0 mileage with NaN
bikes['mileage'].replace(0, np.nan, inplace=True)

# Impute missing mileage by median within grouped features
bikes['mileage'] = bikes.groupby(['model_name', 'price', 'power'])['mileage'].transform(lambda x: x.fillna(x.med

print(bikes.shape)
```

```
(7826, 8)
```

In [22]:

```
##Skipping the outlier treatment for now. It will be done in EDA section if required for the Analysis.

#Missing Values
```

In [24]:

```
pd.DataFrame([bikes.isnull().sum(), bikes.isnull().sum() / bikes.shape[0] * 100], index=['count', '%']).T
```

Out[24]:

	count	%
model_name	0.0	0.000000
model_year	0.0	0.000000
kms_driven	0.0	0.000000
owner	0.0	0.000000
location	19.0	0.242780
mileage	776.0	9.915666
power	31.0	0.396116
price	0.0	0.000000

In [25]:
```
####Treating the NaNs as :

#Filling values
#drive - mode()
#endV - median() based on car, body, engType and drive
#Dropping Rows
#mileage, engV - Whatever left as NaNs
```

In [27]:
```
print(bikes.columns)
```

```
Index(['model_name', 'model_year', 'kms_driven', 'owner', 'location',
       'mileage', 'power', 'price'],
      dtype='object')
```

In [28]:
```
import numpy as np

# Clean numeric columns first
bikes['mileage'] = bikes['mileage'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
bikes['power'] = bikes['power'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
bikes['price'] = pd.to_numeric(bikes['price'], errors='coerce')

# Replace 0 or invalid mileage with NaN
bikes['mileage'].replace(0, np.nan, inplace=True)

# Fill missing mileage using group-wise median
bikes['mileage'] = bikes.groupby(['model_name', 'model_year', 'power'])['mileage'].transform(lambda x: x.fillna
```

In [29]:
```
# Dropping Rows
bikes.dropna(inplace=True)
bikes.shape
```

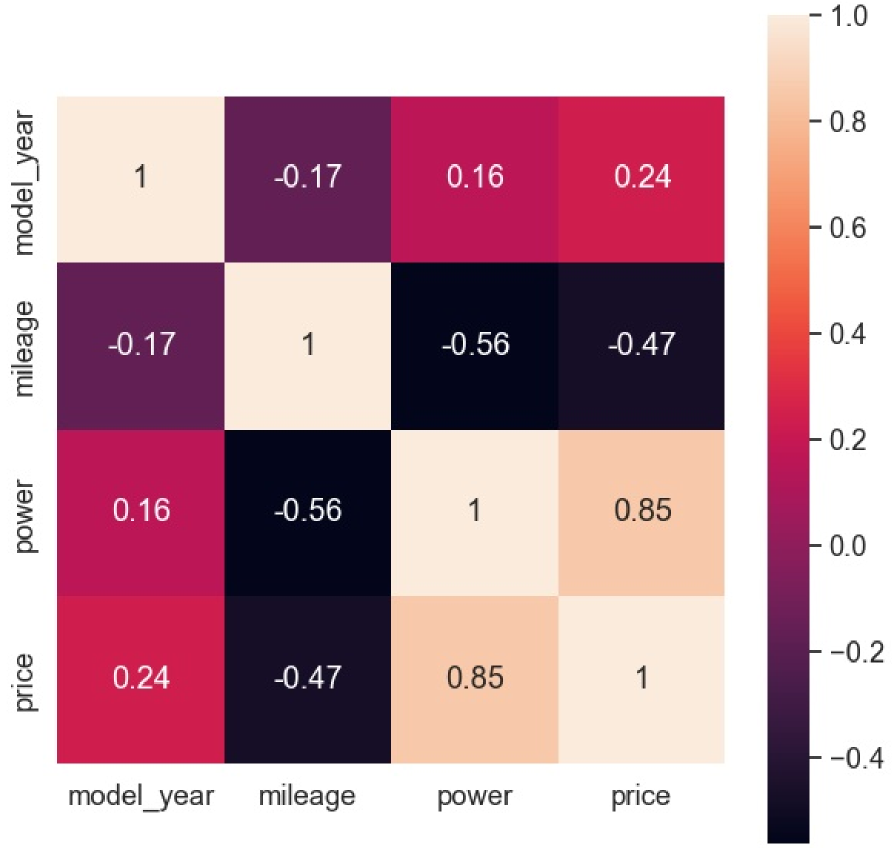
Out[29]: (7054, 8)

In [30]:
```
pd.DataFrame([bikes.isnull().sum(), bikes.isnull().sum() / bikes.shape[0] * 100], index=['count', '%']).T
```

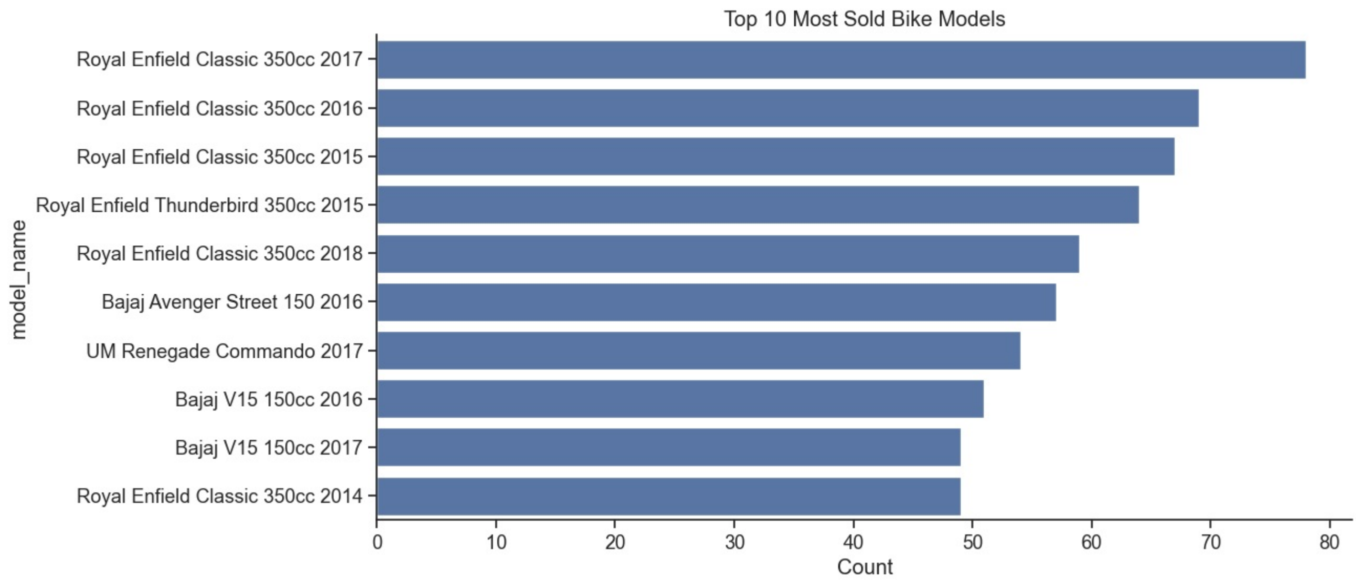
Out[30]:

	count	%
model_name	0.0	0.0
model_year	0.0	0.0
kms_driven	0.0	0.0
owner	0.0	0.0
location	0.0	0.0
mileage	0.0	0.0
power	0.0	0.0
price	0.0	0.0

In [31]:
```
plt.figure(figsize=(7,7))
sns.set(font_scale=1.2)

# Only select numeric columns for correlation
numeric_df = bikes.select_dtypes(include='number')
sns.heatmap(numeric_df.corr(), square=True, annot=True)

plt.show()
```

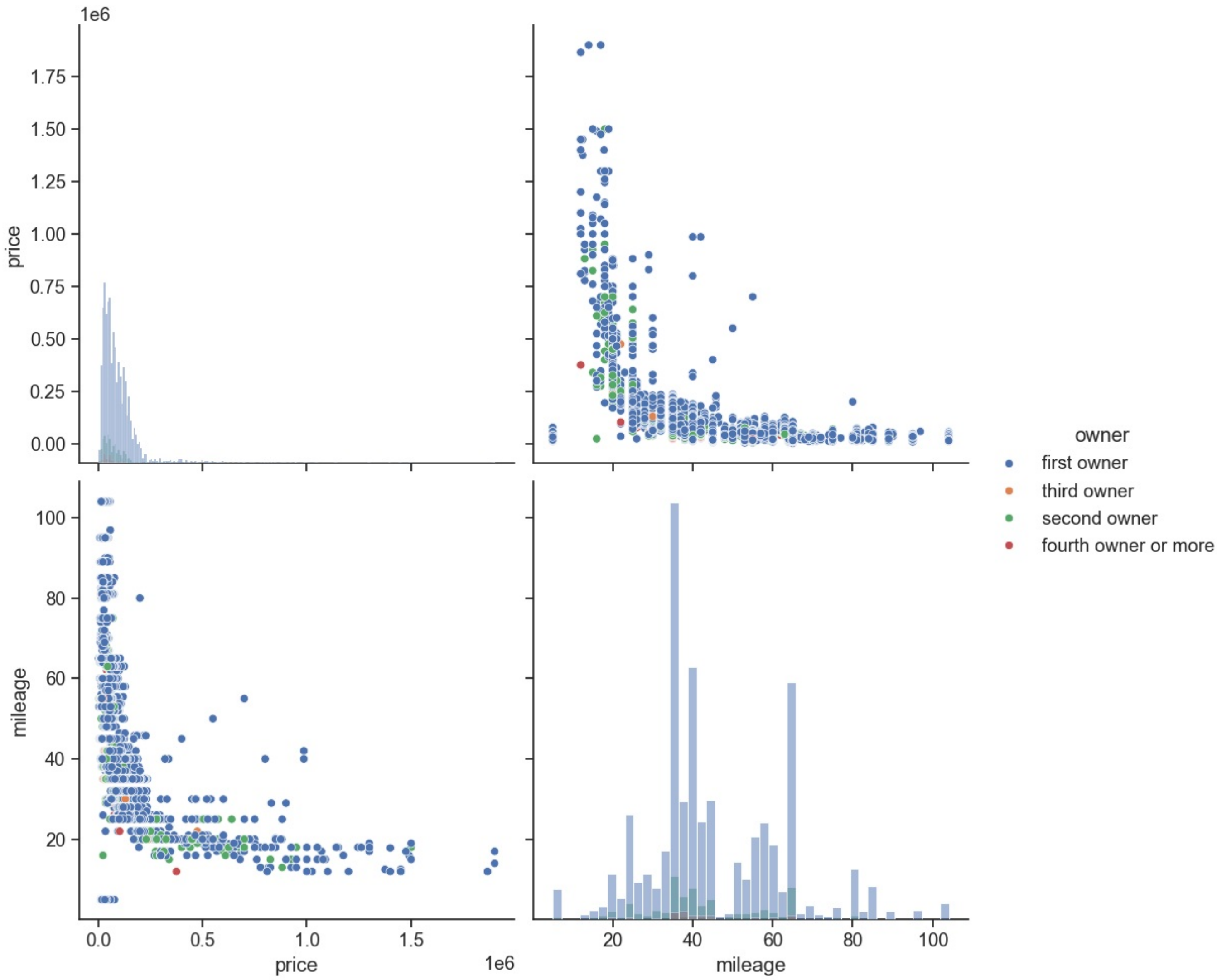


In [35]:
```
plt.figure(figsize=(12,6))
sns.set(style='ticks', font_scale=1.2)
sns.countplot(data=bikes, y='model_name', order=bikes['model_name'].value_counts().index[:10])
plt.title('Top 10 Most Sold Bike Models')
plt.xlabel('Count')
sns.despine()
plt.show()
```



In [36]:
```
# Suppose we want to analyze distribution of price by model_year and owner
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style='ticks', font_scale=1.2)
g = sns.FacetGrid(bikes, row="owner", col="model_year", margin_titles=True, despine=False)
g.map_dataframe(sns.histplot, x="price", bins=20)
g.figure.subplots_adjust(wspace=0.2, hspace=0.4)
g.add_legend()

# Rotate x-tick labels for clarity
for axes in g.axes.flat:
    _ = axes.set_xticklabels(axes.get_xticklabels(), rotation=45)

plt.show()
```

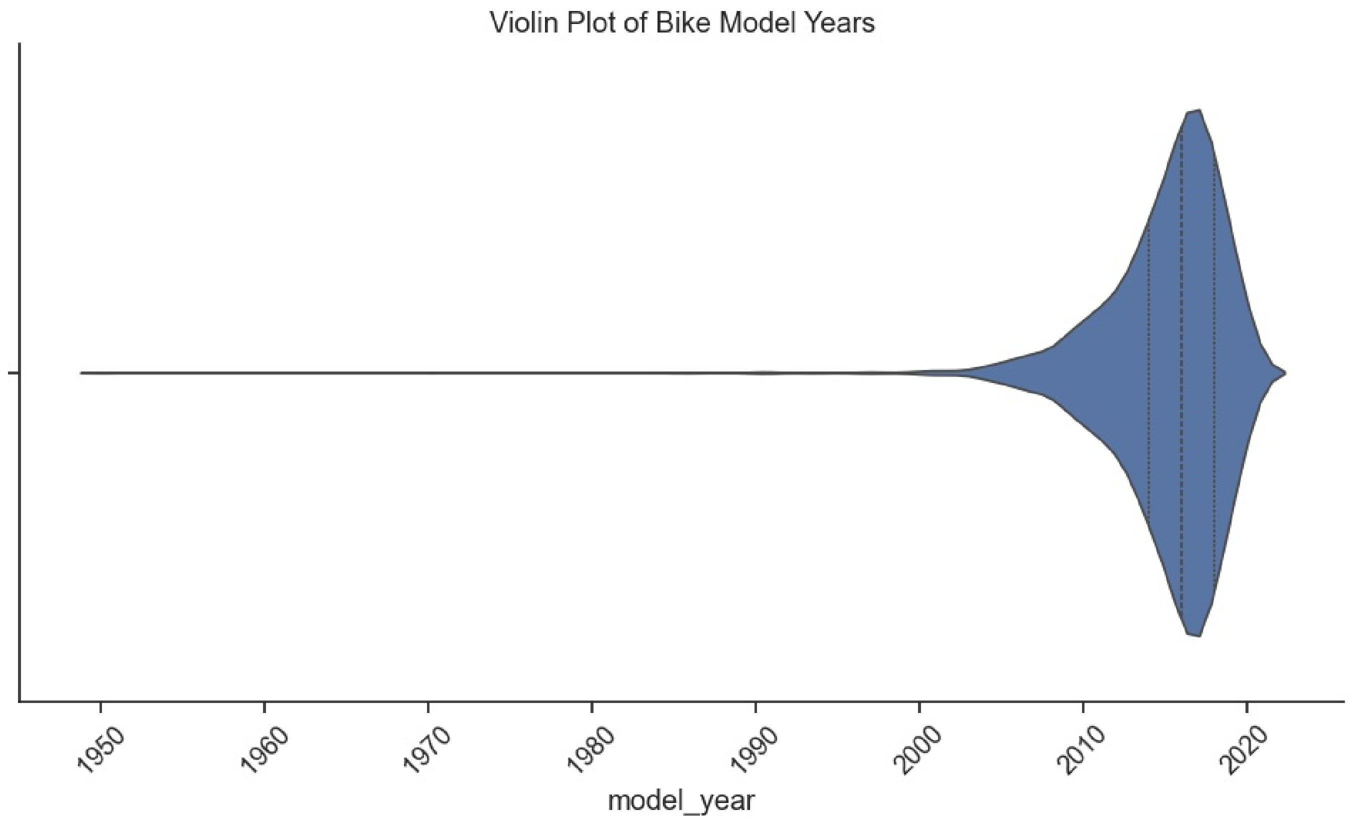
In [37]:
```python
# Make sure mileage is numeric
bikes['mileage'] = bikes['mileage'].astype(str).str.extract(r'(\d+\.?\d*)').astype(float)
bikes['price'] = pd.to_numeric(bikes['price'], errors='coerce')

# Drop NaNs for plotting
plot_data = bikes[['price', 'mileage', 'owner']].dropna()

sns.set(style='ticks', font_scale=1.2)
sns.pairplot(plot_data, diag_kind='hist', hue='owner', height=5, aspect=1)
plt.show()
```



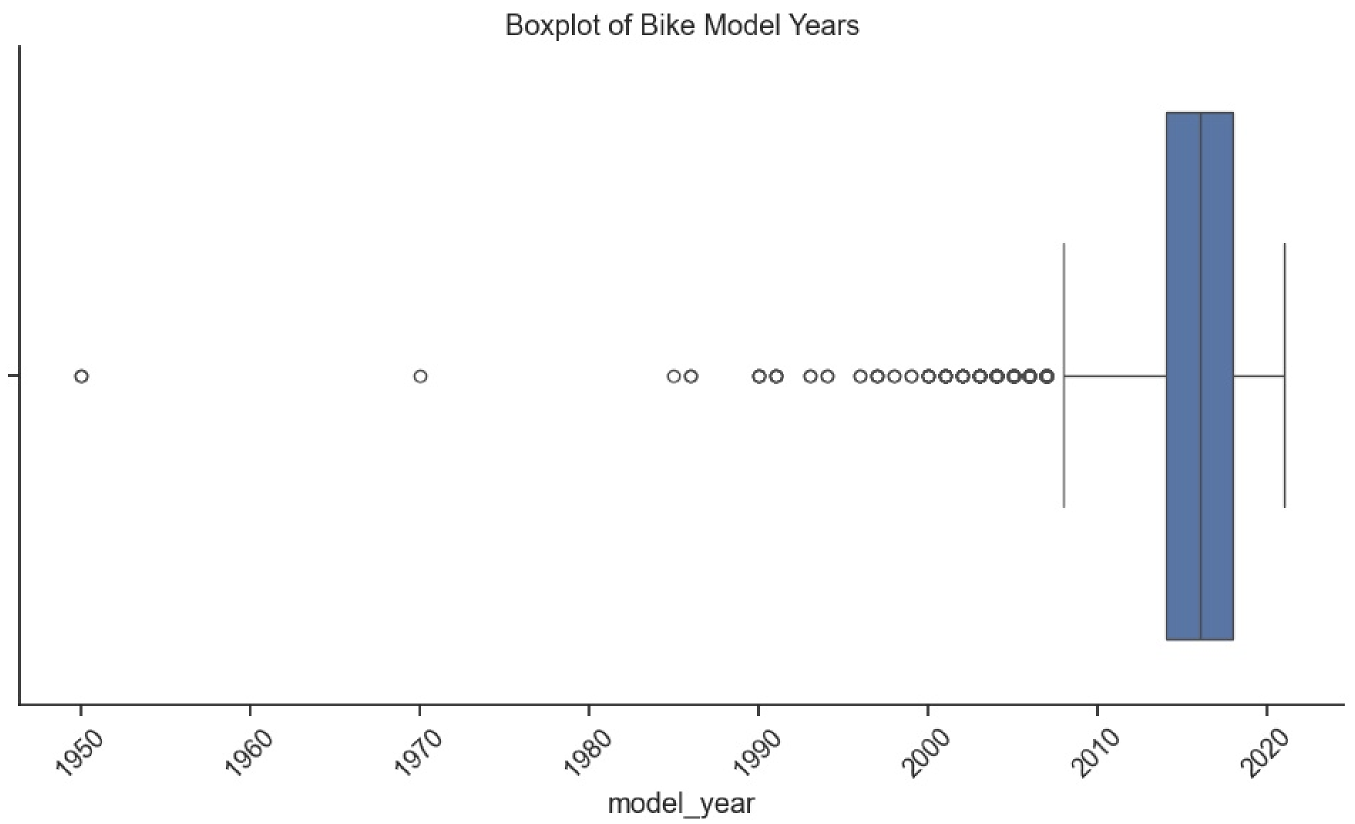
In [38]:
```python
bikes.loc[:,['price','mileage']].corr().iloc[1,0].round(2)
```

Out[38]:
```
np.float64(-0.47)
```

In [44]:
```python
# Violin plot to show the distribution of bikes across model years
plt.figure(figsize=(12, 6))
sns.set(style='ticks', font_scale=1.2)
plots = sns.violinplot(x='model_year', data=bikes, inner="quart", scale='area')
plots.set_title('Violin Plot of Bike Model Years')

# Rotate labels for better readability
plt.xticks(rotation=45)
sns.despine()
plt.show()
```

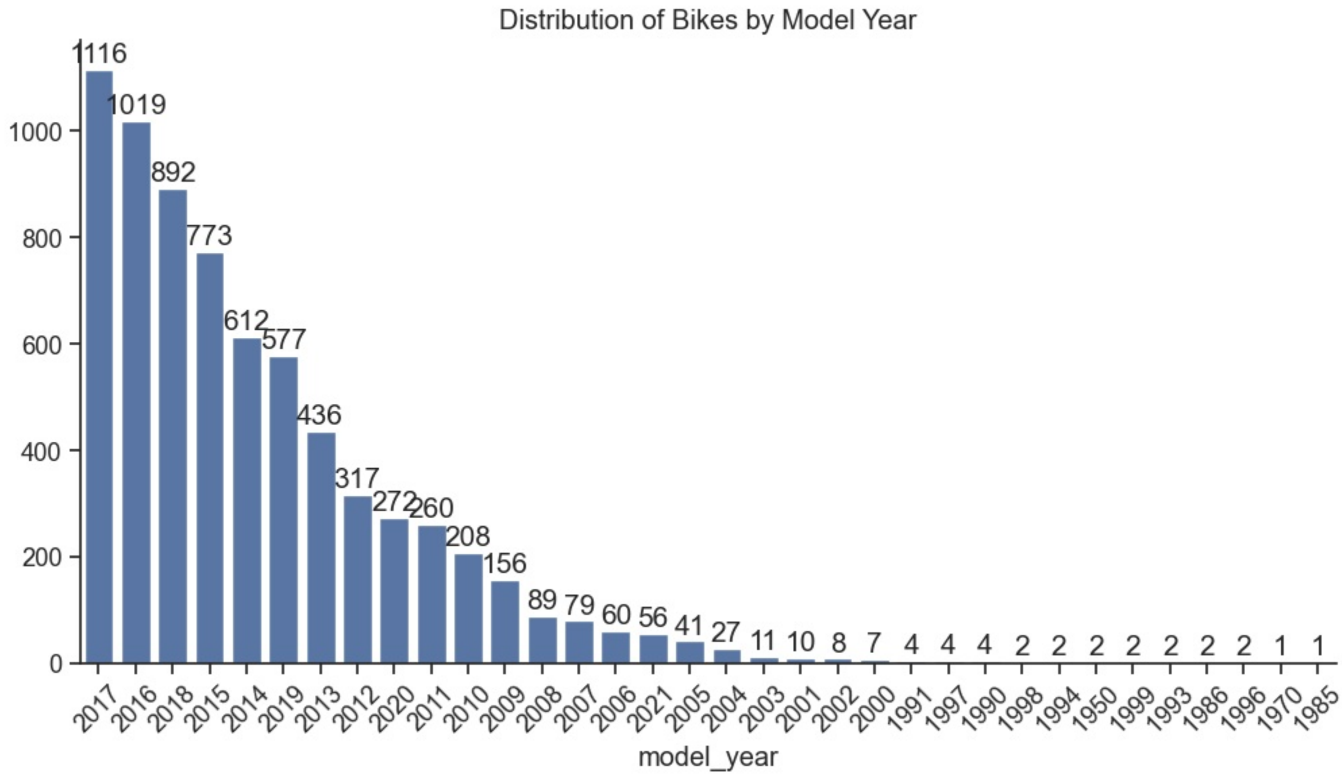


In [45]:
```
# Boxplot to show the distribution of model_year
plt.figure(figsize=(12, 6))
sns.set(style='ticks', font_scale=1.2)
plots = sns.boxplot(x='model_year', data=bikes)
plots.set_title('Boxplot of Bike Model Years')

# Rotate labels for better readability
plt.xticks(rotation=45)
sns.despine()
plt.show()
```



In [47]:
```
# Plotting the count of bikes by 'model_year' using a barplot
plt.figure(figsize=(12, 6))
sns.set(style='ticks', font_scale=1.2)
plots = sns.barplot(x=bikes['model_year'].astype(str).value_counts().index,
                    y=bikes['model_year'].value_counts().values)
plots.set_title('Distribution of Bikes by Model Year')
```

```python
# Annotate the bars with the count values
for bar in plots.patches:
    plots.annotate(str(format(bar.get_height(), '.0f')),
                   (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                   ha='center', va='center', size=15, xytext=(0, 8), textcoords='offset points')

sns.despine()
plt.xticks(rotation=45)  # Rotate labels if necessary
plt.show()
```



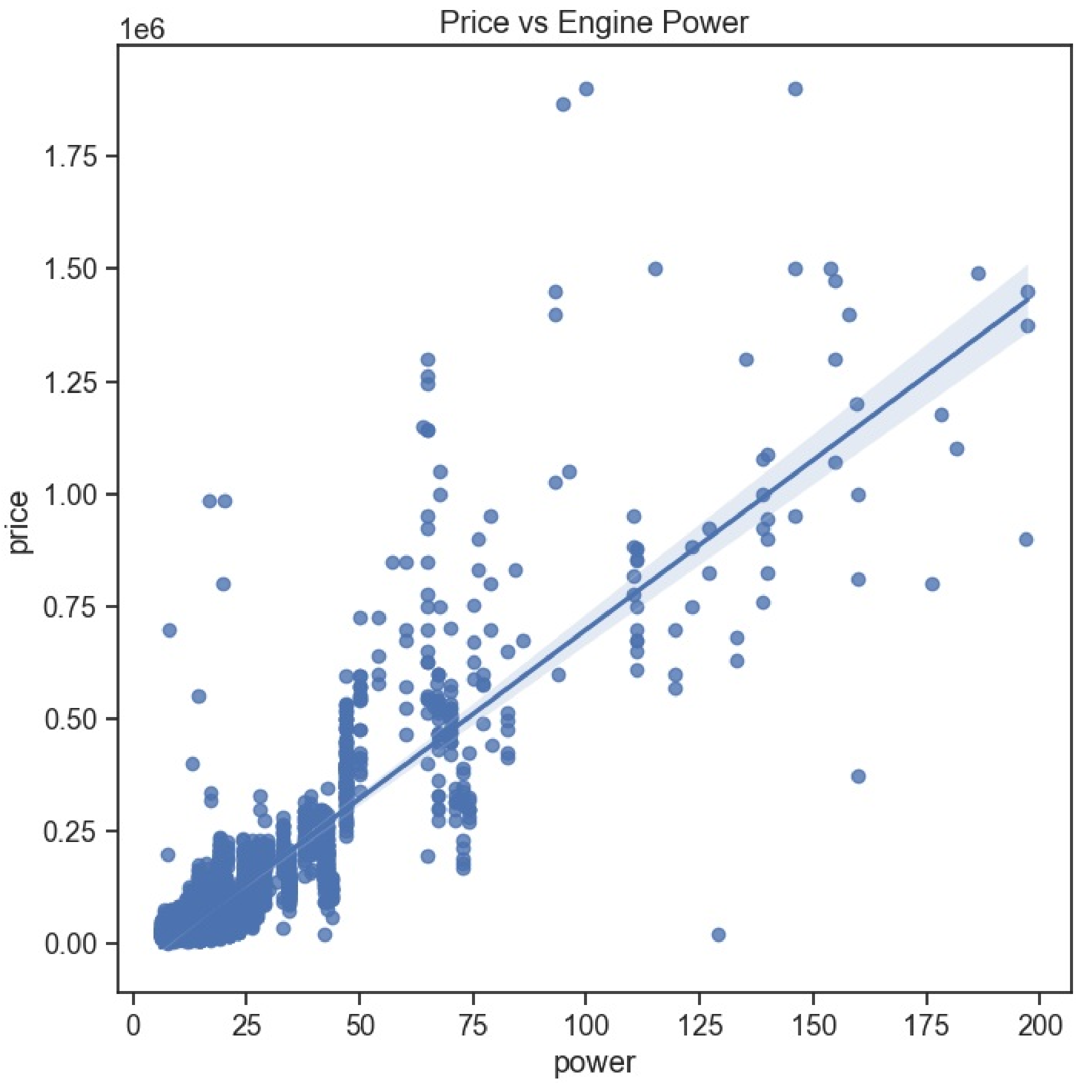
In [49]:
```python
bikes.dtypes
```

Out[49]:
```
model_name      object
model_year    category
kms_driven      object
owner           object
location        object
mileage        float64
power          float64
price            int64
dtype: object
```

In [50]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import patheffects

# Creating a UDF to add median labels
def add_median_labels(ax, fmt='.1f'):
    '''
    This function adds median labels at any orientation
    '''
    lines = ax.get_lines()
    boxes = [c for c in ax.get_children() if type(c).__name__ == 'PathPatch']
    lines_per_box = int(len(lines) / len(boxes))
    for median in lines[4:len(lines):lines_per_box]:
        x, y = (data.mean() for data in median.get_data())
        # choosing value depending on horizontal or vertical plot orientation
        value = x if (median.get_xdata()[1] - median.get_xdata()[0]) == 0 else y
        text = ax.text(x, y, f'{value:{fmt}}', ha='center', va='center',
                       fontweight='bold', color='white')
        # creating median-colored border around white text for contrast
        text.set_path_effects([
            patheffects.Stroke(linewidth=3, foreground=median.get_color()),
            patheffects.Normal(),
        ])

# Clean data: Remove NaNs in 'price' and 'owner' columns
bikes = bikes.dropna(subset=['price', 'owner'])

# Plotting the boxplot for 'price' vs 'owner'
plt.figure(figsize=(16,5))
sns.set(style='ticks', font_scale=1.2)
```

```
# Boxplot for 'price' vs 'owner'
ax = sns.boxplot(x='price', y='owner', saturation=1, data=bikes, showfliers=False)
ax.set_title('Price Distribution Between Owners (without Outliers)')

# Add median labels
add_median_labels(ax)

plt.show()
```

In [51]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting Price vs Engine Power (based on the 'power' column in the bikes dataset)
plt.figure(figsize=(8, 8))
sns.set(style='ticks', font_scale=1.2)

# Using 'power' column for engine value and plotting price
sns.regplot(x='power', y='price', data=bikes, x_jitter=0, y_jitter=0).set_title('Price vs Engine Power')

plt.show()
```



In [52]:
```
# Calculate the correlation between 'price' and 'power' (engine power)
correlation = bikes[['price', 'power']].corr().iloc[1, 0].round(2)
```

```
# Display the correlation
print(f"The correlation between price and engine power is: {correlation}")
```

```
The correlation between price and engine power is: 0.85
```

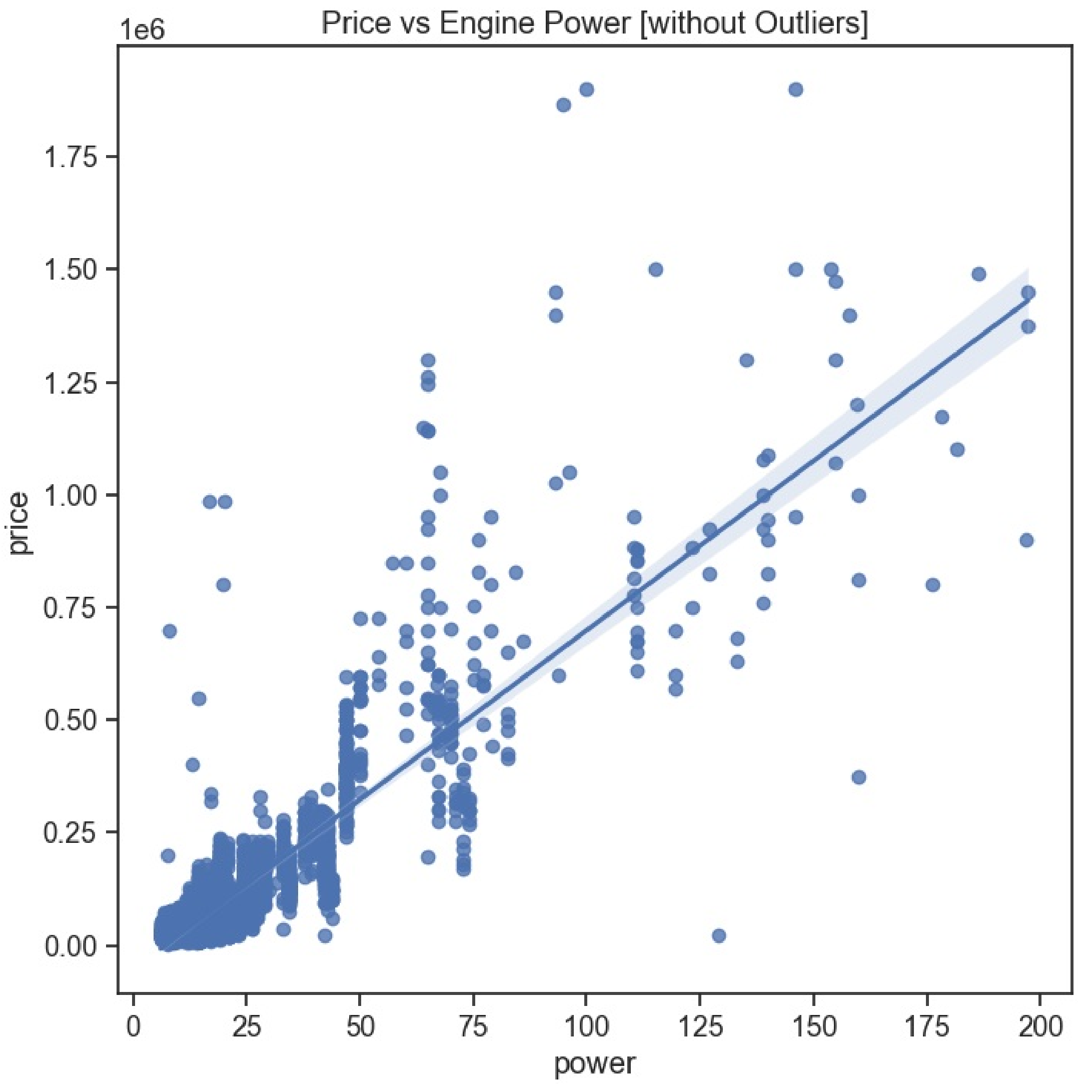
In [53]:

```
# Apply the clipping operation to the 'price' and 'power' columns based on the 1st and 99th percentiles
bikes1 = bikes.loc[:, ['price', 'power']].apply(
    lambda x: x.clip(lower=x.quantile(0.01), upper=x.quantile(0.99))
)

# Display the transformed bikes1 dataset
print(bikes1.head())
```

```
    price  power
0   63500   19.0
1  115000   19.8
2  300000   28.0
3  100000   34.5
5   63400   25.0
```

In [54]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting Price vs Engine Power (using 'power' instead of 'engV')
plt.figure(figsize=(8, 8))
sns.set(style='ticks', font_scale=1.2)
sns.regplot(x='power', y='price', data=bikes, x_jitter=0, y_jitter=0).set_title('Price vs Engine Power [without
plt.show()

# Calculate the correlation between 'price' and 'power' (engine power)
correlation = bikes.loc[:, ['price', 'power']].corr().iloc[1, 0].round(2)

# Display the updated correlation
print('Updated Corr Value between Price and Engine Power:', correlation)
```

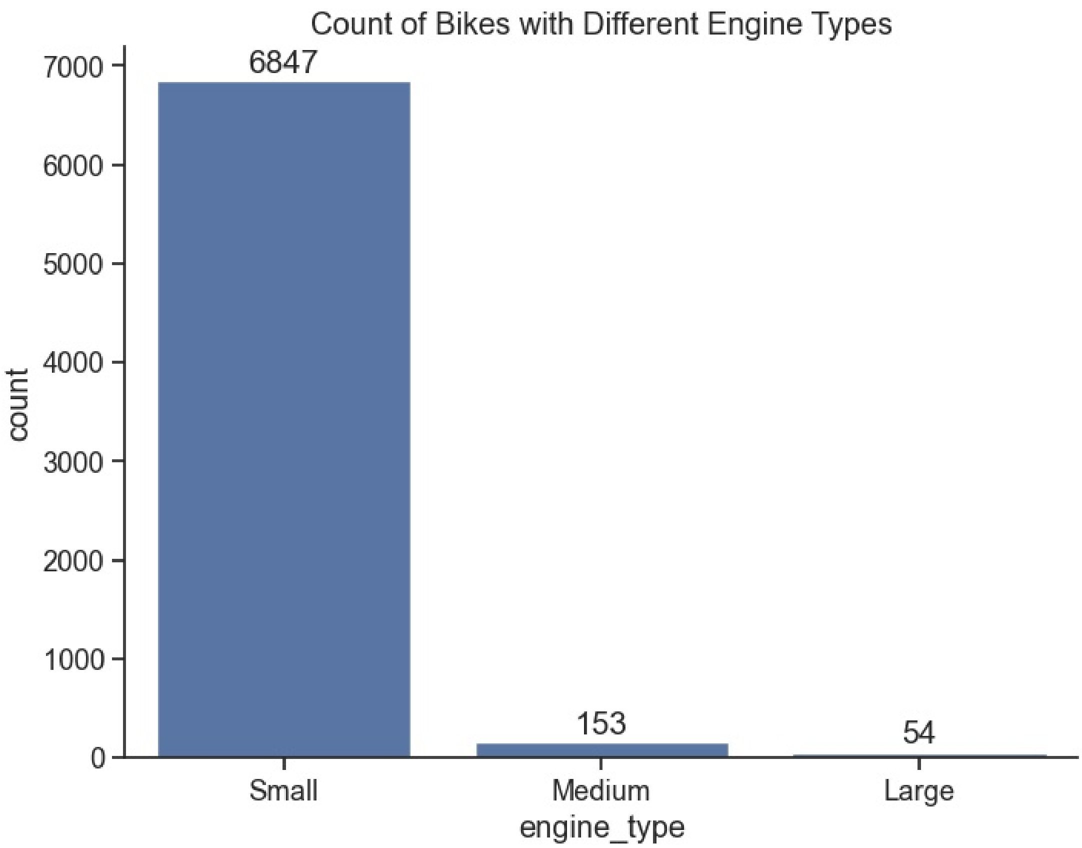



```
Updated Corr Value between Price and Engine Power: 0.85
```

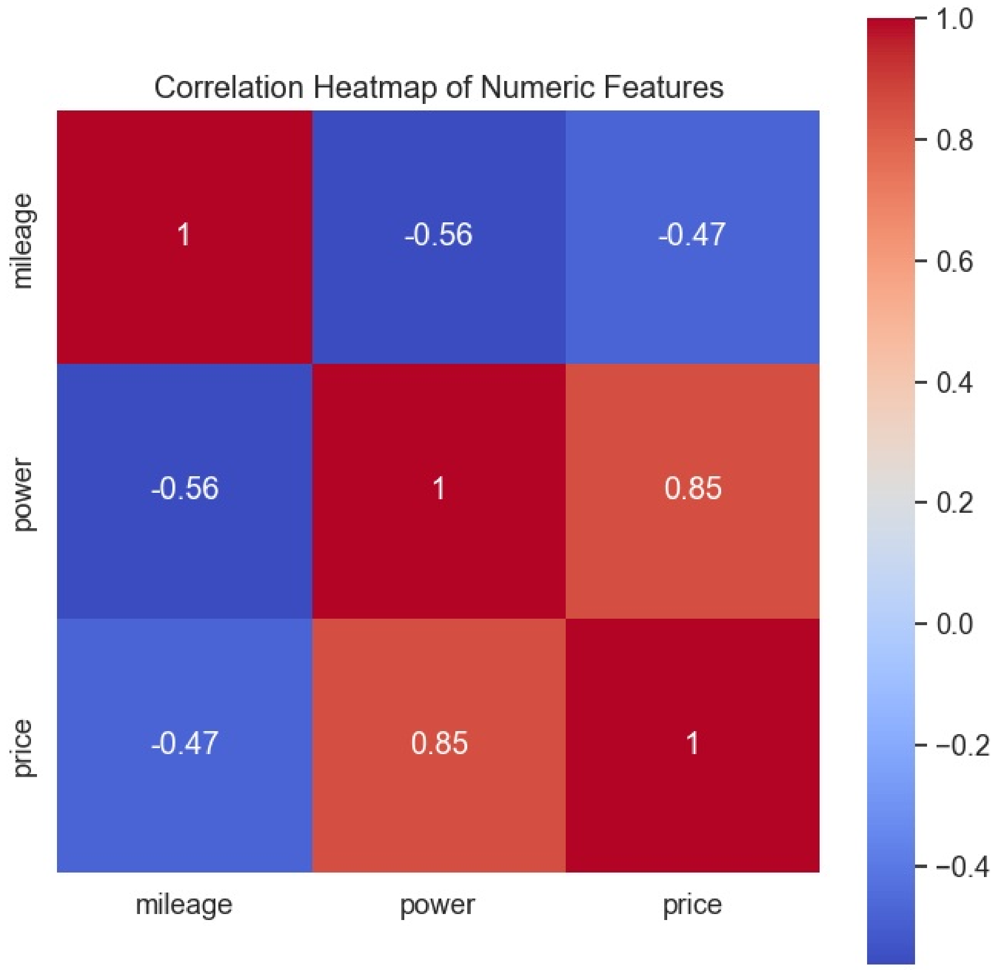
In [55]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

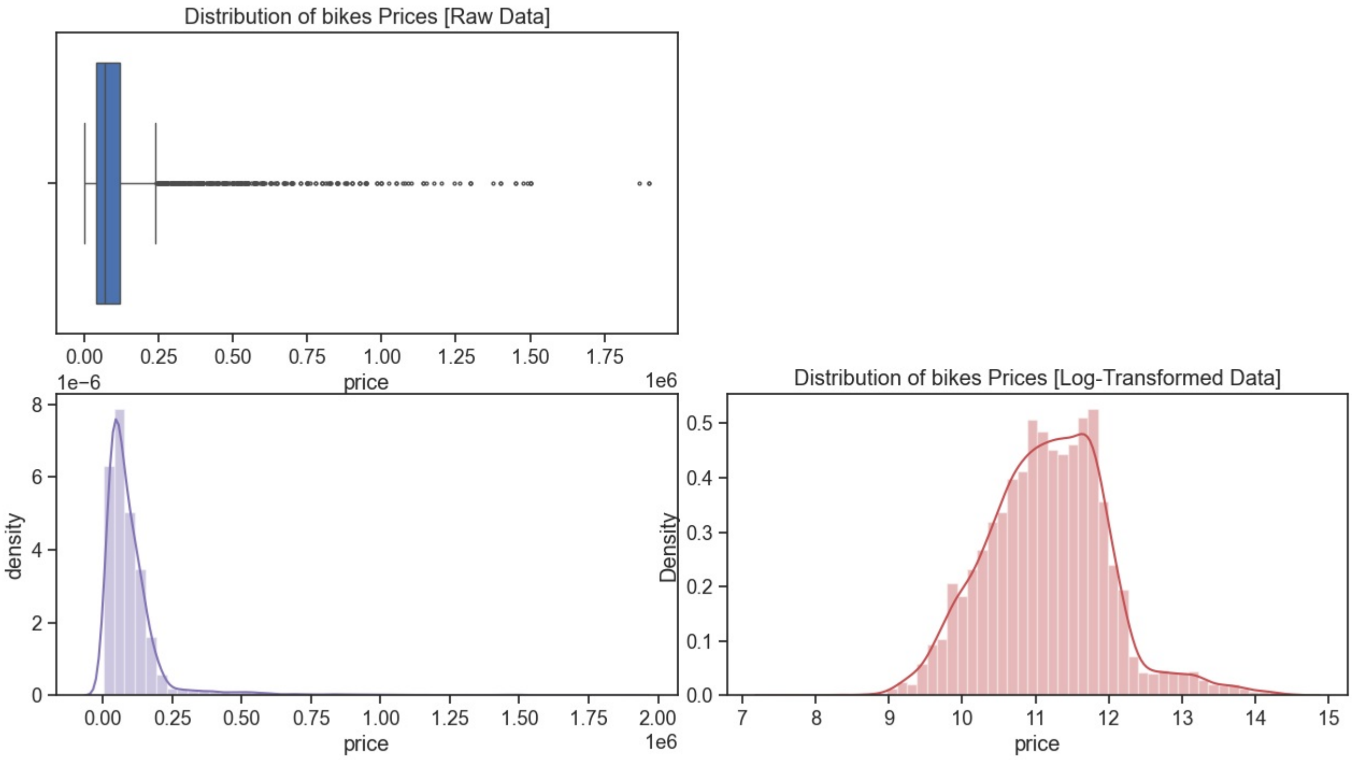
```python
# Create engine types based on 'power' values (for example: small, medium, large)
def categorize_engine(power):
    if power < 50:
        return 'Small'
    elif 50 <= power < 100:
        return 'Medium'
    else:
        return 'Large'

# Apply the categorization function to the 'power' column
bikes['engine_type'] = bikes['power'].apply(categorize_engine)

# Plotting the count of different engine types
plt.figure(figsize=(8, 6))
sns.set(style='ticks', font_scale=1.2)
plots = sns.countplot(x='engine_type', data=bikes, order=bikes['engine_type'].value_counts().index)
plots.set_title('Count of Bikes with Different Engine Types')

# Annotate the bars with the count values
for bar in plots.patches:
    plots.annotate(str(format(bar.get_height(), '.0f')),
                   (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                   ha='center', va='center', size=15, xytext=(0, 8), textcoords='offset points')

sns.despine()
plt.show()
```



In [56]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Select only numeric columns for correlation
numeric_bikes = bikes.select_dtypes(include='number')

plt.figure(figsize=(8,8))
sns.set(font_scale=1.2)
sns.heatmap(numeric_bikes.corr(), square=True, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Numeric Features')
plt.show()
```



In [58]:
```
params = {'figure.figsize': (15.5, 8), 'axes.labelsize': 14}
plt.rcParams.update(params)
sns.set(style='ticks', font_scale=1.2)
gs = gridspec.GridSpec(2, 2, wspace=0.08)
f = pl.figure()

# Boxplot
ax = pl.subplot(gs[0, 0]) # row 0, col 0
sns.boxplot(x='price', saturation=1, data=bikes, showfliers=True, ax=ax, fliersize=2) \
    .set_title('Distribution of bikes Prices [Raw Data]')

ax = pl.subplot(gs[0, 1]) # row 0, col 1
ax.axis('off')

# Distplot
ax = pl.subplot(gs[1, 0]) # row 1, col 0
ax = sns.distplot(bikes['price'],color ='m')
plt.ylabel('density')

# Logarithmic Distribution
ax = pl.subplot(gs[1, 1]) # row 1, col 1
ax = sns.distplot(np.log(bikes['price']),color ='r') \
    .set_title('Distribution of bikes Prices [Log-Transformed Data]')

plt.show()
```



In [59]:
```
bikes.price.skew()
```

Out[59]:
```
np.float64(5.671613309891151)
```

In [60]:
```
#After Outlier Treatment
params = {'figure.figsize': (15.5, 8), 'axes.labelsize': 14}
plt.rcParams.update(params)
sns.set(style='ticks', font_scale=1.2)
gs = gridspec.GridSpec(2, 2, wspace=0.08)
f = pl.figure()

# Boxplot without Outliers (p99)
ax = pl.subplot(gs[0, 0]) # row 0, col 0
sns.boxplot(x='price', saturation=1, data=bikes1, showfliers=True, ax=ax, fliersize=2) \
    .set_title('Distribution of bikes Prices [Outliers clipped at p99]')

ax = pl.subplot(gs[0, 1]) # row 0, col 1
ax.axis('off')

# Distplot without Outliers (p99)
ax = pl.subplot(gs[1, 0]) # row 1, col 0
ax = sns.distplot(bikes1['price'],color ='m')
plt.ylabel('density')

# Logarithmic Distribution without Outliers
ax = pl.subplot(gs[1, 1]) # row 1, col 1
ax = sns.distplot(np.log(bikes1['price']),color ='r') \
    .set_title('Distribution of Car Prices [Log-Transformed Data]')

plt.show()
```

In [61]:
```
print(bikes.describe())
```

```
           mileage        power         price
count  7054.000000  7054.000000  7.054000e+03
mean     44.790615    20.772818  1.028602e+05
std      16.970718    15.135571  1.332381e+05
min       5.000000     6.100000  2.000000e+03
25%      35.000000    13.800000  4.000000e+04
50%      40.000000    19.000000  7.000000e+04
75%      57.000000    24.160000  1.200000e+05
max     104.000000   197.300000  1.900000e+06
```

In [ ]: